FREE eBook

LEARNING pug

#pug

Table of Contents

About

Chapter 1: Getting started with pug

Remarks

Pug is a high performance, robust, elegant, feature rich template engine. It was influenced by Haml and implemented with JavaScript for Node.js and browsers. Implementations exists for Laravel, PHP Scala, Ruby, Python and Java.

It features:

- Express integration
- Conditionals
- Filters
- Includes
- Inheritance
- Interpolation
- Iteration
- Mixins

Pug was previously known under the *Jade* name but has been renamed because of a registered trademark case.

This remark section should also mention any large subjects within pug, and link out to the related topics. Since the Documentation for pug is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
2.0.0-beta11	2017-02-04
2.0.0-beta10	2017-01-29
2.0.0-beta9	2017-01-25
2.0.0-beta1	2016-06-03
1.11.0	2015-06-11

Examples

Installation

To install the Pug template rendering system, follow these steps:

1. Have the Node.js environment installed on your machine
2. Run `npm install pug --save` to install the `pug` module to your current project.

You can now use `pug` in your project through the standard `require` mechanism:

```
const pug = require("pug");
```

If you are using Express in your application, you do not need to `require("pug")`. However, you must set the `view engine` property of your Express application to `pug`.

```
app.set("view engine", "pug");
```

Further, you must set the view directory of your app so that Express knows where to look for your Pug files (for compilation).

```
app.set("views", "path/to/views");
```

Within your Express route, you can then render your Pug files by calling the `res.render` function with the path of the file (starting from the directory set by the `app.set("views")` option).

```
app.get("/", function (req, res, next) {
    // Your route code
    var locals = {
        title: "Home",
    };
    res.render("index", locals);
});
```

In the above, `index` points to a file located at `views/index.pug`, and `locals` represents an object of variables that are exposed to your file. As will be explained in later sections, Pug can access variables passed to it and perform a variety of actions (conditionals, interpolation, iteration, and more).

Hello World Example

First, let's create a template to be rendered!

```
p Hello World, #{name}!
```

Save this in a file ending with the `.pug` extension (you can call it anything you like, but we will use `view.pug` in the following code to compile it).

All that's left to do, now, is compile that template! Create a JS script file (we'll call ours `main.js`), and add the following content:

```
// Import the pug module
const pug = require('pug');

// Compile the template (with the data not yet inserted)
```

```
const templateCompiler = pug.compileFile('view.pug');

// Insert your data into the template file
console.log(templateCompiler({ name: 'John' });
```

When you run this file with `npm main.js`, you should get the following HTML code output in your console:

```
<p>Hello World, John!</p>
```

Congratulations, you just created and compiled your first template! On to more advanced stuff, such as Conditionals, Iteration, and much more!

Read Getting started with pug online: https://riptutorial.com/pug/topic/8613/getting-started-with-pug

Chapter 2: Conditionals

Introduction

Pug can conditionally run code based on variables (passed from your server or based in Pug itself).

Syntax

- if (statement)

  ```
  // Pug code
  ```

- else if (statement)

  ```
  // Pug code
  ```

- else

  ```
  // Pug code
  ```

- unless (statement)

  ```
  // Pug code
  ```

Parameters

Parameter	Details
if (statement)	Evaluates `statement` to see if it returns true or false. The code nested underneath `if` will run only if `statement` returns true.
else if (statement)	Chained to an existing `if` or `else if` statement; it only runs if the previous statement evaluated to false. The code nested underneath the `else if` statement will run only if `statement` evaluates to true.
else	The code nested underneath the `else` statement will run only if all previous statements returned false.
unless (statement)	The negation of `if (statement)`; the code nested underneath `if` will run only if `statement` returns false. It is the same as `if (!statement)`.

Remarks

Official PugJS documentation on conditionals

Examples

If/Else Statement in Pug

Conditionals in Pug can evaluate statements in a manner similar to JavaScript. You can evaluate variables created in Pug, or those passed to it by your route (`res.render`, `pug.renderFile`, etc).

index.js

```
var authorized = true
res.render("index", {
    authorized: authorized
});
```

index.pug

```
- var showLogin = false;
if authorized && showLogin === true
    .welcome Welcome back to our website!
else
    .login
        a(href="/login") Login
```

index.pug output

```
<div class="login"><a href="/login">Login</a></div>
```

If/Else Statement in Pug (with a dash)

You can choose to prepend an `if` or `else` operator with a dash, but it is not necessary. You will need to wrap the statement in parentheses, though (if you omit a dash, you do not need parentheses.)

```
- var showLogin = false;
- if (showLogin === true)
    .welcome Welcome back to our website!
- else
    .login
        a(href="/login") Login
```

index.pug output

```
<div class="login"><a href="/login">Login</a></div>
```

Else If Statement

You can chain any number of `else if` statements to an existing `if` statement, to evaluate a sequence of statements.

index.pug

```
- var page = 60;
if page => 52
    h1 Lots of numbers!
else if page > 26 && page < 52
    h1 A few numbers
else
    h1 Not a lot of numbers
```

index.pug output

```
<h1>Lots of numbers!</h1>
```

Unless Operator

`unless` is the inverse operation of `if` in Pug. It is analogous to `if !(statement)`.

index.pug

```
- var likesCookies = true;
unless likesCookies === true
    h2 You don't like cookies :(
else
    h2 You like cookies!
```

index.pug output

```
<h1>You like cookies!</h1>
```

Note: `else unless` statements do not work with `unless`; you can chain an `else if` statement to an `unless` statement, but `else unless` does not work.

Read Conditionals online: https://riptutorial.com/pug/topic/9662/conditionals

Chapter 3: Interpolation with Pug

Introduction

It's important to be able to use server-side variables in your website. Pug allows you to interpolate data generated by your server in HTML, CSS, and even JavaScript code.

Syntax

- res.render(path, variables) // Searches for a pug file to render at path "path", and passes "variables" to it
- #{variable} // Interpolates "variable" inline with the surrounding Jade code, after evaluating "variable"
- !{variable} // Interpolates "variable" inline with the surrounding Jade code, without evaluating "variable".
- #[element] // Interpolates "element" inside of an existing Pug HTML element. Syntax of interpolated HTML elements is identical to that of normal HTML elements.

Parameters

Parameter	Details
path	Used in `res.render`. This is the path of the Pug file that we are going to render. The path is taken from the root of the folder set on your Express app: `app.set("views", "templates/views")`. For example, `res.render("index")` will search for a Pug file at `templates/views/index.pug`. Subdirectories can be specified too; `res.render("admin/index")` looks for a Pug file at `templates/views/admin/index.pug`.
variables	Used in `res.render`. A JavaScript object of variables to be made accessible to the Pug file defined by `path` (above). Within the Pug file, the keys of the above JavaScript object become available as variables. If `variables = {title: "Hello", color: "red"}`, we could use the `title` and `color` variable. Subproperties of nested objects are also available.
variable	Used in bracket syntax `#{}` or `!{}`. The value of `variable` will be output in the context of its surrounding Pug code. If a pound symbol is prepended to the opening curly bracket, `variable` will be evaluated before being output. If an exclamation point is prepended to the opening curly brace, `variable` **will not** be evaluated.
element	Used in square bracket sytax `#[]`. The HTML element (in Pug syntax, not normal HTML syntax) will be evaluated and output inline with the surrounding Pug code.

Remarks

For more information on PugJS interpolation, see the official PugJS interpolation documentation.

Examples

Server Side Variable Interpolation

It's possible to pass variables from your server into Pug for dynamic content or script generation. Pug templates can access variables passed to the `res.render` function in Express (or `pug.renderFile` if you are not using Express, the arguments are identical).

index.js

```
let colors = ["Red", "Green", "Blue"];
let langs  = ["HTML", "CSS", "JS"];
let title  = "My Cool Website";

let locals = {
    siteColors: colors,
    siteLangs:  langs,
    title:      title
};
res.render("index", locals);
```

Inside your `index.pug` file, you then have access to the `locals` variable by way of its keys. The names of the variables in your Pug file become `siteColors` and `siteNames`.

To set the entirety of an HTML element equal to a variable, use the equals operator `=` to do so. If your variable needs to be embedded inline, use bracket syntax `#{}` to do so.

index.pug

```
doctype html
html
    head
        title= title
    body
        p My favorite color is #{siteColors[0]}.
        p Here's a list of my favorite coding languages
        ul
            each language in siteLangs
                li= language
```

index.pug output

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Cool Website</title>
    </head>
    <body>
```

```
        <p>My favorite color is Red.</p>
        <p>Here's a list of my favorite coding languages</p>
        <ul>
            <li>HTML</li>
            <li>CSS</li>
            <li>JS</li>
        </ul>
    </body>
</html>
```

Raw Variable Interpolation in HTML

Content interpolated with bracket syntax will be evaluated for code, the output of which is included in your HTML output.

> title follows the basic pattern for evaluating a template local, but the code in between `#{` and `}` is evaluated, escaped, and the result buffered into the output of the template being rendered. [Source]

If you need to include raw HTML syntax, use an exclamation point instead of a pound symbol (`!{}` instead of `#{}`).

index.js

```
let tag = "<div>You can't escape me!</div>";
res.render("index", {
    myTag: tag
});
```

index.pug

```
doctype html
html
    head
    body
        !{myTag}
```

index.pug output

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <div>You can't escape me!</div>
    </body>
</html>
```

Value Interpolation in JavaScript Code

Interpolating values is helpful if you need to pass a server-side variable to client-side JavaScript (or other languages that require it).

In the case of variables, numbers, strings, and the like, you can pass these types of variables directly into your JavaScript with bracket syntax plus an explanation point (so that the code inside the brackets *is not* evaluated.) This is useful for parametrizing JavaScript code that require something from your server.

In the below example, we have to wrap `username` in quotation marks in order for JavaScript to interpret it as a string; Pug will output the content of the variable as-is, so we need to put it in quotation marks for it to be a proper JavaScript string. This is not necessary for `number`, where JavaScript will interpret our number as it we intend it to (as a number).

index.js

```
let number   = 24;
let username = "John";
res.render("index", {
    number:   number,
    username: username
});
```

index.pug

```
html
    head
    script.
        // Sets the username of the current user to be displayed site-wide
        function setUsername(username) {
            // ...
        }
        var number   = #{number};
        var username = "#{username}";
        setUsername(username);

    body
        p Welcome to my site!
```

index.pug output

```
<html>
    <head>
        <script>
            // Sets the username of the current user to be displayed site-wide
            function setUsername(username) {
                // ...
            }
            var number   = 24;
            var username = "John";
            setUsername(username);
        </script>
    </head>
    <body>
        <p>Welcome to my site!</p>
    </body>
</html>
```

If you need to interpolate the value of a JavaScript object (e.g. all the information about a user), you must stringify the output in Pug for it to be treated as a JavaScript object. It's also necessary to output the raw contents of the variable, instead of the evaluated form of it. If you were to do output the escaped variable (`var user = #{JSON.stringify(user)}`), you would receive an *escaped* version of the object (where quotation marks and apostrophes are converted to `&quot;`), which is not what we want in order for `JSON.stringify` to work on it.

index.js

```
var myUser = {
    name:    "Leeroy Jenkins",
    id:      1234567890,
    address: "123 Wilson Way, New York NY, 10165"
};

res.render('index', {
    user: myUser
});
```

index.pug

```
doctype html
html
    head
        script.
            window.onload = function () {
                function setUsername(username) {
                    return username;
                }

                var user = !{JSON.stringify(user)};
                document.getElementById("welcome-user").innerHTML = setUsername(user.name);
            };

    body
        div(id="welcome-user")
```

index.pug output

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            window.onload = function() {
                function setUsername(username) {
                    return username;
                }

                var user = {
                    "name": "Leeroy Jenkins",
                    "id": 1234567890,
                    "address": "123 Wilson Way, New York NY, 10165"
                };
                document.getElementById("welcome-user").innerHTML = setUsername(user.name);
        };
        </script>
    </head>
    <body>
        <div id="welcome-user"></div>
    </body>
</html>
```

The `innerHTML` of `#welcome-user` becomes equal to `Leeroy Jenkins`. The contents of the `user` variable are printed directly to the HTML source

---

HTML Element Interpolation

It may be necessary to nest HTML tags inside of each other. Element interpolation is done in a syntax similar to variable interpolation; square brackets instead of curly braces are used here. The syntax of interpolated HTML elements is identical to the implementation of normal HTML elements.

index.pug

```
doctype html
html
    head
        title My Awesome Website
    body
        p The server room went #[b boom]!
        p The fire alarm, however, #[u failed to go off...]
        p Not even #[a(href="https://stackoverflow.com/") Stack Overflow] could save them now.
```

index.pug output

```
<!DOCTYPE html>
<html>
    <head>
        <title>My Awesome Website</title>
    </head>
    <body>
        <p>The server room went <b>boom</b>!</p>
        <p>The fire alarm, however, <u>failed to go off...</u></p>
        <p>Not even <a href="https://stackoverflow.com/">Stack Overflow</a> could save them
now.</p>
    </body>
</html>
```

Read Interpolation with Pug online: https://riptutorial.com/pug/topic/9565/interpolation-with-pug

Chapter 4: Iteration with Pug

Introduction

How to iterate over a simple JSON object and save a lot of typing

Remarks

You need to have Node.js and Pug installed

Examples

Each iteration

Build an `app.js` with a simple *data store*:

```
app.get("/bookstore", function (req, res, next) {
    // Your route data
   var bookStore = [
        {
            title: "Templating with Pug",
            author: "Winston Smith",
            pages: 143,
            year: 2017
        },
        {
            title: "Node.js will help",
            author: "Guy Fake",
            pages: 879,
            year: 2015
        }
    ];
    res.render("index", {
        bookStore: bookStore
    });
});
```

Iterate over the data store using an `index.pug` file and an each loop:

```
each book in bookStore
  ul
    li= book.title
    li= book.author
    li= book.pages
    li= book.year
```

Result will be:

```
<ul>
  <li>Templating with Pug</li>
  <li>Winston Smith</li>
  <li>143</li>
  <li>2017</li>
</ul>
<ul>
  <li>Node.js will help</li>
  <li>Guy Fake</li>
  <li>879</li>
  <li>2015</li>
</ul>
```

Reference

Read Iteration with Pug online: https://riptutorial.com/pug/topic/9545/iteration-with-pug

Chapter 5: Syntax and markup generation

Introduction

A preview of the difference between pug code and the generated markup

Remarks

Pug makes possible to write HTML in a simplest way, using a clean, whitespace sensitive syntax.

Examples

From Pug to HTML

```
doctype html
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript').
      if (foo) bar(1 + 5)
  body
    h1 Pug - node template engine
    #container.col
      if youAreUsingPug
        p You are amazing
      else
        p Get on it!
      p.
        Pug is a terse and simple templating language with a
        strong focus on performance and powerful features.
```

Becomes:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Pug</title>
    <script type="text/javascript">
      if (foo) bar(1 + 5)
    </script>
  </head>
  <body>
    <h1>Pug - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Pug is a terse and simple templating language with a strong focus on performance and
powerful features.</p>
    </div>
  </body>
</html>
```

Read Syntax and markup generation online: https://riptutorial.com/pug/topic/9549/syntax-and-markup-generation

Credits

S. No	Chapters	Contributors
1	Getting started with pug	Community, gandreadis, Shea Belsky, smonff
2	Conditionals	Shea Belsky
3	Interpolation with Pug	Shea Belsky
4	Iteration with Pug	Shea Belsky, smonff
5	Syntax and markup generation	smonff