HAML

Overview

XHTML Abstraction Markup Language (HAML) is designed to be quick and easy to create and read.

Typical HTML:

<html>

<head>

<title>Sativaware : Introduction to HAML</title>

</head>

<body>

Haml is a markup language ...

</body>

</html>

Typical HAML:

%html

%head

%title

Sativaware : Introduction to HAML

%body

Haml is a markup language ...

Install

Add the HAML gem to the project Gemfile:

gem 'haml'

Elements

HAML removes opening and closing tags and makes use of indentation for nesting elements.

To add an element a percentage symbol is prepended to its name, for example to add a body element:

%body

Since div tags are so common, they can be left out and HAML will default to %div.

Classes and IDs

To add class or id to an element, simply chain them onto the end of element tag, for example:

%div.classname.secondclass#id-for-div

Attributes

Add element attributes in a tailing hash:

%tagname{:attr1 => 'value1', :attr2 => 'value2'}

Data attributes can be added in a nested hash inside the data attribute key:

%div{:data => {:attribute => 'foo'}}

Standard HTML syntax for attributes can still be used:

%a(href='#' title='bar')

Ruby

Since HAML is compiled using Ruby, any code in ruby syntax can be added as normal. For example, adding a variable id in as a element id:

.item{id: "item#{@item.id}"}

or just interpolating into an element:

%p Your highest scoring game is #{best\_game}

Using the equals symbol will tell HAML to evaluate code on the Ruby code on the right. For example:

%a{href:

= @item.name

To run Ruby code without it rendering simply use a dash:

- books.shuffle.each\_with\_index do |book, index|

# render something to do with books

- end

Escaping HTML

HTML sensitive symbols (&, /, :) can be escaped into their respective encoded equivalents using the ampersand:

%p

&= "Yes & yes"

=> 'Yes &amp; yes'

HTML can be unescaped using the bang symbol:

%p

!= "This is how you write a paragraph tag <p></p>"

=>'This is how you write a paragraph tag <p></p>'

Filters

Filters can be used to pass code to another compiling program and output its result. For example, rendering markdown can be performed by adding markdown filter before any markdown:

:markdown

# Header

Text \*\*inside\*\* the \*block\*

While you can create custom filters using Haml::Filters, there are many different types of filter included, such as:

* :markdown
* :javascript
* :coffee
* :css
* :ruby
* etc