Kostenloses eBook

LERNEN

dart

Free unaffiliated eBook created from
Stack Overflow contributors.

#dart

Inhaltsverzeichnis

Über

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

It is an unofficial and free dart 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 dart.

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

Kapitel 1: Erste Schritte mit Dart

Bemerkungen

Dart ist eine auf Open Source basierende, klassenbasierte, optional typisierte Programmiersprache zum Erstellen von Webanwendungen - sowohl auf dem Client als auch auf dem Server -, die von Google erstellt wurden. Darts Designziele sind:

- Erstellen Sie eine strukturierte und dennoch flexible Sprache für die Webprogrammierung.
- Sorgen Sie dafür, dass sich Dart den Programmierern vertraut und natürlich anfühlt und daher leicht zu erlernen ist.
- Stellen Sie sicher, dass Dart auf allen modernen Webbrowsern und Umgebungen, von kleinen Handheld-Geräten bis zur serverseitigen Ausführung, eine hohe Leistung bietet.

Dart zielt auf eine breite Palette von Entwicklungsszenarien ab, von einem Ein-Personen-Projekt ohne große Struktur bis zu einem großen Projekt, das formale Typen im Code benötigt, um die Absicht des Programmierers anzugeben.

Zur Unterstützung dieses breiten Spektrums an Projekten bietet Dart die folgenden Funktionen und Tools:

- **Optionale Typen:** Dies bedeutet, dass Sie die Codierung ohne Typen starten und sie bei Bedarf später hinzufügen können.
- **Isolate:** gleichzeitige Programmierung auf Server und Client
- **Einfacher DOM-Zugriff:** Verwendung von CSS-Selektoren (genau wie bei jQuery)
- **Dart IDE-Tools:** Dart-Plug-Ins gibt es für viele häufig verwendete IDEs, z. B. WebStorm .
- **Dartium:** Ein Build des Chromium-Webbrowsers mit einer integrierten virtuellen Dart-Maschine

Links

- Die Dart-Homepage
- Offizielle Dart News & Updates
- The Dartosphere - Eine Sammlung aktueller Dart-Blogbeiträge
- Dartisans Dartisans-Community auf Google+
- Dart Web Development - Google Groups-Seite
- Dart Language Misc - Google Groups-Seite
- DartLang Sub-Reddit

Dokumentation

- Tour der Dartsprache
- Besichtigung der Dart-Bibliotheken
- Dart-Code-Beispiele
- Dart-API-Referenz

FAQ

- Häufig gestellte Fragen

Versionen

Ausführung	Veröffentlichungsdatum
1.22.1	2017-02-22
1.22.0	2017-02-14
1.21.1	2016-01-13
1.21.0	2016-12-07
1.20.1	2016-10-13
1,20,0	2016-10-11
1.19.1	2016-09-07
1.19.0	2016-08-26
1.18.1	2016-08-02
1.18.0	2016-07-27
1.17.1	2016-06-10
1.17.0	2016-06-06
1.16.1	2016-05-23
1.16.0	2016-04-26
1.15.0	2016-03-09
1.14.2	2016-02-09
1.14.1	2016-02-03
1.14.0	2016-01-28

Ausführung	Veröffentlichungsdatum
1.13.2	2016-01-05
1.13.1	2015-12-17
1.13.0	2015-11-18
1.12.2	2015-10-21
1.12.1	2015-09-08
1.12.0	2015-08-31
1.11.3	2015-08-03
1.11.1	2015-07-02
1.11.0	2015-06-24
1.10.1	2015-05-11
1.10.0	2015-04-24
1.9.3	2015-04-13
1.9.1	2015-03-25
1.8.5	2015-01-13
1.8.3	2014-12-01
1.8.0	2014-11-27
1.7.2	2014-10-14
1.6.0	2014-08-27
1.5.8	2014-07-29
1.5.3	2014-07-03
1.5.2	2014-07-02
1.5.1	2014-06-24
1.4.3	2014-06-16
1.4.2	2014-05-27
1.4.0	2014-05-20

Ausführung	Veröffentlichungsdatum
1.3.6	2014-04-30
1.3.3	2014-04-16
1.3.0	2014-04-08
1.2.0	2014-02-25
1.1.3	2014-02-06
1.1.1	2014-01-15
1.0.0.10_r30798	2013-12-02
1.0.0.3_r30188	2013-11-12
0.8.10.10_r30107	2013-11-08
0.8.10.6_r30036	2013-11-07
0.8.10.3_r29803	2013-11-04

Examples

Installation oder Setup

Das Dart-SDK enthält alles, was Sie zum Schreiben und Ausführen von Dart-Code benötigen: VM, Bibliotheken, Analyzer, Paketmanager, Dokumentgenerator, Formatierer, Debugger und mehr. Wenn Sie Webentwicklung betreiben, benötigen Sie auch Dartium.

Automatisierte Installation und Updates

- Dart unter Windows installieren
- Dart auf dem Mac installieren
- Dart unter Linux installieren

Manuelle Installation

Sie können auch jede Version des SDK manuell installieren .

Hallo Welt!

Erstellen Sie eine neue Datei mit dem Namen `hello_world.dart` mit folgendem Inhalt:

```
void main() {
```

```
  print('Hello, World!');
}
```

Navigieren Sie im Terminal zu dem Verzeichnis, in dem sich die Datei `hello_world.dart` und geben Sie Folgendes ein:

```
dart hello_world.dart
```

Drücken Sie die `Eingabetaste` , um `Hello, World!` anzuzeigen. im Terminalfenster.

HTTP-Request

Html

```
<img id="cats"></img>
```

Pfeil

```
import 'dart:html';

/// Stores the image in [blob] in the [ImageElement] of the given [selector].
void setImage(selector, blob) {
  FileReader reader = new FileReader();
  reader.onLoad.listen((fe) {
    ImageElement image = document.querySelector(selector);
    image.src = reader.result;
  });
  reader.readAsDataUrl(blob);
}

main() async {
  var url = "https://upload.wikimedia.org/wikipedia/commons/2/28/Tortoiseshell_she-cat.JPG";

  // Initiates a request and asynchronously waits for the result.
  var request = await HttpRequest.request(url, responseType: 'blob');
  var blob = request.response;
  setImage("#cats", blob);
}
```

Beispiel

siehe Beispiel auf https://dartpad.dartlang.org/a0e092983f63a40b0b716989cac6969a

Getter und Setter

```
void main() {
  var cat = new Cat();

  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? true
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? false
```

```
  print("Feed cat.");
  cat.isHungry = false;
  print("Is cat hungry? ${cat.isHungry}");  // Is cat hungry? false
  print("Is cat cuddly? ${cat.isCuddly}");  // Is cat cuddly? true
}

class Cat {
  bool _isHungry = true;

  bool get isCuddly => !_isHungry;

  bool get isHungry => _isHungry;
  bool set isHungry(bool hungry) => this._isHungry = hungry;
}
```

Dart- Klassen-Getter und -Setter ermöglichen APIs, Änderungen des Objektzustands zu kapseln.

Das Dartpad- Beispiel finden Sie hier:
https://dartpad.dartlang.org/c25af60ca18a192b84af6990f3313233

Erste Schritte mit Dart online lesen: https://riptutorial.com/de/dart/topic/843/erste-schritte-mit-dart

Kapitel 2: Asynchrone Programmierung

Examples

Eine Zukunft mit einem Completer zurückgeben

```
Future<Results> costlyQuery() {
  var completer = new Completer();

  database.query("SELECT * FROM giant_table", (results) {
    // when complete
    completer.complete(results);
  }, (error) {
    completer.completeException(error);
  });

  // this returns essentially immediately,
  // before query is finished
  return completer.future;
}
```

Async und Await

```
import 'dart:async';

Future main() async {
  var value = await _waitForValue();
  print("Here is the value: $value");
  //since _waitForValue() returns immediately if you un it without await you won't get the
result
  var errorValue = "not finished yet";
  _waitForValue();
  print("Here is the error value: $value");// not finished yet
}

Future<int> _waitForValue() => new Future((){

  var n = 100000000;

  // Do some long process
  for (var i = 1; i <= n; i++) {
    // Print out progress:
    if ([n / 2, n / 4, n / 10, n / 20].contains(i)) {
      print("Not done yet...");
    }

    // Return value when done.
    if (i == n) {
      print("Done.");
      return i;
    }
  }
});
```

Siehe Beispiel auf Dartpad: https://dartpad.dartlang.org/11d189b51e0f2680793ab3e16e53613c

Rückrufe in Futures umrechnen

Dart verfügt über eine robuste async-Bibliothek mit Future , Stream und mehr. Manchmal laufen jedoch möglicherweise asynchrone APIs, die *Callbacks* anstelle von *Futures verwenden* . Um die Lücke zwischen Callbacks und Futures zu schließen, bietet Dart die *Completer*- Klasse an. Sie können einen Completer verwenden, um einen Rückruf in eine Zukunft umzuwandeln.

Completers eignen sich hervorragend zum Überbrücken einer Callback-basierten API mit einer Future-basierten API. Angenommen, Ihr Datenbanktreiber verwendet keine Futures, Sie müssen jedoch eine Zukunft zurückgeben. Versuchen Sie diesen Code:

```
// A good use of a Completer.

Future doStuff() {
  Completer completer = new Completer();
  runDatabaseQuery(sql, (results) {
    completer.complete(results);
  });
  return completer.future;
}
```

Wenn Sie eine API verwenden, die bereits eine Zukunft zurückgibt, müssen Sie keinen Completer verwenden.

Asynchrone Programmierung online lesen: https://riptutorial.com/de/dart/topic/2520/asynchrone-programmierung

Kapitel 3: Aufzählungen

Examples

Grundlegende Verwendung

```
enum Fruit {
  apple, banana
}

main() {
  var a = Fruit.apple;
  switch (a) {
    case Fruit.apple:
      print('it is an apple');
      break;
  }

  // get all the values of the enums
  for (List<Fruit> value in Fruit.values) {
    print(value);
  }

  // get the second value
  print(Fruit.values[1]);
}
```

Aufzählungen online lesen: https://riptutorial.com/de/dart/topic/5107/aufzahlungen

Kapitel 4: Ausnahmen

Bemerkungen

Dartcode kann Ausnahmen werfen und abfangen. Ausnahmen sind Fehler, die darauf hinweisen, dass etwas Unerwartetes passiert ist. Wenn die Ausnahme nicht abgefangen wird, wird das Isolat, das die Ausnahme ausgelöst hat, ausgesetzt, und normalerweise werden das Isolat und sein Programm beendet.

Im Gegensatz zu Java sind alle Dart-Ausnahmen ungeprüfte Ausnahmen. Methoden geben nicht an, welche Ausnahmen sie auslösen könnten, und Sie müssen keine Ausnahmen abfangen.

Dart bietet Exception und Fehlertypen sowie zahlreiche vordefinierte Subtypen. Sie können natürlich Ihre eigenen Ausnahmen definieren. Dart-Programme können jedoch jedes Nicht-NULL-Objekt - nicht nur Exception- und Error-Objekte - als Ausnahme auslösen.

Examples

Benutzerdefinierte Ausnahme

```
class CustomException implements Exception {
  String cause;
  CustomException(this.cause);
}

void main() {
  try {
    throwException();
  } on CustomException {
    print("custom exception is been obtained");
  }
}

throwException() {
  throw new CustomException('This is my first custom exception');
}
```

Ausnahmen online lesen: https://riptutorial.com/de/dart/topic/3334/ausnahmen

Kapitel 5: Bemerkungen

Syntax

- // Einzeiliger Kommentar
- / * Mehrzeilig / Inline-Kommentar * /
- /// Dartdoc-Kommentar

Bemerkungen

Es ist empfehlenswert, Ihrem Code Kommentare hinzuzufügen, um zu erklären, warum etwas getan wird, oder um zu erklären, was etwas tut. Dies hilft zukünftigen Lesern Ihres Codes, Ihren Code leichter zu verstehen.

Verwandte Themen nicht bei StackOverflow:

- Effektiver Dart: Dokumentation

Examples

Kommentar zum Zeilenende

Alles rechts von `//` in derselben Zeile wird kommentiert.

```
int i = 0; // Commented out text
```

Mehrzeiliger Kommentar

Alles zwischen `/*` und `*/` ist kommentiert.

```
void main() {
  for (int i = 0; i < 5; i++) {
    /* This is commented, and
    will not affect code */
    print('hello ${i + 1}');
  }
}
```

Dokumentation mit Dartdoc

Wenn Sie einen Dokumentenkommentar anstelle eines regulären Kommentars verwenden, kann dartdoc diesen finden und eine Dokumentation dafür erstellen .

```
/// The number of characters in this chunk when unsplit.
int get length => ...
```

Sie können die meisten Markdown- Formatierungen in Ihren Doc-Kommentaren verwenden. Dartdoc verarbeitet sie entsprechend mit dem Markdown-Paket .

```
/// This is a paragraph of regular text.
///
/// This sentence has *two* _emphasized_ words (i.e. italics) and **two**
/// __strong__ ones (bold).
///
/// A blank line creates another separate paragraph. It has some `inline code`
/// delimited using backticks.
///
/// * Unordered lists.
/// * Look like ASCII bullet lists.
/// * You can also use `-` or `+`.
///
/// Links can be:
///
/// * http://www.just-a-bare-url.com
/// * [with the URL inline](http://google.com)
/// * [or separated out][ref link]
///
/// [ref link]: http://google.com
///
/// # A Header
///
/// ## A subheader
```

Bemerkungen online lesen: https://riptutorial.com/de/dart/topic/2436/bemerkungen

Kapitel 6: Bibliotheken

Bemerkungen

Mithilfe der `import` und `library` können Sie eine modulare und gemeinsam nutzbare Codebasis erstellen. Jede Dart-App ist eine `library` , auch wenn keine Bibliotheksanweisung verwendet wird. Bibliotheken können mit Paketen verteilt werden. Informationen zu pub, einem im SDK enthaltenen Paketmanager, finden Sie unter Pub Package und Asset Manager .

Examples

Bibliotheken verwenden

Verwenden Sie `import` um anzugeben, wie ein Namespace aus einer Bibliothek im Bereich einer anderen Bibliothek verwendet wird.

```
import 'dart:html';
```

Das einzige zu `import` Argument ist ein URI, der die Bibliothek angibt. Für integrierte Bibliotheken verfügt der URI über das spezielle `dart:` Schema. Für andere Bibliotheken können Sie einen Dateisystempfad oder das `package:` Schema verwenden. Das `package:` Schema gibt Bibliotheken an, die von einem Paketmanager wie dem Pub-Tool bereitgestellt werden. Zum Beispiel:

```
import 'dart:io';
import 'package:mylib/mylib.dart';
import 'package:utils/utils.dart';
```

Bibliotheken und Sichtbarkeit

Im Gegensatz zu Java verfügt Dart nicht über die Schlüsselwörter `public` , `protected` und private. Wenn ein Bezeichner mit einem Unterstrich _ beginnt, ist er für seine Bibliothek privat.

Wenn Sie beispielsweise Klasse A in einer separaten Bibliotheksdatei (z. B. `other.dart`) haben, z.

```
library other;

class A {
  int _private = 0;

  testA() {
    print('int value: $_private');  // 0
    _private = 5;
    print('int value: $_private'); // 5
  }
}
```

und importieren Sie es dann in Ihre Hauptanwendung, beispielsweise:

```
import 'other.dart';

void main() {
  var b = new B();
  b.testB();
}

class B extends A {
  String _private;

  testB() {
    _private = 'Hello';
    print('String value: $_private'); // Hello
    testA();
    print('String value: $_private'); // Hello
  }
}
```

Sie erhalten die erwartete Ausgabe:

```
String value: Hello
int value: 0
int value: 5
String value: Hello
```

Bibliothekspräfix angeben

Wenn Sie zwei Bibliotheken mit widersprüchlichen Bezeichnern importieren, können Sie ein Präfix für eine oder beide Bibliotheken angeben. Wenn beispielsweise library1 und library2 eine Element-Klasse haben, haben Sie möglicherweise folgenden Code:

```
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2;
// ...
var element1 = new Element(); // Uses Element from lib1.
var element2 =
    new lib2.Element();       // Uses Element from lib2.
```

Importieren nur eines Teils einer Bibliothek

Wenn Sie nur einen Teil einer Bibliothek verwenden möchten, können Sie die Bibliothek selektiv importieren. Zum Beispiel:

```
// Import only foo and bar.
import 'package:lib1/lib1.dart' show foo, bar;

// Import all names EXCEPT foo.
import 'package:lib2/lib2.dart' hide foo;
```

Lazy Laden einer Bibliothek

Beim verzögerten Laden (auch als verzögertes Laden bezeichnet) kann eine Anwendung eine Bibliothek bei Bedarf laden, wenn dies erforderlich ist. Um eine Bibliothek träge zu laden, müssen

Sie sie zuerst mit verzögert als importieren.

```
import 'package:deferred/hello.dart' deferred as hello;
```

Wenn Sie die Bibliothek benötigen, rufen Sie loadLibrary () mit der Kennung der Bibliothek auf.

```
greet() async {
  await hello.loadLibrary();
  hello.printGreeting();
}
```

In dem obigen Code, die `await` Schlüsselwort Pausen Ausführung , bis die Bibliothek geladen wird. Weitere Informationen zu `async` und zu `await async` finden Sie in weiteren Beispielen hier: Asynchronisierungsunterstützung oder im Abschnitt zur Asynchronisationsunterstützung der Sprachtour.

Bibliotheken online lesen: https://riptutorial.com/de/dart/topic/3332/bibliotheken

Kapitel 7: Dart-JavaScript-Interoperabilität

Einführung

Durch die Dart-JavaScript-Interoperabilität können wir JavaScript-Code aus unseren Dart-Programmen ausführen.

Die Interoperabilität wird durch die Verwendung der `js` Bibliothek zum Erstellen von Dart-Stubs erreicht. Diese Stubs beschreiben die Schnittstelle, die wir mit dem zugrunde liegenden JavaScript-Code haben möchten. Zur Laufzeit ruft der Dart-Stub den JavaScript-Code auf.

Examples

Eine globale Funktion aufrufen

Angenommen, wir `JSON.stringify` die JavaScript-Funktion `JSON.stringify` die ein Objekt empfängt, in einen JSON-String codiert und es zurückgibt.

Wir müssen nur die Funktionssignatur schreiben, als extern kennzeichnen und mit der `@JS` Anmerkung versehen:

```
 @JS("JSON.stringify")
external String stringify(obj);
```

Die `@JS` Annotation wird ab jetzt verwendet, um Dart-Klassen zu markieren, die wir auch in JavaScript verwenden `@JS` .

Umbrechen von JavaScript-Klassen / Namespaces

Angenommen, wir `google.maps` die Google Maps JavaScript-API `google.maps` :

```
@JS('google.maps')
library maps;

import "package:js/js.dart";

@JS()
class Map {
  external Map(Location location);
  external Location getLocation();
}
```

Wir haben jetzt die Map Dart-Klasse, die der JavaScript-Klasse `google.maps.Map` .

Das Ausführen einer `new Map(someLocation)` in Dart `new google.maps.Map(location)` die `new google.maps.Map(location)` in JavaScript auf.

Beachten Sie, dass Sie Ihre Dart-Klasse nicht wie die JavaScript-Klasse benennen müssen:

```
@JS("LatLng")
class Location {
  external Location(num lat, num lng);
}
```

Die `Location` Dart-Klasse entspricht der `google.maps.LatLng` Klasse.

Von der Verwendung inkonsistenter Namen wird abgeraten, da dies zu Verwirrung führen kann.

Objektliterale übergeben

In JavaScript ist es üblich, Objektliterale an Funktionen zu übergeben:

```
// JavaScript
printOptions({responsive: true});
Unfortunately we cannot pass Dart Map objects to JavaScript in these cases.
```

Wir müssen ein Dart-Objekt erstellen, das das Objektliteral darstellt und alle seine Felder enthält:

```
// Dart
@JS()
@anonymous
class Options {
  external bool get responsive;

  external factory Options({bool responsive});
}
```

Beachten Sie, dass die `Options` Dart-Klasse keiner JavaScript-Klasse entspricht. Als solches müssen wir es mit der `@anonymous` Anmerkung `@anonymous` .

Jetzt können wir einen Stub für die ursprüngliche printOptions-Funktion erstellen und ihn mit einem neuen Options-Objekt aufrufen:

```
// Dart
@JS()
external printOptions(Options options);

printOptions(new Options(responsive: true));
```

Dart-JavaScript-Interoperabilität online lesen: https://riptutorial.com/de/dart/topic/9240/dart-javascript-interoperabilitat

Kapitel 8: Daten konvertieren

Examples

JSON

```
import 'dart:convert';

void main() {
    var jsonString = """
      {
        "cats": {
                "abysinnian": {
                    "origin": "Burma",
                    "behavior": "playful"
                }
            }
    }
  """;

  var obj = JSON.decode(jsonString);

  print(obj['cats']['abysinnian']['behavior']); // playful
}
```

Siehe Beispiel zu Dartpad: https://dartpad.dartlang.org/7d5958cf10e611b36326f27b062108fe

Daten konvertieren online lesen: https://riptutorial.com/de/dart/topic/2778/daten-konvertieren

Kapitel 9: Datum und Uhrzeit

Examples

Grundlegende Verwendung von DateTime

```
DateTime now = new DateTime.now();
DateTime berlinWallFell = new DateTime(1989, 11, 9);
DateTime moonLanding = DateTime.parse("1969-07-20 20:18:00");  // 8:18pm
```

Weitere Informationen finden Sie hier .

Datum und Uhrzeit online lesen: https://riptutorial.com/de/dart/topic/3322/datum-und-uhrzeit

Kapitel 10: Filter auflisten

Einführung

Dart filtert Listen durch die Methoden `List.where` und `List.retainWhere` . Die `where` Funktion akzeptiert ein Argument: eine boolesche Funktion, die auf jedes Element der Liste angewendet wird. Wenn die Funktion als `true` ausgewertet wird, bleibt das Listenelement erhalten. Wenn die Funktion den Wert `false` ergibt, wird das Element entfernt.

Das Aufrufen von `theList.retainWhere(foo)` ist praktisch gleichbedeutend mit dem Setzen von `theList = theList.where(foo)` .

Examples

Filtern einer Liste von Ganzzahlen

```
[-1, 0, 2, 4, 7, 9].where((x) => x > 2) --> [4, 7, 9]
```

Filter auflisten online lesen: https://riptutorial.com/de/dart/topic/10948/filter-auflisten

Kapitel 11: Funktionen

Bemerkungen

Dart ist eine echte objektorientierte Sprache, daher sind selbst Funktionen Objekte vom Typ Function. Das heißt, Funktionen können Variablen zugewiesen oder anderen Funktionen als Argumente übergeben werden. Sie können eine Instanz einer Dart-Klasse auch so aufrufen, als wäre sie eine Funktion.

Examples

Funktionen mit benannten Parametern

Verwenden Sie beim Definieren einer Funktion {param1, param2,…}, um benannte Parameter anzugeben:

```
void enableFlags({bool bold, bool hidden}) {
  // ...
}
```

Beim Aufrufen einer Funktion können Sie benannte Parameter mit paramName: value angeben

```
enableFlags(bold: true, hidden: false);
```

Funktionsumfang

Dartfunktionen können auch anonym deklariert oder verschachtelt werden. Um beispielsweise eine verschachtelte Funktion zu erstellen, öffnen Sie einfach einen neuen Funktionsblock innerhalb eines vorhandenen Funktionsblocks

```
void outerFunction() {

    bool innerFunction() {
        /// Does stuff
    }
}
```

Die Funktion `innerFunction` kann jetzt innerhalb und nur in `outerFunction` . Keine anderen Funktionen haben Zugriff darauf.

Funktionen in Dart können auch anonym deklariert werden, was üblicherweise als Funktionsargument verwendet wird. Ein typisches Beispiel ist die `sort` des `List` Objekts. Diese Methode nimmt ein optionales Argument mit der folgenden Signatur an:

```
int compare(E a, E b)
```

In der Dokumentation heißt es, dass die Funktion `0` muss, wenn `a` und `b` gleich sind. Es gibt `-1` wenn `a < b` und `1` wenn `a > b` .

Wenn Sie dies wissen, können wir eine Liste ganzer Zahlen mit einer anonymen Funktion sortieren.

```
List<int> numbers = [4,1,3,5,7];

numbers.sort((int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
});
```

Die anonyme Funktion kann auch an Bezeichner gebunden sein:

```
Function intSorter = (int a, int b) {
   if(a == b) {
      return 0;
   } else if (a < b) {
      return -1;
   } else {
      return 1;
   }
}
```

und als gewöhnliche Variable verwendet.

```
numbers.sort(intSorter);
```

Funktionen online lesen: https://riptutorial.com/de/dart/topic/2965/funktionen

Kapitel 12: Klassen

Examples

Eine Klasse erstellen

Klassen können wie folgt erstellt werden:

```
class InputField {
  int maxLength;
  String name;
}
```

Die Klasse kann mit dem `new` Schlüsselwort instanziiert werden, wonach die Feldwerte standardmäßig null sind.

```
var field = new InputField();
```

Auf Feldwerte kann dann zugegriffen werden:

```
// this will trigger the setter
field.name = "fieldname";

// this will trigger the getter
print(field.name);
```

Mitglieder

Eine Klasse kann Mitglieder haben.

Instanzvariablen können mit / ohne Typanmerkungen deklariert und optional initialisiert werden. Nicht initialisierte Member haben den Wert `null` , sofern vom Konstruktor kein anderer Wert festgelegt wurde.

```
class Foo {
  var member1;
  int member2;
  String member3 = "Hello world!";
}
```

Klassenvariablen werden mit dem Schlüsselwort `static` deklariert.

```
class Bar {
  static var member4;
  static String member5;
  static int member6 = 42;
}
```

Wenn eine Methode keine Argumente annimmt, schnell ist, einen Wert zurückgibt und keine sichtbaren Nebeneffekte aufweist, kann eine Getter-Methode verwendet werden:

```
class Foo {
  String get bar {
    var result;
    // ...
    return result;
  }
}
```

Getter nehmen niemals Argumente an, daher werden die Klammern für die (leere) Parameterliste sowohl zum Deklarieren von Gettern wie oben als auch zum Aufrufen wie folgt ausgelassen:

```
main() {
  var foo = new Foo();
  print(foo.bar); // prints "bar"
}
```

Es gibt auch Setter-Methoden, die genau ein Argument enthalten müssen:

```
class Foo {
  String _bar;

  String get bar => _bar;

  void set bar(String value) {
    _bar = value;
  }
}
```

Die Syntax für den Aufruf eines Setters entspricht der Variablenzuweisung:

```
main() {
  var foo = new Foo();
  foo.bar = "this is calling a setter method";
}
```

Konstrukteure

Ein Klassenkonstruktor muss denselben Namen wie seine Klasse haben.

Erstellen Sie einen Konstruktor für eine Klasse Person:

```
class Person {
  String name;
  String gender;
  int age;

  Person(this.name, this.gender, this.age);
}
```

Das obige Beispiel ist eine einfachere und bessere Möglichkeit, den Konstruktor zu definieren, als

die folgende, auch mögliche Methode:

```
class Person {
  String name;
  String gender;
  int age;

  Person(String name, String gender, int age) {
    this.name = name;
    this.gender = gender;
    this.age = age;
  }
}
```

Jetzt können Sie eine Instanz von Person wie folgt erstellen:

```
var alice = new Person('Alice', 'female', 21);
```

Klassen online lesen: https://riptutorial.com/de/dart/topic/1511/klassen

Kapitel 13: Kontrollfluss

Examples

Ansonsten

Dart hat If Else:

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
} else {
  print('We Must Go Back!');
}
```

Dart hat auch einen ternären `if` Operator:

```
var foo = true;
print(foo ? 'Foo' : 'Bar'); // Displays "Foo".
```

While-Schleife

While-Schleifen und do-While-Schleifen sind in Dart zulässig:

```
while(peopleAreClapping()) {
  playSongs();
}
```

und:

```
do {
  processRequest();
} while(stillRunning());
```

Loops können mit einer Pause beendet werden:

```
while (true) {
  if (shutDownRequested()) break;
  processIncomingRequests();
}
```

Sie können Iterationen in einer Schleife mit continue überspringen:

```
for (var i = 0; i < bigNumber; i++) {
  if (i.isEven){
    continue;
  }
  doSomething();
```

```
}
```

Für Schleife

Zwei Arten von for-Schleifen sind zulässig:

```
for (int month = 1; month <= 12; month++) {
  print(month);
}
```

und:

```
for (var object in flybyObjects) {
  print(object);
}
```

Die `for-in` Schleife ist praktisch, wenn Sie einfach eine `Iterable` Auflistung `Iterable` . Es gibt auch eine `forEach` Methode, die Sie für `Iterable` Objekte `Iterable` können, die sich wie `for-in` verhalten:

```
flybyObjects.forEach((object) => print(object));
```

oder genauer:

```
flybyObjects.forEach(print);
```

Schaltergehäuse

Dart hat einen Schalter, der anstelle von langen if-else-Anweisungen verwendet werden kann:

```
var command = 'OPEN';

switch (command) {
  case 'CLOSED':
    executeClosed();
    break;
  case 'OPEN':
    executeOpen();
    break;
  case 'APPROVED':
    executeApproved();
    break;
  case 'UNSURE':
    // missing break statement means this case will fall through
    // to the next statement, in this case the default case
  default:
    executeUnknown();
}
```

Sie können nur Ganzzahl-, Zeichenfolgen- oder Kompilierungskonstanten vergleichen. Die verglichenen Objekte müssen Instanzen derselben Klasse sein (und nicht eines ihrer Subtypen), und die Klasse darf nicht == überschreiben.

Ein überraschender Aspekt des Wechsels in Dart ist, dass nicht leere Klauseln mit break enden oder seltener fortfahren, werfen oder zurückkehren müssen. Das heißt, nicht leere Fallklauseln können nicht durchfallen. Sie müssen eine nicht leere Case-Klausel explizit beenden, normalerweise mit einem break. Sie erhalten eine statische Warnung, wenn Sie break nicht fortfahren, fortfahren, werfen oder zurückkehren. Der Code wird zur Laufzeit an dieser Stelle angezeigt.

```
var command = 'OPEN';
switch (command) {
  case 'OPEN':
    executeOpen();
    // ERROR: Missing break causes an exception to be thrown!!

  case 'CLOSED': // Empty case falls through
  case 'LOCKED':
    executeClosed();
    break;
}
```

Wenn Sie in einem nicht leeren `case` einen Durchbruch erzielen möchten, können Sie `continue` und ein Etikett verwenden:

```
      var command = 'OPEN';
      switch (command) {
        case 'OPEN':
          executeOpen();
          continue locked;
locked: case 'LOCKED':
          executeClosed();
          break;
      }
```

Kontrollfluss online lesen: https://riptutorial.com/de/dart/topic/923/kontrollfluss

Kapitel 14: Pub

Bemerkungen

Wenn Sie das Dart-SDK installieren, erhalten Sie eines der Tools pub. Das Pub-Tool bietet Befehle für verschiedene Zwecke. Ein Befehl installiert Pakete, ein anderer startet einen HTTP-Server zum Testen, ein anderer bereitet Ihre App auf die Bereitstellung vor und ein anderer veröffentlicht Ihr Paket auf pub.dartlang.org . Sie können auf die pub-Befehle entweder über eine IDE wie WebStorm oder über die Befehlszeile zugreifen.

Eine Übersicht dieser Befehle finden Sie unter Pub-Befehle .

Examples

Pub bauen

Verwenden Sie pub build, wenn Sie bereit sind, Ihre Web-App bereitzustellen. Wenn Sie pub build ausführen, werden die Assets für das aktuelle Paket und alle seine Abhängigkeiten generiert und in einem neuen Verzeichnis namens build gespeichert.

Um `pub build` , führen Sie es einfach im Stammverzeichnis des Pakets aus. Zum Beispiel:

```
$ cd ~/dart/helloworld
$ pub build
Building helloworld......
Built 5 files!
```

Pub servieren

Dieser Befehl startet einen Entwicklungsserver oder Dev-Server für Ihre Dart-Web-App. Der dev-Server ist ein HTTP-Server auf localhost, der die Assets Ihrer Webanwendung bereitstellt.

Starten Sie den Dev-Server aus dem Verzeichnis, das die Datei `pubspec.yaml` Ihrer Web-App `pubspec.yaml` :

```
$ cd ~/dart/helloworld
$ pub serve
Serving helloworld on http://localhost:8080
```

Pub online lesen: https://riptutorial.com/de/dart/topic/3335/pub

Kapitel 15: Reguläre Ausdrücke

Syntax

- var regExp = RegExp (r '^ (. *) $', multiLine: true, caseSensitive: false);

Parameter

Parameter	Einzelheiten
`String source`	Der reguläre Ausdruck als `String`
`{bool multiline}`	Ob dies ein mehrzeiliger regulärer Ausdruck ist. (stimmt mit `^` und `$` am Anfang und Ende jeder Zeile überein
`{bool caseSensitive}`	Wenn der Ausdruck die Groß- und Kleinschreibung berücksichtigt

Bemerkungen

Dart-reguläre Ausdrücke haben dieselbe Syntax und Semantik wie reguläre JavaScript-Ausdrücke. Die Angabe von regulären JavaScript-Ausdrücken finden Sie unter http://ecma-international.org/ecma-262/5.1/#sec-15.10 .

Dies bedeutet, dass jede JavaScript-Ressource, die Sie online über reguläre Ausdrücke finden, für Dart gilt.

Examples

Erstellen und verwenden Sie einen regulären Ausdruck

```
var regExp = new RegExp(r"(\w+)");
var str = "Parse my string";
Iterable<Match> matches = regExp.allMatches(str);
```

Wenn Sie reguläre Ausdrücke schreiben, empfiehlt es sich, "rohe Zeichenfolgen" (Präfix mit `r`) zu verwenden, damit Sie in Ihrem Ausdruck nicht umgekippte umgekehrte Schrägstriche verwenden können.

Reguläre Ausdrücke online lesen: https://riptutorial.com/de/dart/topic/3624/regulare-ausdrucke

Kapitel 16: Sammlungen

Examples

Neue Liste erstellen

Listen können auf verschiedene Arten erstellt werden.

Die empfohlene Methode ist die Verwendung einer `List` wörtliche:

```
var vegetables = ['broccoli', 'cabbage'];
```

Der `List` Konstruktor kann ebenfalls verwendet werden:

```
var fruits = new List();
```

Wenn Sie eine stärkere Eingabe bevorzugen, können Sie einen Typparameter auf eine der folgenden Arten angeben:

```
var fruits = <String>['apples', 'oranges'];
var fruits = new List<String>();
```

Zum Erstellen einer kleinen, anwählbaren Liste, die entweder leer ist oder einige bekannte Anfangswerte enthält, wird die Literalform bevorzugt. Es gibt spezialisierte Konstruktoren für andere Arten von Listen:

```
var fixedLengthList1 = new List(8);
var fixedLengthList2 = new List.filled(8, "initial text");
var computedValues = new List.generate(8, (n) => "x" * n);
var fromIterable = new List<String>.from(computedValues.getRange(2, 5));
```

Siehe auch den Effective Dart Style Guide zu Sammlungen .

Ein neues Set erstellen

Sets können über den Konstruktor erstellt werden:

```
var ingredients = new Set();
ingredients.addAll(['gold', 'titanium', 'xenon']);
```

Eine neue Karte erstellen

Karten können auf verschiedene Arten erstellt werden.

Mit dem Konstruktor können Sie eine neue Karte wie folgt erstellen:

```
var searchTerms = new Map();
```

Typen für den Schlüssel und den Wert können auch mit Generics definiert werden:

```
var nobleGases = new Map<int, String>();
var nobleGases = <int, String>{};
```

Karten können ansonsten mit dem Kartenliteral erstellt werden:

```
var map = {
    "key1": "value1",
    "key2": "value2"
};
```

Ordnen Sie jedes Element in der Sammlung zu.

Alle Sammlungsobjekte enthalten eine `map` , die eine `Function` als Argument verwendet, für die ein einzelnes Argument erforderlich ist. Dies gibt ein `Iterable` zurück, das von der Sammlung unterstützt wird. Wenn die `Iterable` iteriert wird, ruft jeder Schritt die Funktion mit einem neuen Element der Auflistung auf, und das Ergebnis des Aufrufs wird das nächste Element der Iteration.

Sie können ein `Iterable` in eine Collection `Iterable.toSet()` indem Sie die `Iterable.toSet()` oder `Iterable.toList()` Methode verwenden oder einen Collection-Konstruktor verwenden, der ein Iterable wie `Queue.from` oder `List.from` .

Beispiel:

```
main() {
  var cats = [
    'Abyssinian',
    'Scottish Fold',
    'Domestic Shorthair'
  ];

  print(cats); // [Abyssinian, Scottish Fold, Domestic Shorthair]

  var catsInReverse =
  cats.map((String cat) {
    return new String.fromCharCodes(cat.codeUnits.reversed);
  })
  .toList(); // [nainissybA, dloF hsittocS, riahtrohS citsemoD]

  print(catsInReverse);
}
```

Das Dartpad-Beispiel finden Sie hier:
https://dartpad.dartlang.org/a18367ff767f172b34ff03c7008a6fa1

Eine Liste filtern

Dart ermöglicht das einfache Filtern einer Liste anhand von `where` .

```
var fruits = ['apples', 'oranges', 'bananas'];
fruits.where((f) => f.startsWith('a')).toList(); //apples
```

Natürlich können Sie einige AND- oder OR-Operatoren in Ihrer where-Klausel verwenden.

Sammlungen online lesen: https://riptutorial.com/de/dart/topic/859/sammlungen

Kapitel 17: Zeichenketten

Examples

Verkettung und Interpolation

Sie können den Operator plus (`+`) verwenden, um Zeichenfolgen zu verketten:

```
'Dart ' + 'is ' + 'fun!'; // 'Dart is fun!'
```

Sie können auch benachbarte String-Literale für die Verkettung verwenden:

```
'Dart ' 'is ' 'fun!';    // 'Dart is fun!'
```

Sie können `${}` um den Wert von Dart-Ausdrücken in Zeichenfolgen zu interpolieren. Die geschweiften Klammern können beim Auswerten von Bezeichnern weggelassen werden:

```
var text = 'dartlang';
'$text has ${text.length} letters'; // 'dartlang has 8 letters'
```

Gültige Zeichenfolgen

Eine Zeichenfolge kann ein- oder mehrzeilig sein. Einzeilige Zeichenfolgen werden mit übereinstimmenden einfachen oder doppelten Anführungszeichen geschrieben, und mehrzeilige Zeichenfolgen werden in dreifache Anführungszeichen geschrieben. Im Folgenden sind alle gültigen Dart-Zeichenfolgen aufgeführt:

```
'Single quotes';
"Double quotes";
'Double quotes in "single" quotes';
"Single quotes in 'double' quotes";

'''A
multiline
string''';

"""
Another
multiline
string""";
```

Aus Teilen bauen

Das programmgesteuerte Generieren eines Strings wird am besten mit einem StringBuffer ausgeführt . Ein StringBuffer generiert kein neues String-Objekt, bis `toString()` aufgerufen wird.

```
var sb = new StringBuffer();
```

```
sb.write("Use a StringBuffer");
sb.writeAll(["for ", "efficient ", "string ", "creation "]);
sb.write("if you are ")
sb.write("building lots of strings");

// or you can use method cascades:

sb
  ..write("Use a StringBuffer")
  ..writeAll(["for ", "efficient ", "string ", "creation "])
  ..write("if you are ")
  ..write("building lots of strings");

var fullString = sb.toString();

print(fullString);
// Use a StringBufferfor efficient string creation if you are building lots of strings

sb.clear(); // all gone!
```

Zeichenketten online lesen: https://riptutorial.com/de/dart/topic/5003/zeichenketten

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Dart	4444, Challe, Community, Damon, Florian Loitsch, Gomiero, Kleak, losnake, martin, Raph, Timothy C. Quinn
2	Asynchrone Programmierung	Challe, Damon, Ray Hulha, Zied Hamdi
3	Aufzählungen	Challe
4	Ausnahmen	Challe
5	Bemerkungen	Challe
6	Bibliotheken	Challe, Ganymede
7	Dart-JavaScript-Interoperabilität	Meshulam Silk
8	Daten konvertieren	Damon
9	Datum und Uhrzeit	Challe
10	Filter auflisten	jxmorris12
11	Funktionen	Jan Vladimir Mostert, Kim Rostgaard Christensen
12	Klassen	Ganymede, Hoylen, Jan Vladimir Mostert, Raph
13	Kontrollfluss	Ganymede, Jan Vladimir Mostert, Pacane, Raph
14	Pub	Challe
15	Reguläre Ausdrücke	enyo
16	Sammlungen	Alexi Coard, Damon, Jan Vladimir Mostert, Kleak, lrn, Pacane, Raph
17	Zeichenketten	Challe