|  |  |
| --- | --- |
| **Tags** | |
| Text at the start of a line (or after only white space) represents an HTML tag. Indented tags are nested, creating the tree structure of HTML. Pug knows which tags are self-closing (img, br, link) | |
| ul  li First  li Second  li Third | <ul>  <li>First</li>  <li>Second</li>  <li>Third</li>  </ul> |
| img | <img/> |
| **Block Expansion** | |
| To save space, Pug provides an inline syntax for nested tags. | |
| a: img | <a><img/></a> |

|  |  |
| --- | --- |
| . **Class Literal** | |
| Classes may be defined using a .classname syntax: | |
| a**.**button | <a **class**="button"></a> |
| div's are such a common tag, it is the default if you omit the tag name: | |
| **.**primary | <div **class**="primary"></div> |
| # **ID Literal** | |
| IDs may be defined using a #idname syntax: | |
| a**#**linkToSource  **#**content | <a **id**="linkToSource"></a>  <div **id**="content"></div> |

|  |  |
| --- | --- |
| if **Conditionals** else, unless | |
| Pug’s first-class conditional syntax allows for optional parentheses | |
| - var user = { descr: 'user details' }  - var authorized = false  **#user**  if user.descr  h2.green Description  p**.description**= user.descr  else if authorized  h2.blue Description  p.description.  User has no description,  But is authorized  else  h2.red Description  p.description No description or auth | <div **id="user"**>  <h2 class="green">Description </h2>  <p **class="description"**>user details</p>  </div> |
| unless user.isAnonymous  p You're logged in as #{user.name} | if !user.isAnonymous  p You're logged in as #{user.name} |

|  |  |
| --- | --- |
| **case** | |
| case friends  when 0  p you have no friends  when 1  p you have a friend  when 2  when 3   * break   default  p you have #{friends} friends | (if var friends = 10)  <p>you have 10 friends</p>  ( if var friends = 0)  <p>you have no friends</p>  (if var friends = 2) |
| case friends  when 0 : p you have no friends  when 1 : p you have a friend  default: p you have #{friends} friends | (if var friends = 1)  <p>you have a friend</p>  (if var friends = 17)  <p>you have 17 friends</p> |

|  |  |
| --- | --- |
| each (for) **Iteration** while | |
| ul  **each** val **in** [1, 2, 3]  li= val | <ul>  <li>1</li>  <li>2</li>  <li>3</li>  </ul> |
| ul  **each** **e**, **index** **in** ['zero', 'one', 'two']  li= **index** + ': ' + **e** | <ul>  <li>0: zero</li>  <li>1: one</li>  <li>2: two</li>  </ul> |
| ul  **each** val, **index** in {1:'one', 2:'two'}  li= index + ': ' + val | <ul>  <li>1: one</li>  <li>2: two</li>  </ul> |
| - var values = [];  ul  **each** val **in** values.length ? values : ['There are', 'no values']  li= val | <ul>  <li>There are</li>  <li>no values</li>  </ul> |
| - var values = [];  ul  **each** val **in** values  li= val  **else**  li There are no values | <ul>  <li>There are no values</li>  </ul> |
| - var n = 0;  ul  **while** n < 3  li= n++ | <ul>  <li>0</li>  <li>1</li>  <li>2</li>  </ul> |

|  |  |
| --- | --- |
| // **Comments** | |
| // One line comment  p Something here  p Something following | <!—One line comment-->  <p>Something here</p>  <p>Something following</p> |
| The dash will not send the comment to the output buffer | |
| //- will not output within markup  p Something here  p Something following | <p>Something here</p>  <p>Something following</p> |
| //- **Block Comments** // | |
| body  //-  Comments  Use as much text as you want.  // | <body>  <!—Comments  Use as much text as you want.-->  </body> |

|  |  |
| --- | --- |
| **&attributes** | |
| Pronounced as “and attributes”, the &attributes syntax can be used to unwind the JSON object to key=value pairs to create attributes of an element. | |
| div**#help**(data="current")&attributes ({**'news-src': 'CNN'**}) | <div **id="help"** data="current" **news-src="CNN"**></div> |
| - var attributes = {};  - attributes**.class = 'breaking'**;  div#top(**data-src="CNN"**) &attributes(attributes) | <div **class="breaking"** id="top" **data-src="CNN"></**div> |

|  |  |
| --- | --- |
| **Attributes** | |
| a(href='bing.com') Bing  a(class='button' href='bing.com') Bing  a(class='button'**,** href='bing.com') Bing | <a href="bing.com">Bing</a>  <a class="button" href="bing.com">Bing</a>  <a class="button" href="bing.com">Bing</a> |
| Normal JavaScript expressions work fine, too: | |
| - var authenticated = true  body(class=authenticated ? 'authed' : 'anon') | <body class="authed"></body> |
| **Attribute Interpolation** | |
| Including variables in your attribute: Write attribute in JavaScript: | |
| - var url = 'pug-test.html';  a(href='/' + url) Link  - url = 'https://example.com/'  a(href=url) Another link | <a href="/pug-test.html">Link</a>  <a href="https://example.com/">Another link</a> |
| If JavaScript runtime supports ES2015 template strings you can use its syntax to simplify attributes: | |
| - var btnType = 'info'  - var btnSize = 'lg'  button(type='button' class='btn btn-' + btnType + ' btn-' + btnSize)  button(type='button' class=`btn btn-${btnType} btn-${btnSize}`) | <button class="btn btn-info btn-lg" type="button"></button>  <button class="btn btn-info btn-lg" type="button"></button> |
| **Boolean Attributes** | |
| Boolean values (true and false) are accepted. No value specified? true is assumed. | |
| input(type='checkbox' checked)  input(type='checkbox' checked=false)  input(type='checkbox' checked=true.toString()) | <input type="checkbox" checked="checked" />  <input type="checkbox" />  <input type="checkbox" checked="true" /> |
| **Style Attributes** | |
| The style attribute can be a string or it can be an object. | |
| a(style={color: 'red', background: 'green'}) | <a style="color:red; background:green;"></a> |
| **Class Attributes** | |
| The class attribute can be a string or it can be an array of class names. | |
| - var **classes** = [**'alpha', 'beta', 'rho'**]  a(class=**classes**)  //- the class attribute may also be repeated to merge arrays  a.bang(class=**classes** class=['bing']) | <a class="**alpha beta rho**"></a>  <a class="bang **alpha beta rho** bing"></a> |
| It can also be an object which maps class names to true or false values. This is useful for applying conditional classes. (Class is ‘active’ if currentUrl matches the href for the tag) | |
| - var currentUrl = '/about'  a(class={active: currentUrl === '/'} href='/') Home  a(class={active: currentUrl === '/about'} href='/about') About | <a href="/">Home</a>  <a class="active" href="/about">About</a> |

|  |  |
| --- | --- |
| **Code** | |
| - **Unbuffered (not sent to output buffer) Code** | |
| Execute the code but it does not get rendered in the HTML | |
| - for (var x = 0; x < 3; x++)  li item | <li>item</li>  <li>item</li>  <li>item</li> |
| **block unbuffered code:** | |
| -  var list = [Edge", "Tech", "Academy"]  each item in list  li= item | <li>Edge</li>  <li>Tech</li>  <li>Academy</li> |
| = **Buffered Code** | |
| evaluates the expression and outputs the result. Output is HTML escaped. | |
| p  = 'Code is <escaped>!' | <p>Code is &lt;escaped&gt;!</p> |
| p= 'Code is' + ' <escaped>!' | <p>Code is&lt;escaped&gt;!</p> |
| != **Unescaped Buffered Code** | |
| Unescaped code starts with != It evaluates the expression and outputs the result. Unsafe for user input: | |
| p  != '<strong>not</strong> escaped!' | <p><strong>not</strong> escaped!</p> |
| p!= 'This is' + ' <strong>not</strong> escaped!' | <p>This is <strong>not</strong> escaped!</p> |

|  |  |
| --- | --- |
| **String Interpolation, Escaped** | |
| Observe the placement of the following locals: **title**, **author**, and **panic**.  title follows basic pattern. Code between #{ and } is evaluated and escaped | |
| - var **title** = "Hitchhikers Guide";  - var **author** = "Douglas Adams";  - var **panic** = "<span>Panic</span>";  h1= **title**  p Written by #{**author**}  p This will be safe: don’t #{**panic**}  - var msg = "not my inside voice";  p This is #{msg.toUpperCase()} | <h1>Hitchhikers Guide</h1>  <p>Written by Douglas Adams</p>  <p>This will be safe: don’t &lt;span&gt;Panic&lt;/span&gt;</p>  <p>This is NOT MY INSIDE VOICE</p> |
| **!** **String Interpolation, Unescaped** | |
| - var darth = "<em>I’m your father, Luke</em>";  .quote  p Darth: !{darth} | <div class="quote">  <p>Darth: <em>I’m your father, Luke!</em></p>  </div> |
| **#[ Tag Interpolation ]** | |
| p.  This is a very long and boring paragraph that spans multiple lines.  Suddenly there is a **#[strong** strongly worded phrase**]** that cannot be  **#[em** ignored**]**.  p.  An interpolated tag with an attribute:  #[q(lang="es") ¡Hola!] | <p>This is a very long and boring paragraph that spans multiple lines. Suddenly there is a <strong>strongly worded phrase</strong> that cannot be  <em>ignored</em>.</p>  <p>An interpolated tag with an attribute:  <q lang="es">¡Hola!</q></ p> |
| **Whitespace Control** | |
| Pug removes all spaces before and after tags. Check out the following example: | |
| p  | If I don't write the paragraph with tag interpolation, tags like  strong strong  | and  em em  | might produce unexpected results.  p.  If I do, whitespace is #[strong respected] and #[em everybody] is happy. | <p>If I don't write the paragraph with tag interpolation, tags like<strong>strong</strong> and<em>em</em>might produce unexpected results.</p>  <p>If I do, whitespace is <strong>respected</strong> and <em>everybody</em> is happy.</p> |

|  |  |
| --- | --- |
| **include** | |
| **//- index.pug**  doctype html  html  **include** **includes/head.pug**  **body**  **h1 My Site**  **p Welcome to my small site.**  **include includes/footer.pug** | <!DOCTYPE html>  <html>  <head>  <title>My Site</title>  <script src="/javascripts/jquery.js"></script>  <script src="/javascripts/app.js"></script>  </head>  **<body>**  **<h1>My Site</h1>**  **<p>Welcome to my small site.</p>**  <footer id="footer">  <p>Copyright (c) foobar</p>  </footer>  **</body>**  </html> |
| **//- includes/head.pug**  head  title My Site  script(src='/javascripts/jquery.js')  script(src='/javascripts/app.js') |
| **//- includes/footer.pug**  footer#footer  p Copyright (c) foobar |
| If the path is absolute it is resolved by prepending options.basedir. Paths relative to the current file being compiled. Extension is assumed to be .pug | |
| **//- index.pug**  **doctype html**  **html**  **head**  **style**  include style.css  **body**  **h1 My Site**  **p Welcome to my super site.**  **script**  **include script.js** | **<!DOCTYPE html>**  **<html>**  **<head>**  **<style>**  /\* style.css \*/  h1 { color: red; }  **</style>**  **</head>**  **<body>**  **<h1>My Site</h1>**  **<p>Welcome to my super site.</p>**  **<script>**  **// script.js**  **console.log('You are awesome');**  **</script>**  **</body>**  **</html>** |
| /\* style.css \*/  h1 { color: red; } |
| **// script.js**  **console.log('You are awesome');** |

|  |  |
| --- | --- |
| **Template Inheritance** block | |
| * **Pug Template** inheritance works via the block and extends keywords. * A block is simply a “block” of Pug that a child template **may** replace. * Below blocks provide default content for: scripts, content, foot. * Extends: Then, define one or more blocks to override the parent block content. Then if foot block is not redefined, it will use the parent’s default and output “some footer content”. | |
| **//- layout.pug**  html  head  title My Site - #{title}  block scripts  **~~script(src='/default.js')~~**  body  block content  block foot  #footer  p some footer content | <html>  <head>  <title>My Site – More Pug!</title>  **<script src="/jquery.js"> </script>**  **<script src="/javascripts/pets.js"></script>**  </head>  <body>  <h1>More Pug!</h1>  **<p>fluffy</p>**  **<p>rascal</p>**  **<p>zeke</p>**  **<div id="footer">**  **<p>some footer content</p>**  **</div>**  </body>  </html> |
| //- pageA.pug  **extends layout.pug**  block scripts  **script(src='/jquery.js')**  **script(src='/pets.js')**  block content  h1= title  - var pets =**['fluffy', 'rascal', zeke’]**  each petName in pets  **include pet.pug** |
| **//- pet.pug**  **p= petName** |
| It is possible to override a block to provide additional blocks. Content now exposes a sidebar and primary block for overriding. | |
| **//- sub-layout.pug**  **extends layout.pug**  block content  .sidebar  block sidebar  p nothing  .primary  block primary  p nothing | //- page-b.pug  **extends sub-layout.pug**  block content  .sidebar  block sidebar  p Custom Sidebar |
| **Block** append**,** prepend | |
| Pug allows you to replace (default), prepend, or append blocks. Use for default scripts in a head block used on every page. Only named blocks and mixin definitions can appear at the top (unindented) level of a child template. Parent templates define a page’s overall structure, and child templates can only append, prepend, or replace specific blocks. Want related scripts **plus** the defaults? append the block | |
| **//- layout.pug**  html  head  block head  script(src=’/vendor/jquery.js’)  script(src=’/vendor/vendor2.js’)  body  block content | //- page.pug  **extends layout.pug**  **block** **append** head  script(src=’/vendor/three.js’)  script(src=’/game.js’) |

|  |  |
| --- | --- |
| **Mixin** | |
| Mixins allow you to create reusable blocks of Pug. | |
| //- Declaration  mixin list  ul  li Doe  li Ray  li Me  //- Use  **+list**  **+list** | **<ul>**  **<li>Doe</li>**  **<li>Ray</li>**  **<li>Me</li>**  **</ul>**  <ul>  <li>Doe</li>  <li>Ray</li>  <li>Me</li>  </ul> |
| mixin pet(name)  li.pet= name  ul  **+pet('Fluffy')**  +pet('Rascal')  **+pet('Zeke')** | <ul>  **<li class="pet">Fluffy</li>**  <li class="pet">Rascal</li>  **<li class="pet">Zeke</li>**  </ul> |
| **Mixin Blocks** | |
| Mixins can also take a block of Pug to act as the content: | |
| mixin article(title)  .article  .article-wrapper  h1= title  if block  **block**  else  p No content provided  **+article('Hello world')**  +article('Hello world')  **p This is my**  **p Amazing article** | **<div class="article">**  **<div class="article-wrapper">**  **<h1>Hello world</h1>**  **<p>No content provided</p>**  **</div>**  **</div>**  **<div class="article">**  **<div class="article-wrapper">**  **<h1>Hello world</h1>**  **<p>This is my</p>**  **<p>Amazing article</p>**  **</div>**  **</div>** |
| **Mixin Attributes** | |
| Mixins get an implicit attributes argument, taken from the attributes passed to the mixin: | |
| mixin link(href, name)  //- attributes == {class: "btn"}  a(class!=attributes.class href=href)= name  +link('/top', 'top')(class="btn") | <a class="btn" href="/top">top</a> |
| You can also use mixins with **&attributes:** | |
| mixin link(href, name)  a(href=href)**&attributes**(attributes)= name  +link('/top', 'top')(class="btn")  +link('/next', 'next') | <a class="btn" href="/top">top</a>  <a href="/next">next</a> |
| The syntax +link(class="btn") is also valid and equivalent to +link()(class="btn"), since Pug tries to detect if parentheses’ contents are attributes or arguments. Use the second syntax to be safe. | |
| **Rest Arguments** ... | |
| You can write mixins that take an unknown number of arguments using the “rest arguments” syntax. | |
| mixin list(id, ...items)  ul(id=id)  each item in items  li= item  +list('my-list', 1, 2, 3, 4) | <ul id="my-list">  <li>1</li>  <li>2</li>  <li>3</li>  <li>4</li>  </ul> |

|  |  |
| --- | --- |
| **Plain Text** | |
| **Inline in a Tag** | |
| The first term on a line is the tag itself. Everything after the tag and one space will be the text contents of that tag. Useful when the plain text content is short. | |
| p This is plain old <em>text</em> content. | <p>This is plain old <em>text</em> content.</p> |
| **Literal HTML** | |
| When lines begin with a left angle bracket (<) they are treated as plain text, which is useful for writing literal HTML tags in places that could otherwise be inconvenient. Literal HTML do not self-close. | |
| <html>  body  p Indenting the body tag here would make no difference.  p HTML itself isn't whitespace-sensitive.  </html> | <html>  <body>  <p>Indenting the body tag here would make no difference.</p>  <p>HTML itself isn't whitespace-sensitive.</p>  </body>  </html> |
| **|** **Piped Text** | |
| Another way to add plain text to templates is to prefix a line with a pipe character (|). This method is useful for mixing plain text with inline tags, as we discuss later, in the Whitespace Control section. | |
| P  | The pipe always goes at the beginning of its own line,  | not counting indentation. | <p>The pipe always goes at the beginning of its own line, not counting indentation.</p> |
| **Block in a Tag** . | |
| Often you might want large blocks of text within a tag. A good example is writing JavaScript and CSS code in the script and style tags. To do this, just add a . (period) right after the tag name, or after the closing parenthesis, if the tag has attributes. Plain text contents of the tag must be indented one level | |
| script.  **if (usingPug)**  **console.log('you are awesome')**  **else**  **console.log('use pug')** | <script>  **if (usingPug)**  **console.log('you are awesome')**  **else**  **console.log('use pug')**  </script> |
| create a dot block of plain text after other tags within the parent tag. | |
| div  p This text belongs to the p tag.  br  .  This text belongs to the div tag. | <div>  <p>This text belongs to the p tag.</p><br/>This text belongs to the div tag.</div> |