EJS (Embedded JavaScript) Cheatsheet

A quick reference guide covering essential EJS syntax, tags, control flow, includes, data handling, and best practices for generating dynamic HTML with plain JavaScript.

EJS Basics & Output

Core EJS Tags

`<% code %>`	Execute JavaScript code. No output.
`<%= output %>`	Output the value of `output`, escaped (HTML safe).
`<%- output %>`	Output the raw, unescaped value of `output`.
`<%# comment %>`	EJS comment. Not rendered in the output HTML.
`<%@ include %>`	Include a partial template (older syntax, `<%- include(...) %>` is preferred).
`<%_ code _%>`	Execute JavaScript code, stripping leading whitespace.
`<%-= output -%>`	Output raw value, stripping trailing whitespace.

Escaped vs Raw Output

Use `<%= %>` by default to prevent Cross-Site Scripting (XSS).

```
<p>Escaped: <%= '<b>Bold Text</b>' %></p>
```

Output:

```
<p>Escaped: &lt;b&gt;Bold Text&lt;/b&gt;</p>
```

Use `<%- %>` when you are sure the content is safe HTML or you specifically need unescaped output.

```
<p>Raw: <%- '<b>Bold Text</b>' %></p>
```

Output:

```
<p>Raw: <b>Bold Text</b></p>
```

Be extremely cautious with user-supplied data in `<%- %>`.

Control Flow & Includes

Conditional Statements

Use `<% %>` for `if`, `else if`, `else` blocks.

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

Ensure curly braces `{}` are within the `<% %>` tags.

You can spread the logic across multiple EJS tags.

```
<% if (items.length > 0) { %>
  <h2>Items:</h2>
<% } %>
```

Loops

Iterate over arrays using `for`, `forEach`, or `while`.

Using `forEach`:

```
<ul>
<% items.forEach(function(item) { %>
  <li><%= item %></li>
<% }); %>
</ul>
```

Using a standard `for` loop:

```
<ul>
<% for (var i = 0; i < items.length; i++) { %>
  <li>Item #<%= i + 1 %>: <%= items[i] %></li>
<% } %>
</ul>
```

Looping over object properties:

```
<% for (const key in obj) { %>
  <p><%= key %>: <%= obj[key] %></p>
<% } %>
```

Remember to close your loop tags (`<% }); %>` or `<% } %>`).

Includes

Include other EJS files (partials) using `<%- include('path/to/partial') %>`.

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

<main>
  <h1><%= title %></h1>
  <p>Main content goes here.</p>
</main>

<footer>
  <%- include('./partials/footer', { year: new Date().getFullYear() }) %>
</footer>
```

The `include` path is relative to the EJS `views` directory option, or relative to the current file if the `includer` option is set.

You can pass local variables to the included partial as a second argument (an object).

In `footer.ejs`:

```
<p>&copy; <%= year %> My Website</p>
```

Variables from the parent scope are also available in the partial unless locals are explicitly passed.

Data Handling & Features

Accessing Data

Data passed to `ejs.render()` or `res.render()` is available as variables directly in the template scope.

Given data `{ title: 'My Page', items: ['Apple', 'Banana'] }`

```
<h1><%= title %></h1>
<ul>
<% items.forEach(function(item) { %>
  <li><%= item %></li>
<% }); %>
</ul>
```

Nested object properties can be accessed using dot notation.

Given data `{ user: { name: 'Alice', id: 123 } }`

```
<p>User ID: <%= user.id %></p>
<p>Name: <%= user.name %></p>
```

Be mindful of potential `undefined` errors when accessing nested properties. Use conditional checks if needed.

Defining Variables

You can define variables within `<% %>` tags using `var`, `let`, or `const`.

```
<% let greeting = 'Hello, EJS!'; %>
<h1><%= greeting %></h1>
```

Variables defined this way are scoped to the current template or include.

Avoid complex logic within templates. Ideally, prepare data and variables in your server-side code before passing them to EJS.

Calling Functions

You can call functions defined in the data object passed to EJS, or standard JavaScript functions.

Given data `{ formatName: function(user) { return user.firstName + ' ' + user.lastName; }, currentUser: { firstName: 'Bob', lastName: 'Smith' } }`

```
<p>Formatted Name: <%= formatName(currentUser) %></p>
```

Standard JS functions like `Date.now()`, `JSON.stringify()`, etc., are available.

```
<p>Current Timestamp: <%= Date.now() %></p>
```

Custom Delimiters

Default delimiters:	`<% %>`, `<%= %>`, `<%- %>`, `<%# %>`, `<%@ %>`
Change delimiters via options:	Pass `delimiter: '%'` option to `ejs.render()` or setup in framework (e.g., Express `app.set('view options', { delimiter: '%' })`). Example with delimiter `%`: `{% code %}`, `{% = output %}`, `{% - output %}`
Change open/close via options:	Pass `open: '['`, `close: ']''` option. Example with delimiters `[` and `]`: `[% code %]`, `[%= output %]`, `[%- output %]`

Tips & Best Practices

Error Handling

EJS errors (syntax errors, runtime JS errors) during rendering will typically throw exceptions in your server-side code.

Ensure you have proper `try...catch` blocks or error handling middleware (like in Express) to catch rendering errors and display a user-friendly error page.

Use the `debug` option in development (`ejs.render(template, data, { debug: true })`) to get detailed error messages with line numbers.

Check for the existence of variables or properties before accessing them if they might be missing.

```
<% if (typeof user !== 'undefined' && user.address) { %>
  <p>Address: <%= user.address %></p>
<% } %>
```

Performance Considerations

EJS compiles templates into JavaScript functions on the first render (per file).

In production, templates are cached by default. Ensure caching is enabled for performance.

Avoid complex, heavy computations within EJS templates. Perform data processing in your application logic before rendering.

Keep templates relatively small and use includes (`<%- include %>`) to break down large pages into manageable partials.

General Best Practices

Separation of Concerns: Keep presentation logic (HTML structure, loops, conditionals based on simple flags) in EJS templates and business logic (data fetching, complex calculations, database interactions) in your application code.

Pass Only Necessary Data: Don't dump your entire database object into the template context. Pass only the specific data required for that template to render.

Consistent Formatting: Use consistent indentation and spacing within EJS tags to improve readability.

Comments: Use `<%# %>` for template-specific notes that shouldn't appear in the final HTML.

Using EJS with Node.js/Express

Install EJS: `npm install ejs`

Set EJS as the view engine in Express:

```
const express = require('express');
const app = express();

app.set('view engine', 'ejs');
app.set('views', './views'); // Optional: specify views directory

// ... routes ...
```

Render a view from a route handler:

```
app.get('/', (req, res) => {
  res.render('index', { title: 'Home Page', message: 'Welcome!' });
});
```

Access passed data in `index.ejs`:

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <h1><%= message %></h1>
</body>
</html>
```