lustache - Logic-less {{mustache}} templates with Lua

What could be more logical awesome than no logic at all?
lustache is an implementation of the mustache template system in Lua.

Mustache is a logic-less template syntax. It can be used for HTML, config files, source code - anything. It works by expanding tags in a template using values provided in a hash or object.

We call it "logic-less" because there are no if statements, else clauses, or for loops. Instead there are only tags. Some tags are replaced with a value, some nothing, and others a series of values.

For a language-agnostic overview of mustache's template syntax, see the mustache(5) manpage.

Where to use lustache?

You can use lustache to render mustache templates anywhere you can use Lua.

lustache exposes itself as a module, so you only have to require the file. and assign it.

Usage

Installation

Download lustache.lua and place it in your project, or install it with luarocks using luarocks install lustache. On OSX, you can brew install luarocks. Below is quick example how to use lustache:

```
local lustache = require "lustache"

view_model = {
   title = "Joe",
   calc = function ()
      return 2 + 4
   end
}

output = lustache:render("{{title}} spends {{calc}}", view_model)
```

In this example, the lustache:render function takes two parameters: 1) the mustache template and 2) a view_model object that contains the data and code needed to render the template.

Running Tests

Lustache uses the busted testing framework.

Run luarocks make, then busted spec. Install busted through luarocks install busted.

Templates

A mustache template is a string that contains any number of mustache tags. Tags are indicated by the double mustaches that surround them. {{person}} is a tag, as is {{#person}}. In both examples we refer to person as the tag's key. There are several types of tags available in lustache.

Variables

The most basic tag type is a simple variable. A {{name}} tag renders the value of the name key in the current context. If there is no such key, nothing is rendered.
All variables are HTML-escaped by default. If you want to render unescaped HTML, use the triple mustache: {{{name}}}. You can also use & to unescape a variable.
Template:

```
* {{name}}
* {{age}}
* {{company}}
* {{{company}}}
* {{&company}}
```

View:

```
{
   name = "Chris",
   company = "<b>GitHub</b>"
}
```

Output:

```
* Chris
*
* &lt;b&gt;GitHub&lt;/b&gt;
* <b>GitHub</b>
```

```
* <b>GitHub</b>
```

Dot notation may be used to access keys that are properties of objects in a view.

Template:

```
* {{name.first}} {{name.last}}
* {{age}}
```

View:

```
{
   name = {
      first = "Michael",
      last = "Jackson"
   },
   age = "RIP"
}
```

Output:

```
* Michael Jackson
* RIP
```

Sections

Sections render blocks of text one or more times, depending on the value of the key in the current context.

A section begins with a pound and ends with a slash. That is, `{{#person}}` begins a `person` section, while `{{/person}}` ends it. The text between the two tags is referred to as that section's "block".
The behavior of the section is determined by the value of the key.

False Values or Empty Lists

If the `person` key exists and has a value of `nil` or `false`, or is an empty list, the block will not be rendered.
Template:

```
Shown.
{{#person}}
Never shown!
{{/person}}
```

View:

```
{
```

```
  person = false
}
```

Output:

Shown.

Non-Empty Lists

If the person key exists and is not nil or false, and is not an empty list the block will be rendered one or more times.
When the value is a list, the block is rendered once for each item in the list. The context of the block is set to the current item in the list for each iteration. In this way we can loop over collections.

Template:

```
{{#stooges}}
<b>{{name}}</b>
{{/stooges}}
```

View:

```
{
  stooges = {
    { name = "Moe" },
    { name = "Larry" },
    { name = "Curly" }
  }
}
```

Output:

```
<b>Moe</b>
<b>Larry</b>
<b>Curly</b>
```

When looping over an array of strings, a . can be used to refer to the current item in the list.
Template:

```
{{#musketeers}}
* {{.}}
{{/musketeers}}
```

View:

```
{
  musketeers = { "Athos", "Aramis", "Porthos", "D'Artagnan" }
}
```

Output:

```
* Athos
* Aramis
* Porthos
* D'Artagnan
```

If the value of a section variable is a function, it will be called in the context of the current item in the list on each iteration.

Template:

```
{{#beatles}}
* {{name}}
{{/beatles}}
```

View:

```
{
   beatles = {
      { first_name = "John", last_name = "Lennon" },
      { first_name = "Paul", last_name = "McCartney" },
      { first_name = "George", last_name = "Harrison" },
      { first_name = "Ringo", last_name = "Starr" }
   },
   name = function (self)
      return self.first_name .. " " .. self.last_name
   end
}
```

Output:

```
* John Lennon
* Paul McCartney
* George Harrison
* Ringo Starr
```

Functions

If the value of a section key is a function, it is called with the section's literal block of text, un-rendered, as its first argument. The second argument is a special rendering function that uses the current view as its view argument.

Template:

```
{{#bold}}Hi {{name}}.{{/bold}}
```

View:

```
{
   name = "Tater",
   bold = function (text, render)
         return "<b>" .. render(text) .. "</b>"
   end
}
```

Output:

```
<b>Hi Tater.</b>
```

Inverted Sections

An inverted section opens with `{{^section}}` instead of `{{#section}}`. The block of an inverted section is rendered only if the value of that section's tag is `nil`, `false`, or an empty list.

Template:

```
{{#repos}}<b>{{name}}</b>{{/repos}}
{{^repos}}No repos :({{/repos}}
```

View:

```
{
   repos = {}
}
```

Output:

```
No repos :(
```

Comments

Comments begin with a bang and are ignored. The following template:

```
<h1>Today{{! ignore me }}.</h1>
```

Will render as follows:

```
<h1>Today.</h1>
```

Comments may contain newlines.

Partials

Partials begin with a greater than sign, like {{> box}}.

Partials are rendered at runtime (as opposed to compile time), so recursive partials are possible. Just avoid infinite loops.

They also inherit the calling context. Whereas in ERB you may have this:

```
<%= partial :next_more, :start => start, :size => size %>
```

Mustache requires only this:

```
{{> next_more}}
```

Why? Because the `next_more.mustache` file will inherit the `size` and `start` variables from the calling context. In this way you may want to think of partials as includes, or template expansion, even though it's not literally true.
For example, this template and partial:

```
base.mustache:
<h2>Names</h2>
{{#names}}
  {{> user}}
{{/names}}

user.mustache:
<strong>{{name}}</strong>
```

Can be thought of as a single, expanded template:

```
<h2>Names</h2>
{{#names}}
  <strong>{{name}}</strong>
{{/names}}
```

In lustache an object of partials may be passed as the third argument to `lustache:render`. The object should be keyed by the name of the partial, and its value should be the partial text.

Set Delimiter

Set Delimiter tags start with an equals sign and change the tag delimiters from `{{` and `}}` to custom strings.
Consider the following contrived example:

```
* {{ default_tags }}
{{=<% %>=}}
* <% erb_style_tags %>
<%={{ }}=%>
* {{ default_tags_again }}
```

Here we have a list with three items. The first item uses the default tag style, the second uses ERB style as defined by the Set Delimiter tag, and the third returns to the default style after yet another Set Delimiter declaration.

According to ctemplates, this "is useful for languages like TeX, where double-

braces may occur in the text and are awkward to use for markup."

Custom delimiters may not contain whitespace or the equals sign.

Testing

lustache uses the lunit testing framework. In order to run the tests you'll need to install lunit, which can be done through luarocks or your choice of lua package manager.

```
$ luarocks install lunit
```

Then run the tests.

```
$ lunit spec/*
```

Thanks

lustache began as a direct port of Jan Lehnardt's excellent mustache.js. It would be significantly further behind without the available code from the many contributors. <3

License