EJS cheatsheet

EJS(Embedded JavaScript) is a simple templating language that lets you generate HTML markup with plain JavaScript.

Get Started

Hello world

install

```
npm install ejs
```

hello.ejs

```
<% if (user.email) { %>
  <h1><%= user.email %></h1>
<% } %>
```

CLI

```
$ ejs hello.ejs -o hello.html
```

Render with Data

```
let ejs = require('ejs');

let people = ['geddy', 'neil', 'alex'];
let tpl = '<%= people.join(", "); %>';

let html = ejs.render(tpl, {people: people}
console.log(html);
```

Pass EJS a template string and some data.

Browser Support

```
<script src="ejs.js"></script>
<script>
  let people = ['geddy', 'neil', 'alex'];
  let html = ejs.render('<%= people.join(",
</script>
```

Use ejs in a script tag.

Variables

`<%= var %>`	Prints the value of the variable
`<%- var %>`	Prints without HTML escaping

CLI

Render and specify an output file.

```
$ ejs hello.ejs -o hello.html
```

Feed it a template file and a data file

```
$ ejs hello.ejs -f data.json -o hello.html
```

Comments

```
<%# This line will denote a comment %>
```

```
<%# This is a multi-line EJS comment.
    It can span multiple lines,
    but will not be displayed
    in the final HTML output.
%>
```

Method

```
let ejs = require('ejs');
let template = ejs.compile(str, options);

template(data);
// => Rendered HTML string

ejs.render(str, data, options);
// => Rendered HTML string

ejs.renderFile(filename, data, options, fun
    // str => Rendered HTML string
});
```

Including Files

```
<%- include('partials/navbar.ejs') %>
```

Include a template with data:

```
<% include('header', { title: 'My Page' })
```

```
<ul>
  <% users.forEach(function(user){ %>
    <%- include('item', {user: user}); %>
  <% }); %>
</ul>
```

To include a template, needs a file name option, paths are relative

Docs

Conditionals

```
<% if (userLoggedIn) { %>
  <p>Welcome, <%= username %>!</p>
<% } else { %>
  <p>Please log in.</p>
<% } %>
```

Using loops

```
<% if (userLoggedIn) { %>
  <p>Welcome, <%= username %>!</p>
<% } else { %>
  <p>Please log in.</p>
<% } %>
```

Custom delimiters

```
let ejs = require('ejs'),
    users = ['geddy', 'neil', 'alex'];

// Just one template
ejs.render('<?= users.join(" | "); ?>',
    {users: users},
    {delimiter: '?'});
// => 'geddy | neil | alex'
```

```
// Or globally
ejs.delimiter = '$';
ejs.render('<$= users.join(" | "); $>',
    {users: users});
// => 'geddy | neil | alex'
```

Caching

```
let ejs = require('ejs'),
LRU = require('lru-cache');

// LRU cache with 100-item limit
ejs.cache = LRU(100);
```

Custom file loader

```
let ejs = require('ejs');
let myFileLoader = function (filePath) {
  return 'myFileLoader: ' + fs.readFileSync
};

ejs.fileLoader = myFileLoader;
```

Layouts

```
<%- include('header'); -%>
<h1>
  Title
</h1>
<p>
  My page
</p>
<%- include('footer'); -%>
```

Client-side support

Example

```
<div id="output"></div>
<script src="ejs.min.js"></script>
<script>
  let people = ['geddy', 'neil', 'alex'],
      html = ejs.render('<%= people.join(", "); %>', {people: people
  // With jQuery:
  $('#output').html(html);
  // Vanilla JS:
  document.getElementById('output').innerHTML = html;
</script>
```

Caveats

```
let str = "Hello <%= include('file', {person: 'John'}); %>",
      fn = ejs.compile(str, {client: true});

fn(data, null, function(path, d){ // include callback
  // path -> 'file'
  // d -> {person: 'John'}
  // Put your code here
  // Return the contents of file as a string
}); // returns rendered string
```

Options

Options list

cache	Compiled functions are cached, requires filename
filename	Used by cache to key caches, and for includes
root	Set project root for includes with an absolute path (e.g., /file.ejs). Can be an array to try to resolve include from multiple directories.
views	An array of paths to use when resolving includes with relative paths.
context	Function execution context
compileDebug	When false, no debug instrumentation is compiled
client	Returns standalone compiled function
delimiter	Character to use for inner delimiter, by default '%'
openDelimiter	Character to use for opening delimiter, by default '<'
closeDelimiter	Character to use for closing delimiter, by default '>'
debug	Outputs generated function body
strict	When set to `true`, generated function is in strict mode
_with	Whether or not to use with() {} constructs. If false, then the locals will be stored in the locals object. (Implies `--strict`)
localsName	Name to use for the object storing local variables when not using with Defaults to locals
rmWhitespace	Remove all safe-to-remove whitespace, including leading and trailing whitespace. It also enables a safer version of -%> line slurping for all scriptlet tags (it does not strip new lines of tags in the middle of a line).
escape	The escaping function used with <%= construct. It is used in rendering and is .toString()ed in the generation of client functions. (By default escapes XML).
outputFunctionName	Set to a string (e.g., 'echo' or 'print') for a function to print output inside scriptlet tags.
async	When true, EJS will use an async function for rendering. (Depends on async/await support in the JS runtime.

Tags

Tags list

<%	'Scriptlet' tag, for control-flow, no output
<%_	'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
<%=	Outputs the value into the template (HTML escaped)
<%-	Outputs the unescaped value into the template
<%#	Comment tag, no execution, no output
<%%	Outputs a literal '<%'
%>	Plain ending tag
-%>	Trim-mode ('newline slurp') tag, trims following newline
_%>	'Whitespace Slurping' ending tag, removes all whitespace after it

Cli

Cli list

cache	Compiled functions are cached, requires filename
-o / --output-file FILE	Write the rendered output to FILE rather than stdout.
-f / --data-file FILE	Must be JSON-formatted. Use parsed input from FILE as data for rendering.
-i / --data-input STRING	Must be JSON-formatted and URI-encoded. Use parsed input from STRING as data for rendering.
-m / --delimiter CHARACTER	Use CHARACTER with angle brackets for open/close (defaults to %).
-p / --open-delimiter CHARACTER	Use CHARACTER instead of left angle bracket to open.
-c / --close-delimiter CHARACTER	Use CHARACTER instead of right angle bracket to close.
-s / --strict	When set to `true`, generated function is in strict mode
-n / --no-with	Use 'locals' object for vars rather than using `with` (implies --strict).
-l / --locals-name	Name to use for the object storing local variables when not using `with`.
-w / --rm-whitespace	Remove all safe-to-remove whitespace, including leading and trailing whitespace.
-d / --debug	Outputs generated function body
-h / --help	Display this help message.
-V/v / --version	Display the EJS version.

Examples of use :

```
$ ejs -p [ -c ] ./template_file.ejs -o ./output.html
$ ejs ./test/fixtures/user.ejs name=Lerxst
$ ejs -n -l _ ./some_template.ejs -f ./data_file.json
```