

# Attributes

Tag attributes look similar to HTML (with optional commas), but their values are just regular JavaScript.

(NOTE: Examples on this page use the pipe character ( `|` ) for **whitespace control**.)

```
a(href='//google.com') Google
|
|
a(class='button' href='//google.com') Google
|
|
a(class='button', href='//google.com') Google
```

```
<a href="//google.com">Google</a>
<a class="button" href="//google.com">Google</a>
<a class="button" href="//google.com">Google</a>
```

Normal JavaScript expressions work fine, too:

```
- var authenticated = true
body(class=authenticated ? 'authed' : 'anon')
```

```
<body class="authed"></body>
```

## Multiline Attributes ¶

If you have many attributes, you can also spread them across many lines:

```
input(
  type='checkbox'
  name='agreement'
  checked
```

```
)
<input type="checkbox" name="agreement" checked="checked" />
```

If your JavaScript runtime supports ES2015 **template strings** (including Node.js/io.js 1.0.0 and later), you can use that syntax for attributes. This is really useful for attributes with really long values:

```
input(data-json=`
  {
    "very-long": "piece of ",
    "data": true
  }
`)
```

```
<input data-json="
  {
    &quot;very-long&quot;;: &quot;piece of &quot;;,
    &quot;data&quot;;: true
  }
" />
```

## Quoted Attributes ¶

If your attribute name contains odd characters that might interfere with JavaScript syntax, either quote it using `""` or `' '`, or use commas to separate different attributes. Examples of such characters include `[]` and `()` (frequently used in Angular 2).

```
//- In this case, `(click)` is treated as a
//- function call instead of a attribute name,
//- resulting in the unusual error.
div(class='div-class' (click)='play()')
```

```
index.pug:4:11
    2| //- function call instead of a attribute name,
    3| //- resulting in the unusual error.
    > 4| div(class='div-class' (click)='play()')
      -----^
```

Syntax Error: Assigning to rvalue

```
div(class='div-class', (click)='play()')  
div(class='div-class' '(click)='play()')
```

```
<div class="div-class" (click)="play()"></div>  
<div class="div-class" (click)="play()"></div>
```

## Attribute Interpolation ¶

### Caution

Previous versions of Pug/Jade supported an interpolation syntax such as:

```
a(href="#{url}") Link
```

This syntax is **no longer supported**. Alternatives are found below. (Check our migration guide for more information on other incompatibilities between Pug v2 and previous versions.)

Here are some alternatives you can use to include variables in your attribute:

1. Simply write the 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>
```

2. If your JavaScript runtime supports ES2015 **template strings** (including Node.js/io.js 1.0.0 and later), you can also use its syntax to simplify your attributes:

```
- var btnType = 'info'
- var btnSize = 'lg'
button(type='button' class='btn btn-' + btnType + ' btn
|
|
button(type='button' class=`btn btn-${btnType} btn-${bt
```

```
<button class="btn btn-info btn-lg" type="button"></but
<button class="btn btn-info btn-lg" type="button"></but
```

## Unescaped Attributes ¶

By default, all attributes are escaped—that is, special characters are replaced with escape sequences—to prevent attacks (such as cross site scripting). If you need to use special characters, use `!=` instead of `=`.

```
div(escaped="<code>")
div(unescaped!="<code>")
```

```
<div escaped="&lt;code&gt;"></div>
<div unescaped="<code>"></div>
```

### Caution

**Unescaped buffered code can be dangerous.** You must be sure to sanitize any user inputs to avoid cross-site scripting.

## Boolean Attributes ¶

Boolean attributes are mirrored by Pug. Boolean values ( `true` and `false` ) are accepted. When no value is specified `true` is assumed.

```
input(type='checkbox' checked)
|
|
```



```
input(type='checkbox' checked=true)
|
|
input(type='checkbox' checked=false)
|
|
input(type='checkbox' checked=true.toString())
```

```
<input type="checkbox" checked="checked" />
<input type="checkbox" checked="checked" />
<input type="checkbox" />
<input type="checkbox" checked="true" />
```

If the doctype is `html`, Pug knows not to mirror the attribute, and instead uses the terse style (understood by all browsers).

```
doctype html
|
|
input(type='checkbox' checked)
|
|
input(type='checkbox' checked=true)
|
|
input(type='checkbox' checked=false)
|
|
input(type='checkbox' checked=true && 'checked')
```

```
<!DOCTYPE html>
<input type="checkbox" checked>
<input type="checkbox" checked>
<input type="checkbox">
<input type="checkbox" checked="checked">
```

## Style Attributes ¶

The `style` attribute can be a string, like any normal attribute; but it can also be an object, which is handy when styles are generated by JavaScript.

```
a(style={color: 'red', background: 'green'})
```

```
<a style="color:red;background:green;"></a>
```

## Class Attributes ¶

The `class` attribute can be a string, like any normal attribute; but it can also be an array of class names, which is handy when generated from JavaScript.

```
- var classes = ['foo', 'bar', 'baz']
a(class=classes)
|
|
|
// - the class attribute may also be repeated to merge arrays
a.bang(class=classes class=['bing'])
```

```
<a class="foo bar baz"></a>
<a class="bang foo bar baz bing"></a>
```

It can also be an object which maps class names to `true` or `false` values. This is useful for applying conditional classes

```
- var currentUrl = '/about'
a(class={active: currentUrl === '/' } href='/') Home
|
|
|
a(class={active: currentUrl === '/about' } href='/about') Abc
```

```
<a href="/">Home</a>
<a class="active" href="/about">About</a>
```

## Class Literal ¶

Classes may be defined using a `.classname` syntax:

```
a.button
```

```
<a class="button"></a>
```

Since `div` 's are such a common choice of tag, it is the default if you omit the tag name:

```
.content
```

```
<div class="content"></div>
```

## ID Literal ¶

IDs may be defined using a `#idname` syntax:

```
a#main-link
```

```
<a id="main-link"></a>
```

Since `div` 's are such a common choice of tag, it is the default if you omit the tag name:

```
#content
```

```
<div id="content"></div>
```

## &attributes ¶

Pronounced as “and attributes”, the `&attributes` syntax can be used to explode an object into attributes of an element.

```
div#foo(data-bar="foo")&attributes({'data-foo': 'bar'})
```

```
<div id="foo" data-bar="foo" data-foo="bar"></div>
```

The above example uses an object literal. But you can also use a variable whose value is an object, too. (See also: [Mixin Attributes](#)).

```
- var attributes = {};  
- attributes.class = 'baz';
```

```
div#foo(data-bar="foo")&attributes(attributes)
```

```
<div class="baz" id="foo" data-bar="foo"></div>
```

### Caution

Attributes applied using `&attributes` are not automatically escaped. You must be sure to sanitize any user inputs to avoid cross-site scripting (XSS). If passing in `attributes` from a mixin call, this is done automatically.

### Get an update when we release new features

Name (optional):

Email:

Sign Up

We respect your [email privacy](#).