Haml is a markup language predominantly used with Ruby that cleanly and simply describes the HTML of any web document without the use of inline code. It is a popular alternative to using Rails templating language (.erb) and allows you to embed Ruby code into your markup.

It aims to reduce repetition in your markup by closing tags for you based on the structure of the indents in your code. The result is markup that is well-structured, DRY, logical, and easier to read.

You can also use Haml on a project independent of Ruby, by installing the Haml gem on your machine and using the command line to convert it to html.

$ haml input_file.haml output_file.html

```
/ -------------------------------------------
/ Indenting
/ -------------------------------------------

/
  Because of the importance indentation has on how your code is rendered, the
  indents should be consistent throughout the document. Any differences in
  indentation will throw an error. It's common-practice to use two spaces,
  but it's really up to you, as long as they're constant.


/ -------------------------------------------
/ Comments
/ -------------------------------------------

/ This is what a comment looks like in Haml.

/
  To write a multi line comment, indent your commented code to be
  wrapped by the forward slash

-# This is a silent comment, which means it wont be rendered into the doc at all


/ -------------------------------------------
/ Html elements
/ -------------------------------------------

/ To write your tags, use the percent sign followed by the name of the tag
%body
  %header
    %nav

/ Notice no closing tags. The above code would output
  <body>
    <header>
      <nav></nav>
    </header>
  </body>

/ The div tag is the default element, so they can be written simply like this
.foo
```

```
/ To add content to a tag, add the text directly after the declaration
%h1 Headline copy

/ To write multiline content, nest it instead
%p
  This is a lot of content that we could probably split onto two
  separate lines.

/
  You can escape html by using the ampersand and equals sign ( &= ). This
  converts html-sensitive characters (&, /, :) into their html encoded
  equivalents. For example

%p
  &= "Yes & yes"

/ would output 'Yes &amp; yes'

/ You can unescape html by using the bang and equals sign ( != )
%p
  != "This is how you write a paragraph tag <p></p>"

/ which would output 'This is how you write a paragraph tag <p></p>'

/ CSS classes can be added to your tags either by chaining .classnames to the tag
%div.foo.bar

/ or as part of a Ruby hash
%div{:class => 'foo bar'}

/ Attributes for any tag can be added in the hash
%a{:href => '#', :class => 'bar', :title => 'Bar'}

/ For boolean attributes assign the value 'true'
%input{:selected => true}

/ To write data-attributes, use the :data key with its value as another hash
%div{:data => {:attribute => 'foo'}}


/ -------------------------------------------
/ Inserting Ruby
/ -------------------------------------------

/
  To output a Ruby value as the contents of a tag, use an equals sign followed
  by the Ruby code

%h1= book.name

%p
  = book.author
  = book.publisher
```

```
/ To run some Ruby code without rendering it to the html, use a hyphen instead
- books = ['book 1', 'book 2', 'book 3']

/ Allowing you to do all sorts of awesome, like Ruby blocks
- books.shuffle.each_with_index do |book, index|
  %h1= book

  if book do
    %p This is a book

/ Adding ordered / unordered list
%ul
  %li
    =item1
    =item2

/
  Again, no need to add the closing tags to the block, even for the Ruby.
  Indentation will take care of that for you.

/ -------------------------------------------
/ Inserting Table with bootstrap classes
/ -------------------------------------------

%table.table.table-hover
  %thead
    %tr
      %th Header 1
      %th Header 2

    %tr
      %td Value1
      %td value2

  %tfoot
    %tr
      %td
        Foot value


/ -------------------------------------------
/ Inline Ruby / Ruby interpolation
/ -------------------------------------------

/ Include a Ruby variable in a line of plain text using #{}
%p Your highest scoring game is #{best_game}


/ -------------------------------------------
/ Filters
/ -------------------------------------------

/
```

```
  Use the colon to define Haml filters, one example of a filter you can
  use is :javascript, which can be used for writing inline js

:javascript
  console.log('This is inline <script>');
```

Additional resources

- What is HAML? - A good introduction that does a much better job of explaining the benefits of using HAML.
- Official Docs - If you'd like to go a little deeper.