Building User Interfaces

Javascript
An Introduction

Professor Bilge Mutlu

Disclaimer

This is not a comprehensive introduction to JS, so below are links to great additional resources:

- MDN Web Docs
- DevDocs
- W3 Schools
- FreeCodeCamp

What we will learn today?

» History and overview of web programming

» Syntax, JS for Java developers

» Interacting with user-facing elements

TopHat Attendance

TopHat Questions

What we will you need?

- A modern web browser (developer tools enabled)
- A source-code editor (e.g., Visual Studio Code, Atom, Sublime Text)

A little bit of history

» JavaScript (JS) was developed by Netscape Communications (Brendan Eich) in 1995 to make the web more dynamic — a "glue language" for HTML — *Marc Andreessen*

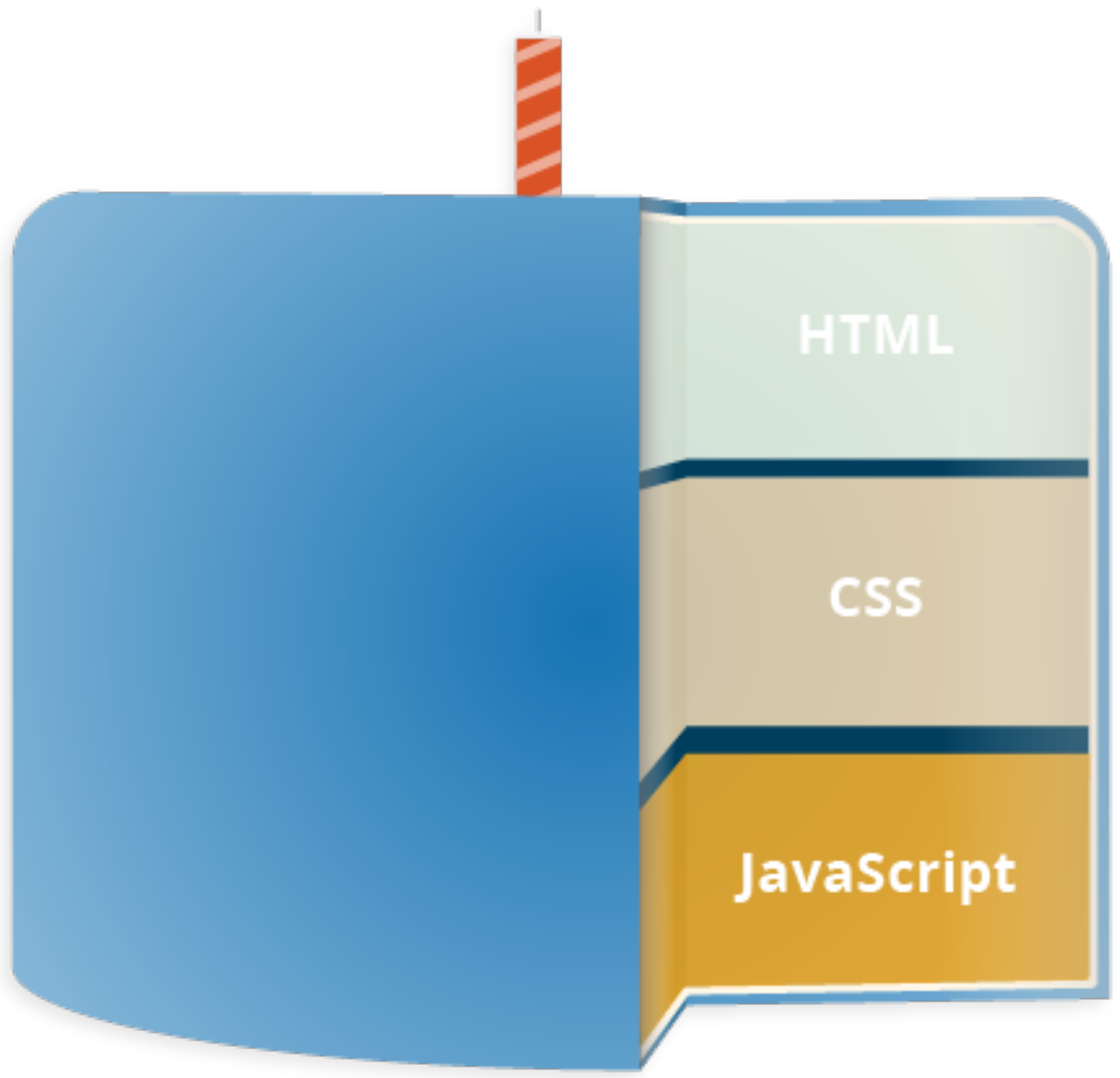
» Mocha > LiveScript > JavaScript / VBScript > JScript (Microsoft)

» Client-side and server-side JS (e.g., Node.js)

» Standardization through ECMAScript (ES)

How does the "front-end" of the web work?

A three-layered cake[1]



[1] The three layers of designing for the web

Let's see an example

Consider the following *very* simple HTML page:

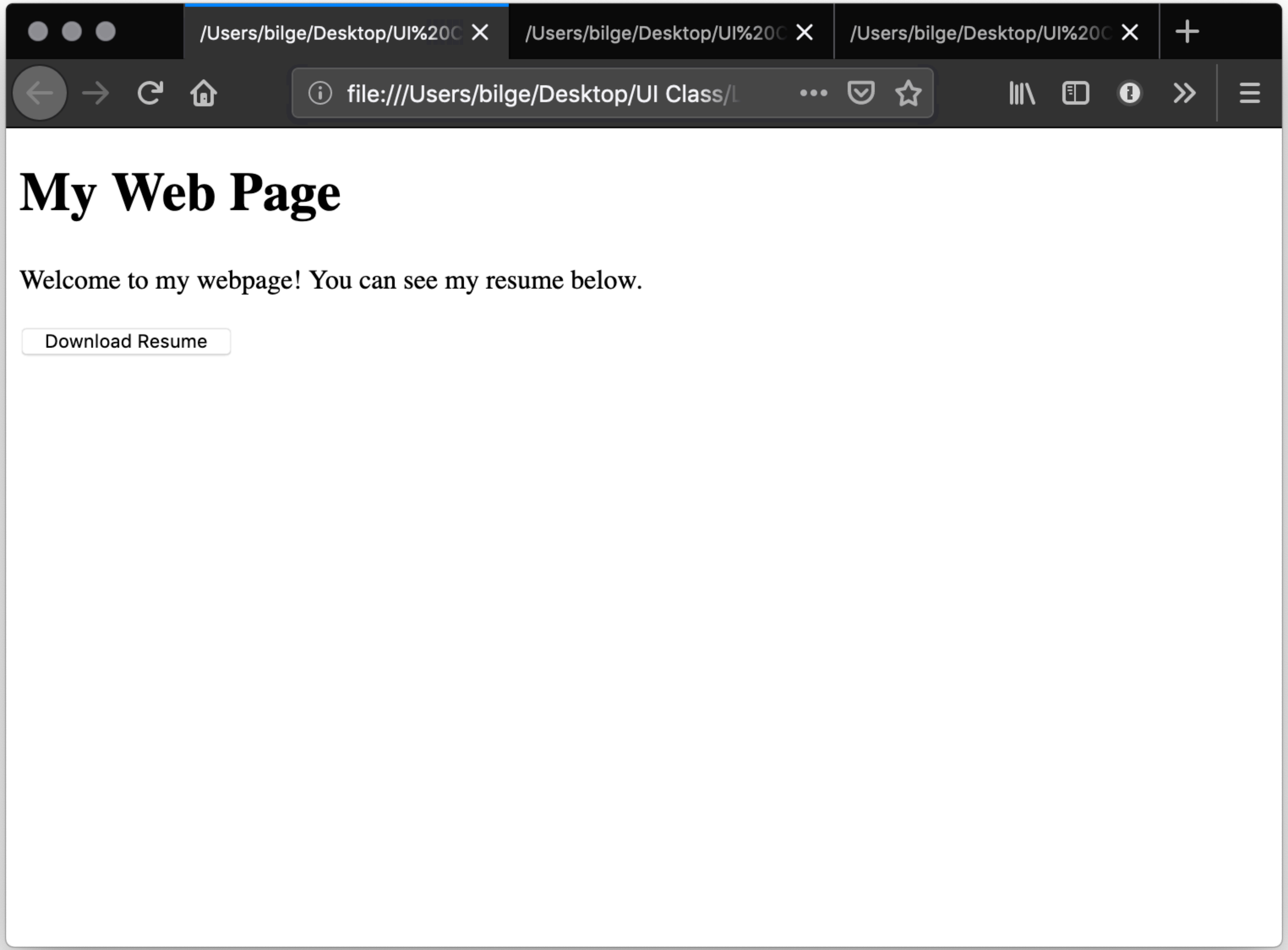
```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>My Web Page</h1>

<p>Welcome to my webpage! You can see my resume below.</p>

<button>Download Resume</button>

</body>
</html>
```


/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C

file:///Users/bilge/Desktop/UI Class/L

My Web Page

Welcome to my webpage! You can see my resume below.

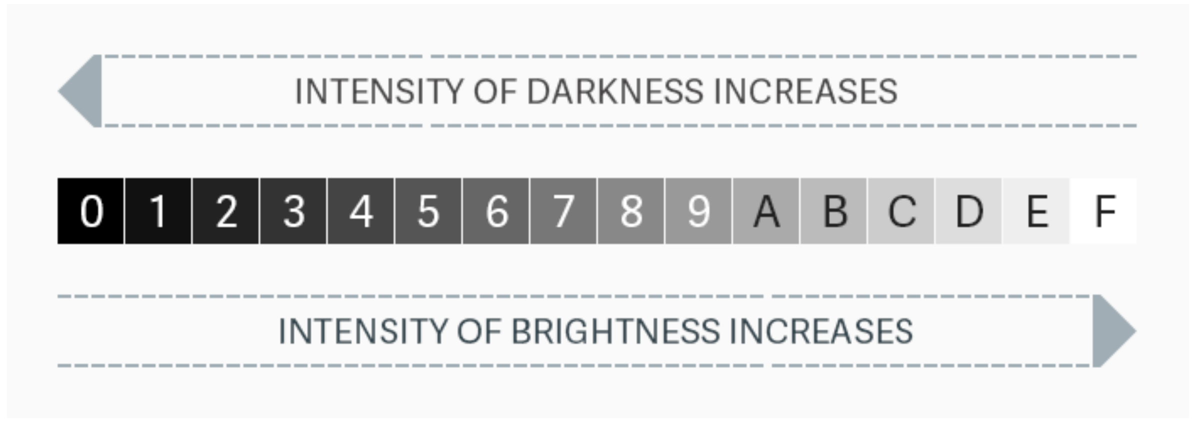
Download Resume

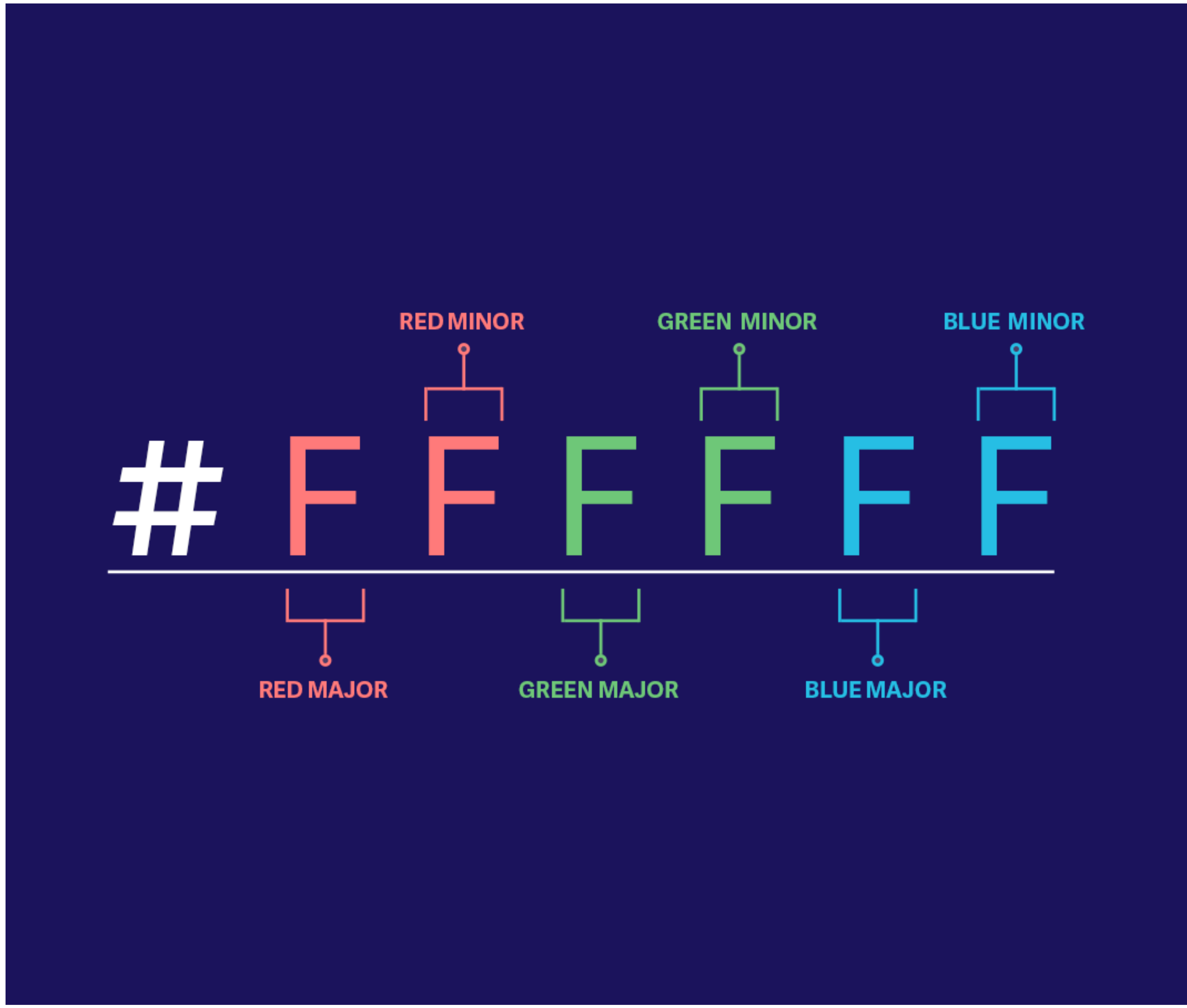
Let's improve its appearance. Within `head` and then `style`:

```
body {background-color: lightgrey;}
h1   {
    color: darkslategray;
    text-align: center;
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif}
p    {
    color: darkolivegreen;
    margin-left: 50px;
    margin-right: 50px;
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif}
button {
    background-color: darkolivegreen;
    border: none;
    color: white;
    padding: 15px 32px;
    text-align: center;
    display: inline-block;
    font-size: 16px;
    margin-left: 50px; margin-right: 50px;
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif
}
```

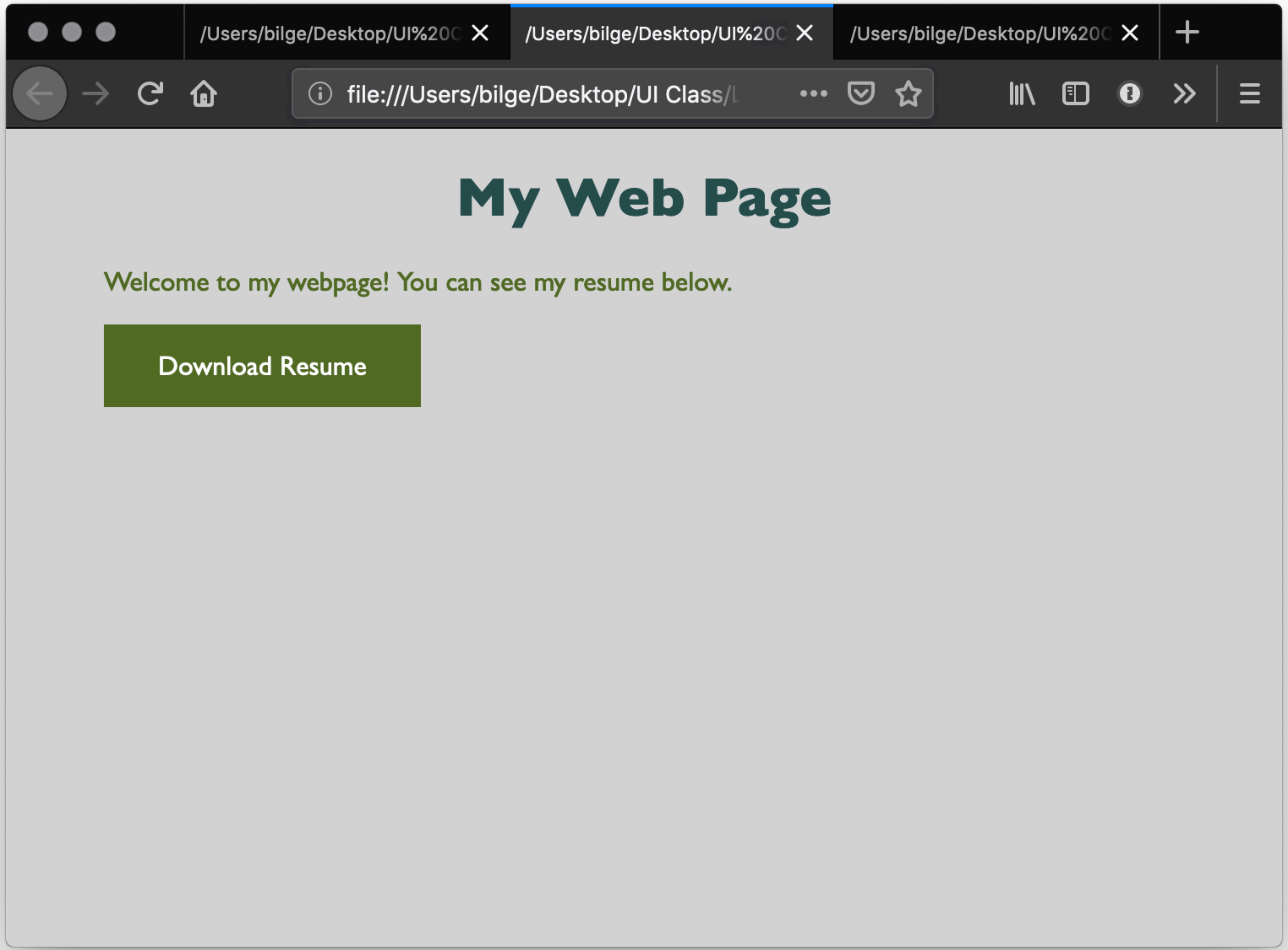
Detour: Specifying Color[2]

- RGB triplet, HEX triplet
- Majors > tone, minors > shade
- Values 0–9–A–F
- Search for "hex color"


INTENSITY OF DARKNESS INCREASES
0 1 2 3 4 5 6 7 8 9 A B C D E F
INTENSITY OF BRIGHTNESS INCREASES



[2] Nitish Khagwal



/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
file:///Users/bilge/Desktop/UI Class/L

My Web Page

Welcome to my webpage! You can see my resume below.

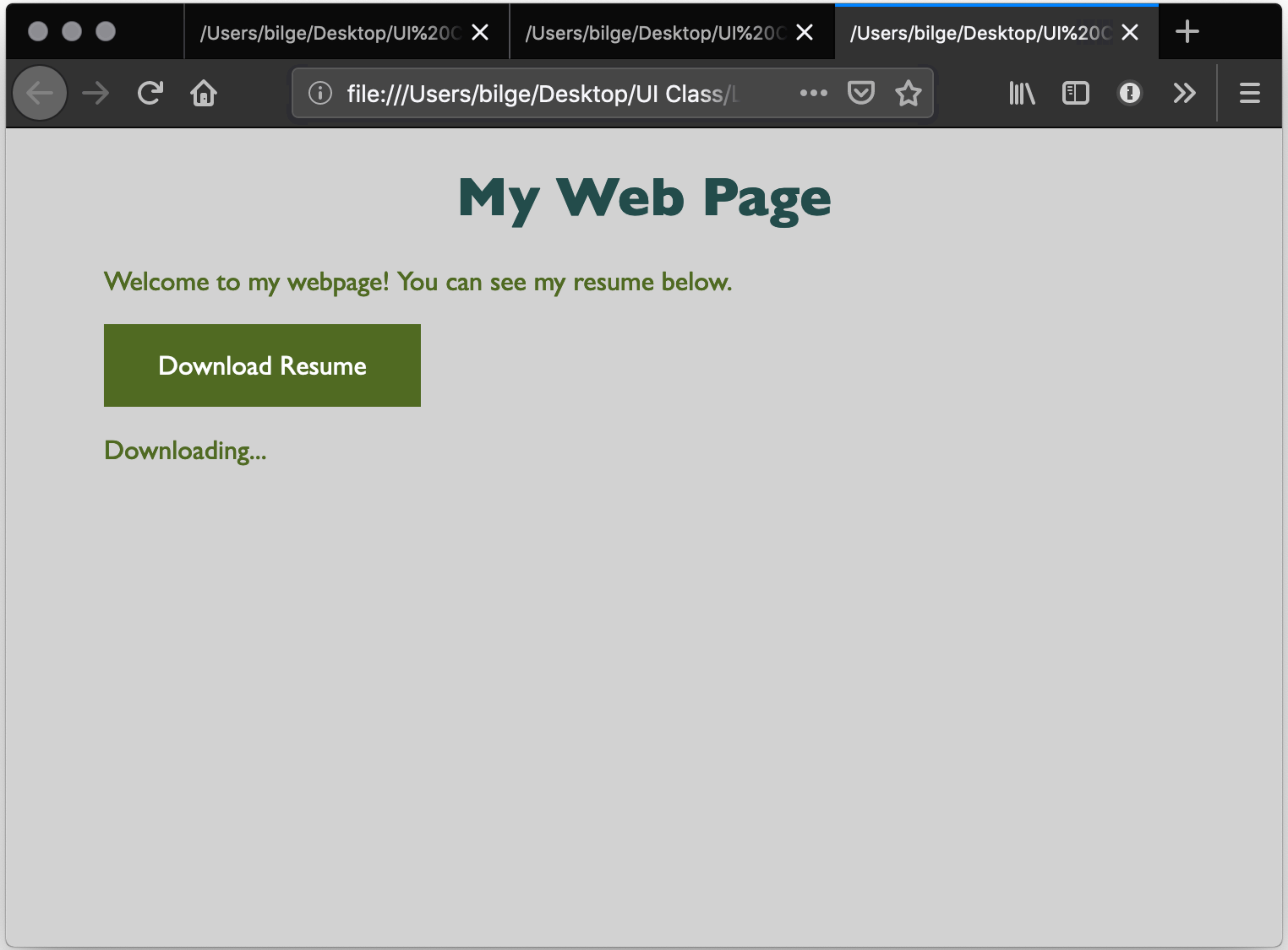
Let's add some *minor* interactivity. Within `head` and then `script`:

```
function myFunction() {
    document.getElementById("message").innerHTML = "Downloading...";
}
```

Then within `body`:

```
<button onclick="myFunction()">Download Resume</button>


<p id="message"></p>
```



/Users/bilge/Desktop/UI%20C

/Users/bilge/Desktop/UI%20C

/Users/bilge/Desktop/UI%20C

file:///Users/bilge/Desktop/UI Class/L

My Web Page

Welcome to my webpage! You can see my resume below.

Download Resume

Downloading...

How does JS interact with the page?

1. Internal JS
2. External JS
3. Inline JS handler

Internal JS

```html
<head>
    <script>
        // JS goes here
    </script>
</head>
```

External JS

Create a `script.js` file, which will contain your JS code, and include within `head`:

```
<script src="script.js" defer></script>
```

Internal JS handlers

```
<button onclick="myFunction()">Download Resume</button>
```

Pro Tips: Internal JS handlers result in inefficient and unorganized code. Different loading strategies are used for internal JS (listening for `DOMContentLoaded` event; including `script` after the page content) and external JS (`defer` attribute).

How is JS interpreted?

- All modern browsers have a JS engine, e.g., v8, SpiderMonkey[3]
- Node.js encompasses v8 within a C++-based environment to compile JS outside the browser[4]
- In this class, we will exclusively work within the browser environment.

---

[3] List of ECMAScript engines

[4] Node.js

How do I start JS development?

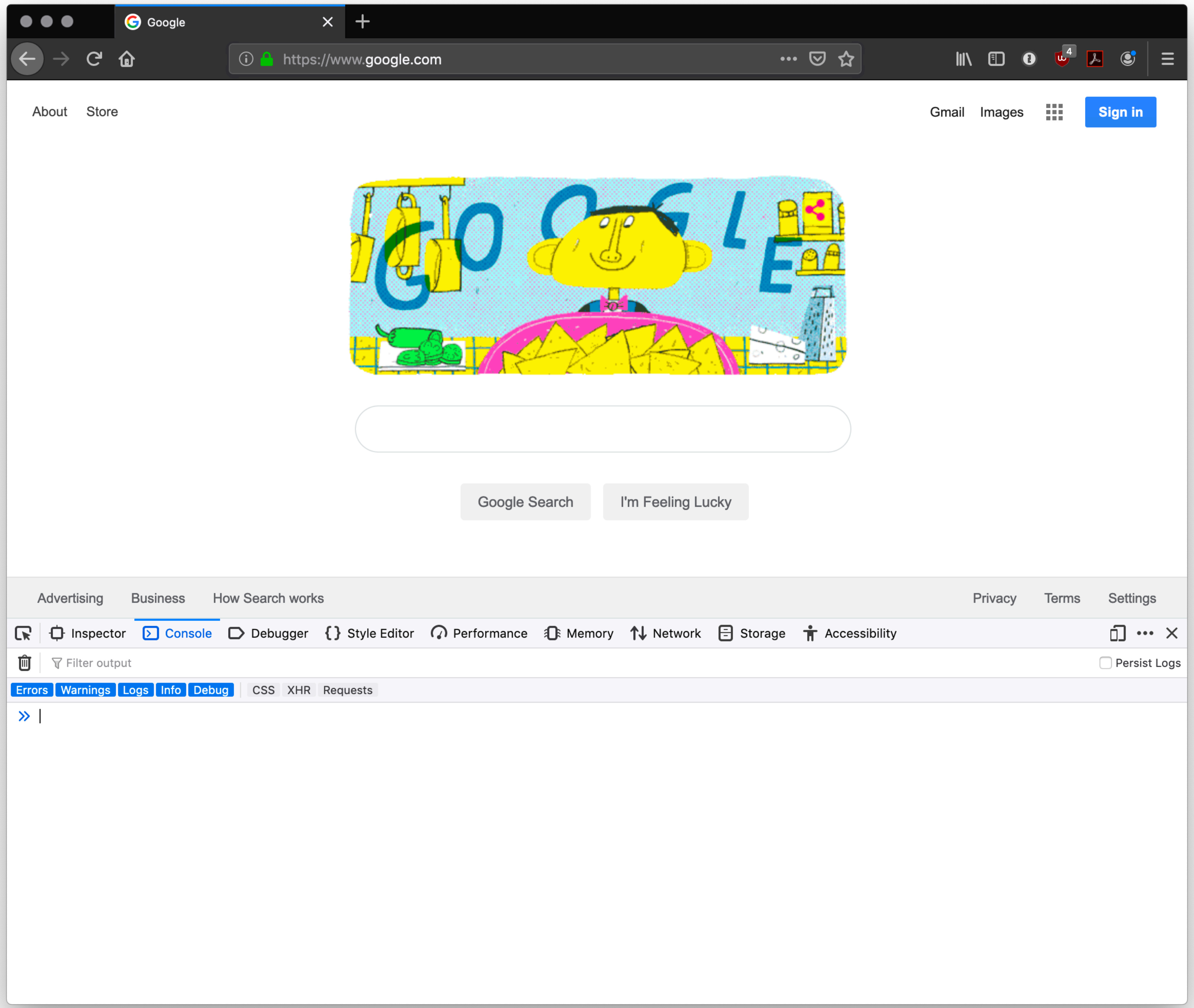
1. In the **browser** — best for testing ideas, code, etc.
2. In a **coding environment** — best for application development

Running JS in the browser

`Ctrl-Shift-K` or `Command-Option-K`

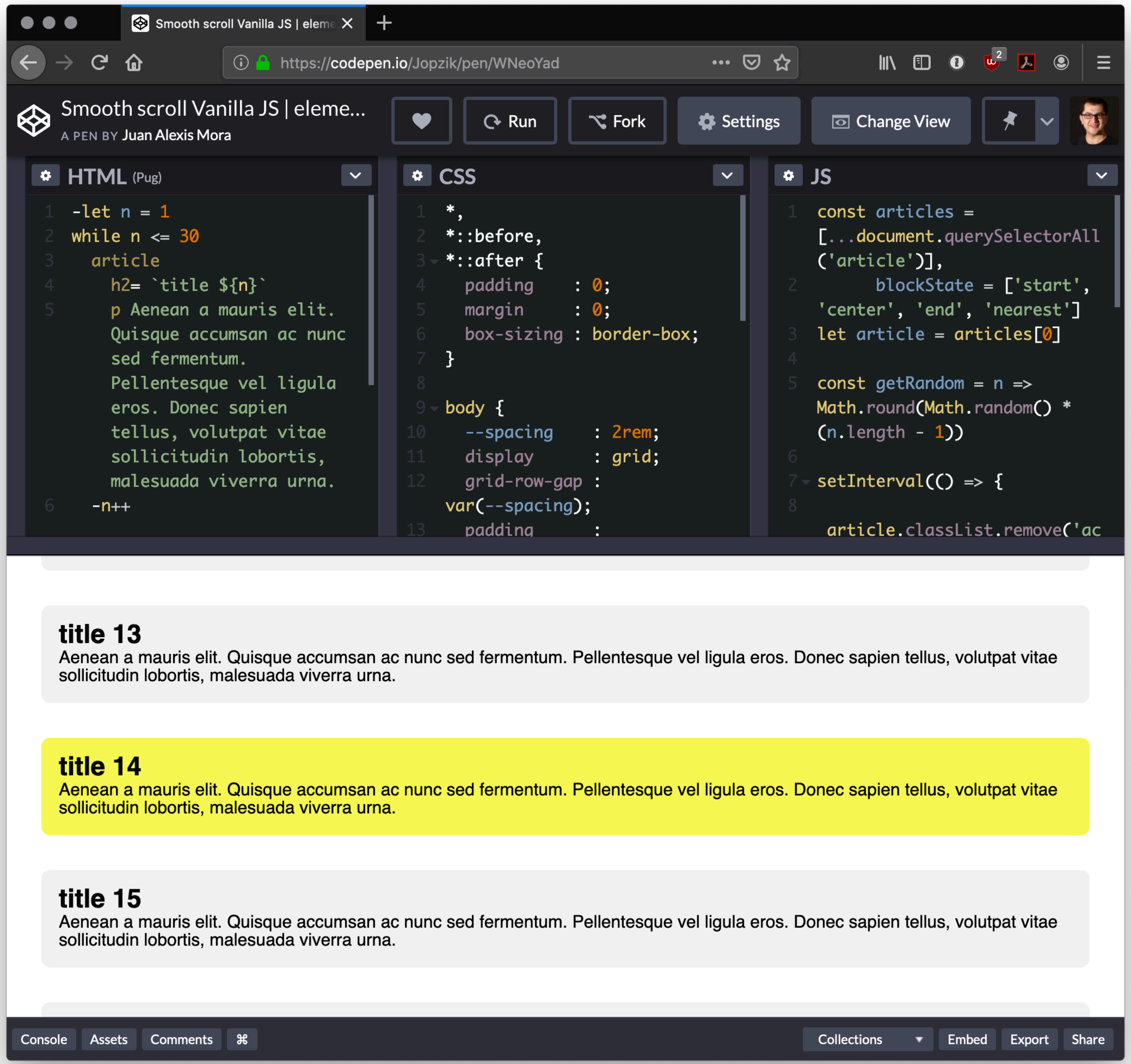
Try out:

```
console.log("On Wisconsin!")
```



Running JS in an online sandbox

- » https://codepen.io/
- » https://codesandbox.io/
- » https://glitch.com/
- » https://playcode.io/
- » https://jsfiddle.net/
- » https://jsbin.com/


Smooth scroll Vanilla JS | eleme
https://codepen.io/Jopzik/pen/WNeoYad
Smooth scroll Vanilla JS | eleme...
A PEN BY Juan Alexis Mora
Run
Fork
Settings
Change View
HTML (Pug)
CSS
JS
Console
Assets
Comments
⌘
Collections
Embed
Export
Share

```
-let n = 1
while n <= 30
  article
    h2= `title ${n}`
    p Aenean a mauris elit.
    Quisque accumsan ac nunc
    sed fermentum.
    Pellentesque vel ligula
    eros. Donec sapien
    tellus, volutpat vitae
    sollicitudin lobortis,
    malesuada viverra urna.
  -n++
```

```
*,
*::before,
*::after {
  padding    : 0;
  margin     : 0;
  box-sizing : border-box;
}

body {
  --spacing    : 2rem;
  display      : grid;
  grid-row-gap :
var(--spacing);
  padding      :
```

```
const articles =
[...document.querySelectorAll
('article')],
      blockState = ['start',
'center', 'end', 'nearest']
let article = articles[0]

const getRandom = n =>
Math.round(Math.random() *
(n.length - 1))

setInterval(() => {

 article.classList.remove('ac
```

title 13
Aenean a mauris elit. Quisque accumsan ac nunc sed fermentum. Pellentesque vel ligula eros. Donec sapien tellus, volutpat vitae sollicitudin lobortis, malesuada viverra urna.

title 14
Aenean a mauris elit. Quisque accumsan ac nunc sed fermentum. Pellentesque vel ligula eros. Donec sapien tellus, volutpat vitae sollicitudin lobortis, malesuada viverra urna.

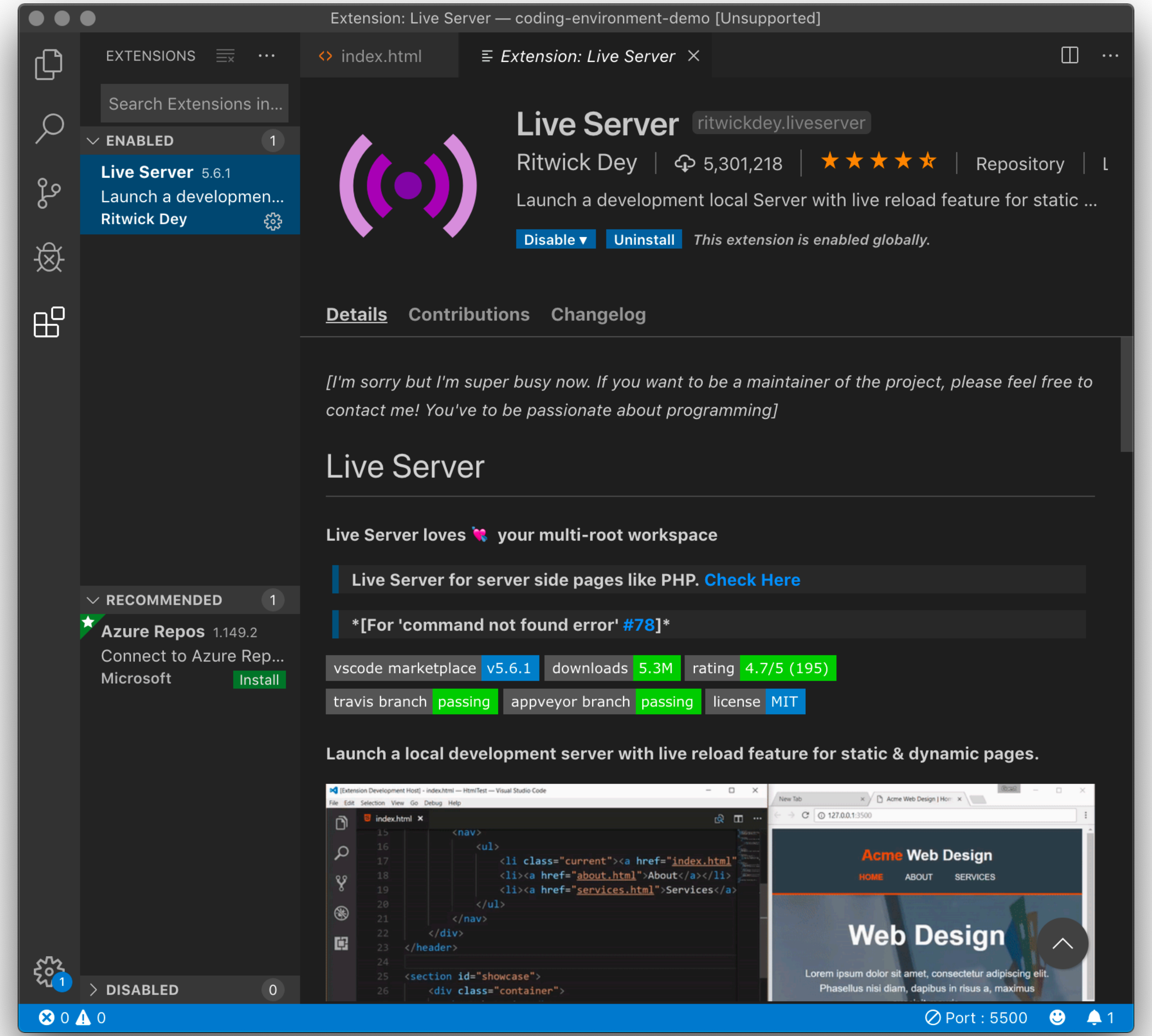
title 15
Aenean a mauris elit. Quisque accumsan ac nunc sed fermentum. Pellentesque vel ligula eros. Donec sapien tellus, volutpat vitae sollicitudin lobortis, malesuada viverra urna.

Running JS in a coding environment

If you are using VS Code install *Live Server*, start a simple HTML file, and try adding:

```
<script>alert("On Wisconsin");</script>
```

http://127.0.0.1:5500/index.html


Extension: Live Server — coding-environment-demo [Unsupported]

EXTENSIONS

Search Extensions in...

ENABLED 1

Live Server 5.6.1
Launch a developmen...
Ritwick Dey

RECOMMENDED 1

Azure Repos 1.149.2
Connect to Azure Rep...
Microsoft Install

DISABLED 0

index.html | Extension: Live Server

Live Server ritwickdey.liveserver
Ritwick Dey | 5,301,218 | ★★★★★ | Repository | L
Launch a development local Server with live reload feature for static ...
Disable ▾ Uninstall *This extension is enabled globally.*

Details Contributions Changelog

[I'm sorry but I'm super busy now. If you want to be a maintainer of the project, please feel free to contact me! You've to be passionate about programming]

Live Server

Live Server loves 💘 your multi-root workspace

Live Server for server side pages like PHP. Check Here

[For 'command not found error' #78]

vscode marketplace v5.6.1 | downloads 5.3M | rating 4.7/5 (195)
travis branch passing | appveyor branch passing | license MIT

Launch a local development server with live reload feature for static & dynamic pages.

Acme Web Design
HOME ABOUT SERVICES
Web Design
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus nisi diam, dapibus in risus a, maximus

0 0 | Port : 5500 | 1

What is this "TypeScript" I hear about?

Definition: TypeScript is a strict syntactical superset of JS developed to enable the development of large-scale applications and to add static typing.

Alternatives: CoffeeScript, LiveScript, Babel

Preprocessors compile code written in TS, CS, LS, and Babel into JS that can be executed by a JS engine.

TypeScript code:

```
var peerMentors: string[] = ['Mathias', 'Jeff'];
var firstPeerMentor: string = array[0];
```

Compiles into JS code:

```
var peerMentors = ['Mathias', 'Jeff'];
var firstPeerMentor = array[0];
```

Syntax, JS for Java Developers

Variables

Definition: Variables are *containers* that hold reusable data.

» ES6 defines seven standard data types: *numbers*, *string*, *boolean*, *null*, *undefined*, *symbol*, *object*

» JS is a dynamically, or loosely, typed language, and data type is inferred from the declaration and can be changed over time — Let's try!

» Three variable containers:

```
var userName = "Jack";
let userName = "Jill";
const interestRate = 4.25;
```

» `var` and `let` work identically but have different *scopes*

» `var` declares a variable that is globally accessible

» `let` declares a variable that is only accessible within the current block, e.g., a `for` loop

» `const` declares a variable that is unchangeable — Let's try!

» JS has a flexible and powerful declaration syntax, for example:

```
var firstName = "Andy", lastName = "Schoen", age = 28;
var firstName = "Andy",
lastName = "Schoen",
age = 28;
var fullName = firstName + " " + lastName;
```

» Because JS is dynamically typed, you can query the data type:

```
typeof firstName;
"string"
```

TopHat Question

Objects

Definition: Objects are unordered collection of related data of primitive or reference types.

- Object elements are defined using `key: value` statements.

```
var teachingAssistant = {
    firstName: "Andy",
    lastName: "Schoen",
    age: 28
}
teachingAssistant;
> {firstName: "Andy", lastName: "Schoen", age: 28}
```

Object Properties

» Different notations to access object properties

```
teachingAssistant.lastName;
> "Schoen"
teachingAssistant["lastName"];
> "Schoen"
let userFocus = "lastName";
teachingAssistant[userFocus];
> "Schoen"
```

Arrays

Definition: An array is a variable that contains multiple elements.
- Like variables, arrays are also dynamically typed.
- JS arrays can contain elements of different types.

```
var myGradStudents = ["Andy", "David", "Laura"];
myGradStudents[3] = "Nathan";
myGradStudents;
> ["Andy", "David", "Laura", "Nathan"]
myGradStudents[4] = 4;
myGradStudents;
> ["Andy", "David", "Laura", "Nathan", 4]
```

Functions[5]

Definition: A procedure that includes a set of statements that performs a task or calculates a value. The function must be defined and called within the same scope.

» Functions can be used to perform specific tasks.

```
function fahrenheitToCelcius(temperature) {
    return (temperature - 32) * 5/9;
}
fahrenheitToCelcius(77);
> 25
```

[5] Functions

» Functions can also serve as methods associated with objects.

```
var weatherReport = {
    temperature: 77,
    humidity: 64,
    wind: 6,
    celcius: function() {
    return (this.temperature - 32) * 5/9;
    }
}
weatherReport.temperature;
77
weatherReport.celcius();
25
```

Anonymous functions

Definition: Anonymous functions are declared without named identifiers that refer to them.

Form 1:

```
var firstItem = function (array) {return array[0]};
```

Form 2 (arrow functions[6]):

```
const firstItem = array => return array[0];
```

[6] Zen Dev

Anonymous vs. Declared[7]

Named	Anonymous
Debugging	Scope
Recursion	Brevity

[7] Scott Logic

Conditionals

Definition: Conditionals allow the code to make decisions and carry out different actions depending on different inputs.

Three types:

1. `if...else` statements
2. `switch` statements
3. Ternary operator

Comparison and logical operators

- `===` and `!==` (identical to/not identical *objects*)
- `==` and `!=` (identical to/not identical *values*)
- `<` and `>` (less/greater than)
- `<=` and `=>` (less/greater than or equal to)
- `&&` (AND)
- `||` (OR)

Example *object* comparison:

```
var ta1 = { name: "Andy" };
var ta2 = { name: "Hanna" };
console.log(ta1 === ta2);
> false
```

Example *value* comparison:

```
var ta1 = { name: "Andy" };
var ta2 = { name: "Andy" };
console.log(ta1.name == ta2.name);
true
```

Pro Tip: In JS, any value that is not `false`, `undefined`, `null`, `0`, `NaN`, or `""` returns `true`.

```
var currentMember = false;


if (currentMember) {
  para.textContent = 'Sign In';
} else {
  para.textContent = 'Sign Up';
}
```

We don't need to explicitly specify `=== true`.

if...else statements

```
<select id="sign">
  <option value="">--Make a choice--</option>
  <option value="wisconsin">Wisconsin</option>
  <option value="minnesota">Minnesota</option>
...

var select = document.querySelector('select');
var para = document.querySelector('p');

select.addEventListener('change', showRate);

function showRate() {
  var choice = select.value;
  if (choice === 'wisconsin') {
    para.textContent = 'Insurance rate is: ' + 4.5;
  } else if (choice === 'minnesota') {
    para.textContent = 'Insurance rate is: ' + 3.5;
...
```

```
var select = document.querySelector('select');
var para = document.querySelector('p');

select.addEventListener('change', showRate);

function showRate() {
  var choice = select.value;
  switch (choice) {
    case 'wisconsin':
          para.textContent = 'Insurance rate is: ' + 4.5;
          case 'minnesota':
          para.textContent = 'Insurance rate is: ' + 3.5;
...
```

Ternary operator

Definition: An operator that tests a condition and returns one output if `true` and another if it is `false`.

Prototype:

```
( condition ) ? doSomething : doSomethingElse;
```

Example:

```
(currentMember) ? para.textContent = 'Sign In' : para.textContent = 'Sign Up';
```

Looping

Definition: Executing one or more statements repeatedly until certain conditions are met. To express a loop, we need a counter, an exit condition, and an iterator.

A `for` loop:

```
for (initializer; exit-condition; final-expression) {
  // statement
}
```

while and do...while loops:

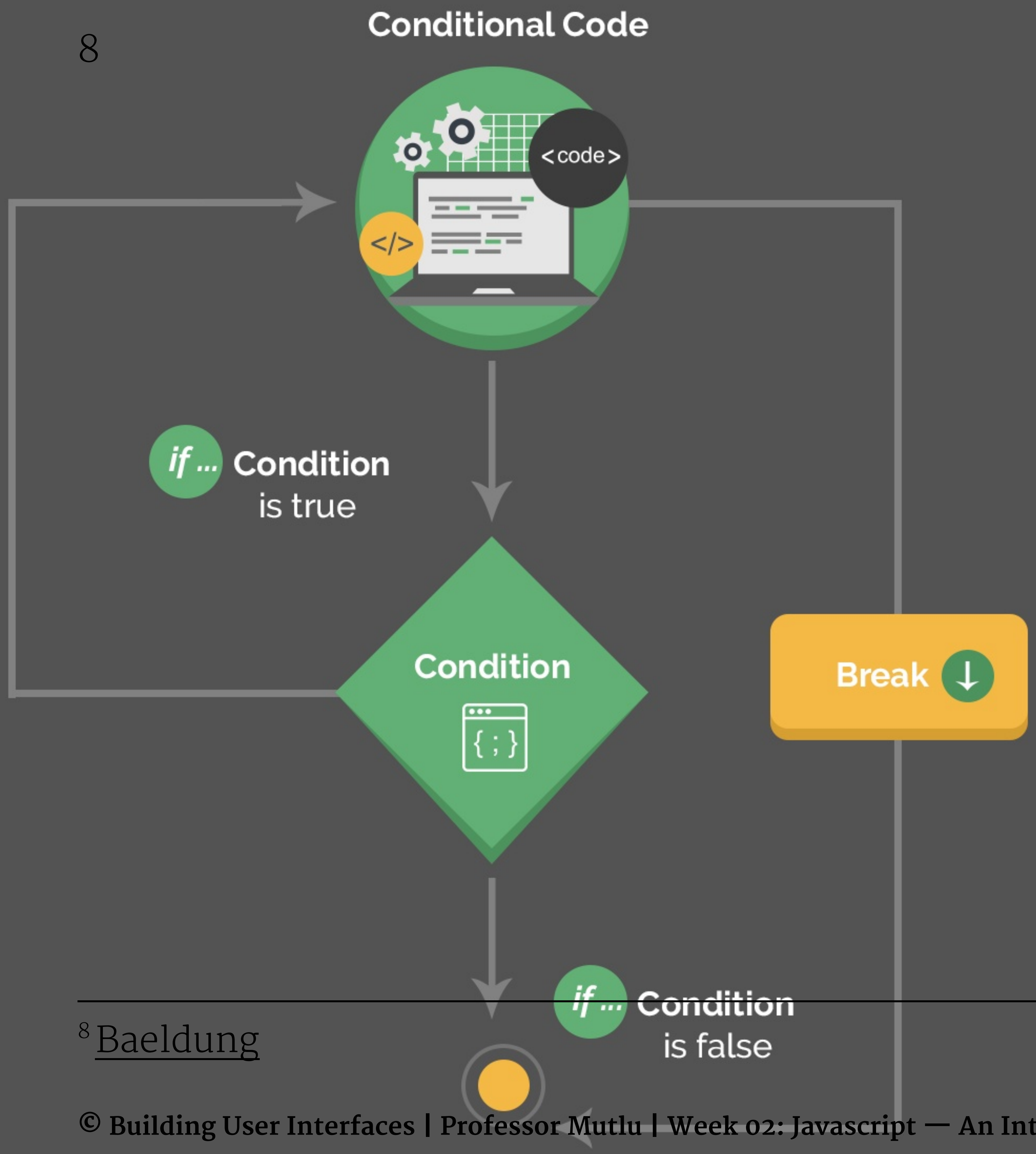
```
initializer
while (exit-condition) {
  // statement
  final-expression
}

initializer
do {
  // statement
  final-expression
} while (exit-condition)
```

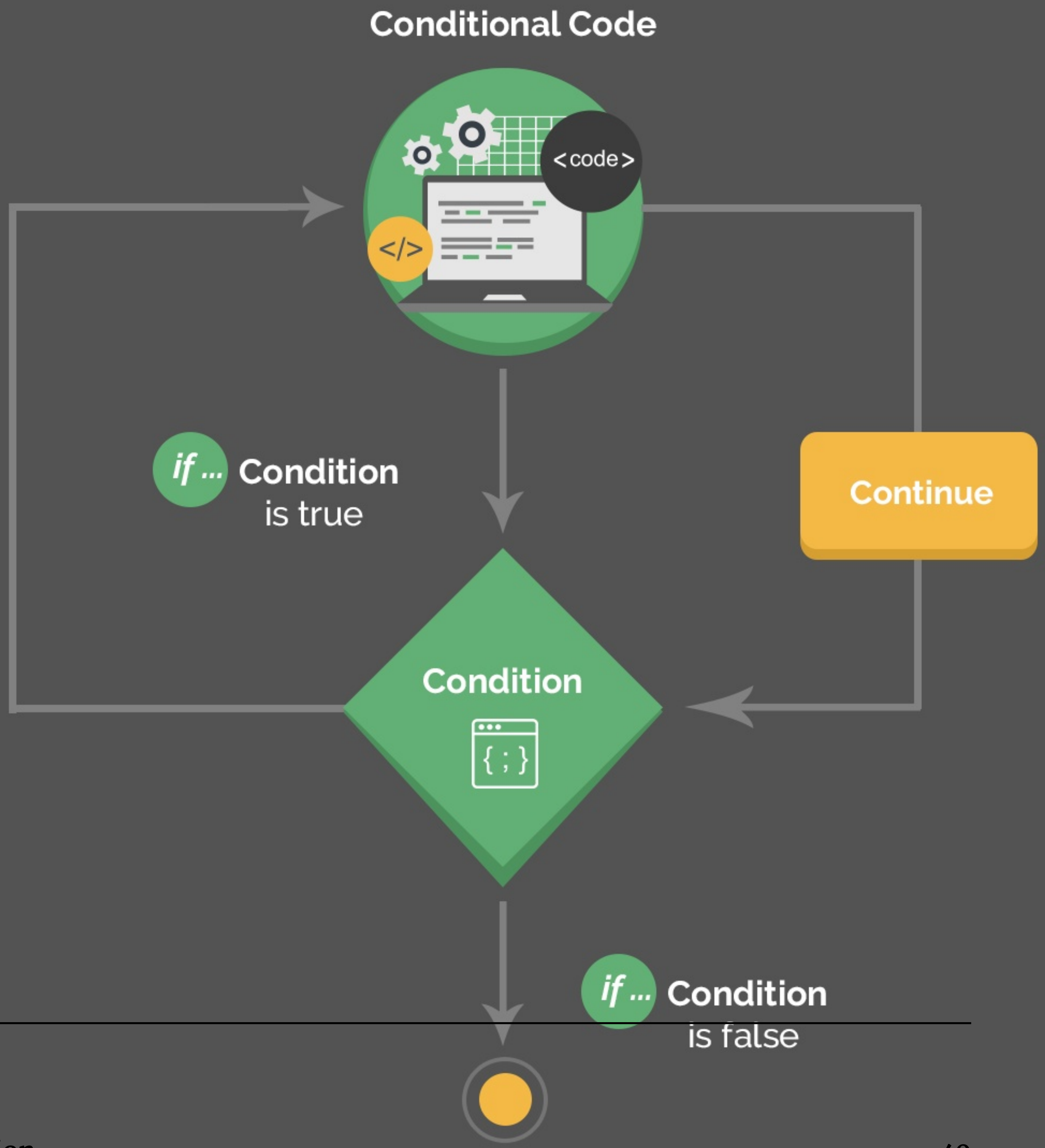
Exiting loops, skipping iterations

```
for (initializer; exit-condition; final-expression) {
  // statement
  if (special-condition-exit) { break; }
    if (special-condition-skip) { continue; }
    // statement
}
```

8



Conditional Code
<code>
</>
if … Condition is true
Condition
{ ; }
Break
if … Condition is false



Conditional Code
<code>
</>
if … Condition is true
Continue
Condition
{ ; }
if … Condition is false

[8] Baeldung

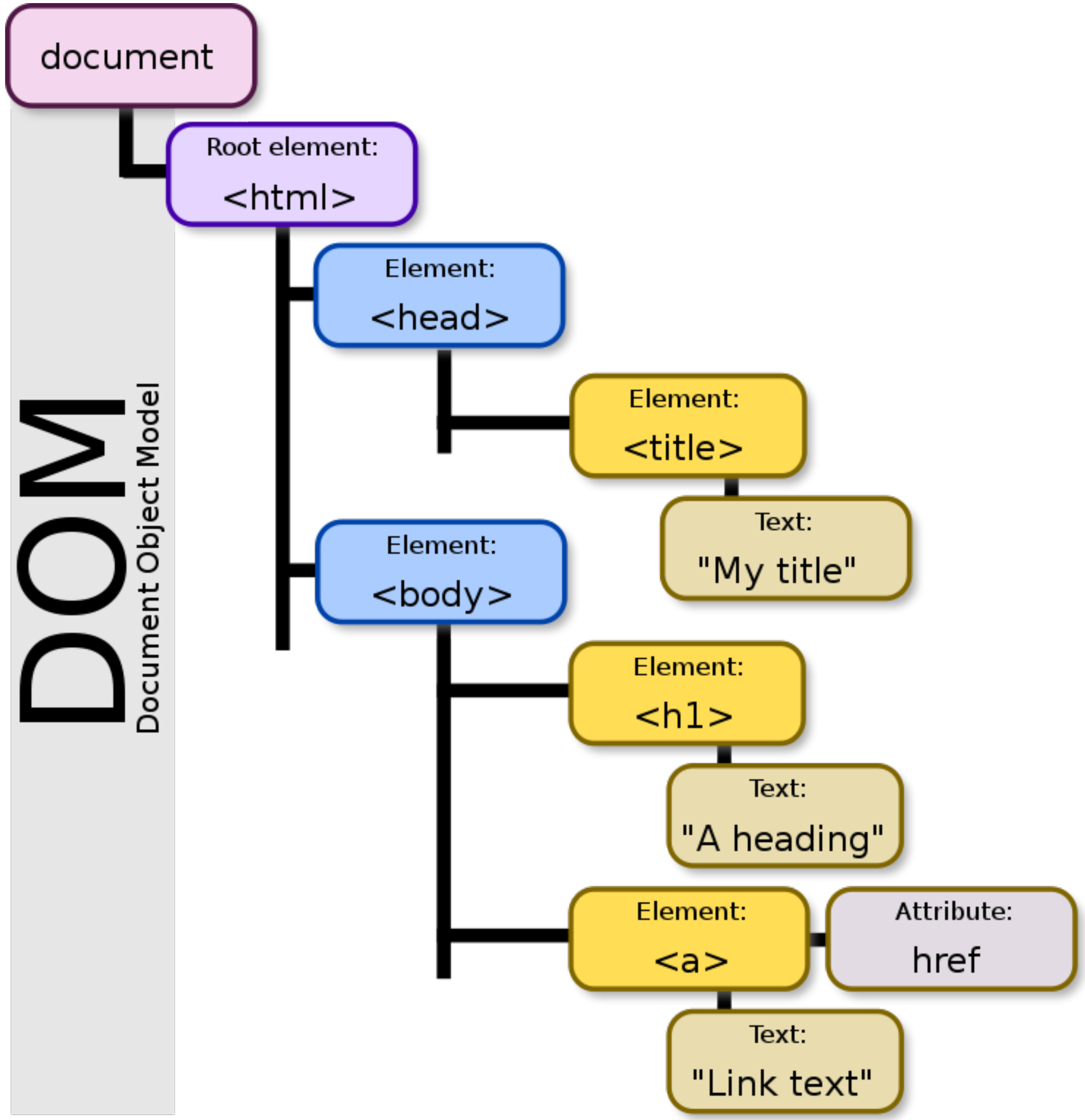
Interacting with User-facing Elements

Document Object Model

Definition: Document Object Model (DOM) translates an HTML or XML document into a tree structure where each node represents an object on the page.

This is great news for us, because JS can interact with this structure.

Source[9]



[9] Wikipedia: DOM

DOM Programming Interface

- **Objects:** HTML elements, such as a paragraph of text.
- **Property:** Value that we can get or set, such as the `id` of an element.
- **Method:** An action we can take, such as adding or deleting an HTML element.

For JS to interact with user-facing elements, we first need to access them…

Accessing HTML elements

Most common way of accessing content is `getElementById()`.

```
<p id="userName"></p>


<script>
    document.getElementById("userName").innerHTML = "Andy Schoen";
</script>
```

We can also find elements using tag name, class name, CSS selectors, and HTML object collections.

Manipulating HTML elements

Changing content:

```
document.getElementById("userName").innerHTML = "aschoen";
```

Changing attributes:

```
document.getElementById("userImage").src = "Headshot.png";
document.getElementById("userName").style.color = "red";
```

DOM Events

Now things are heating up! 🔥

DOM provides access to HTML events, such as `onclick`, `onload`, `onunload`, `onchange`, `onmouseover`, `onmouseout`, `onmousedown`, `onmouseup`, `formaction`.

Three ways of registering functions to events:
1. Inline event handlers
2. DOM on-event handlers
3. Using event listeners

Inline Event Handlers

Example:

```
<p id="currentTemp">77</p>
<button id="convertButton" onclick="convertTemp();">Convert to Celcius</button>
<script>
    function convertTemp() {
        document.getElementById("currentTemp").innerHTML
        = (document.getElementById("currentTemp").innerHTML - 32) * 5/9;
    }
</script>
```

DOM on-event Handlers

Prototype:

```
<script>
    document.getElementById("button").onclick = doSomething();
</script>
```

Example:

```
<p id="currentTemp">77</p>
<button id="convertButton">Convert to Celcius</button>
<script>
    document.getElementById("convertButton").onclick = convertTemp;
    function convertTemp() {
        document.getElementById("currentTemp").innerHTML = (document.getElementById("currentTemp").innerHTML - 32) * 5/9;
    }
</script>
```

Using Event Listeners

Prototype:

```
document.getElementById("button").addEventListener("click", function(){ doSomething() });
```

Example:

```
<p id="currentTemp">77</p>
<button id="convertButton">Convert to Celcius</button>
<script>
    document.getElementById("convertButton").addEventListener("click", function(){ convertTemp() });

    function convertTemp() {
        document.getElementById("currentTemp").innerHTML
        = (document.getElementById("currentTemp").innerHTML - 32) * 5/9;
    }
</script>
```

Pro Tip: When we add event listeners, we are assigning a function to a handler for the handler to execute the function when needed, not calling the function right there.

Do not:

```
document.getElementById("button").addEventListener("click", doSomething() );
```

Do

```
document.getElementById("button").addEventListener("click", function(){ doSomething() });
```

Pro Tip: *Listeners* are the most efficient way to manage events.[10][11]

```
<button>A</button>
<button>B</button>
<button>C</button>
<script>
  document.body.addEventListener("click", event => {
    if (event.target.nodeName == "BUTTON") {
      console.log("Clicked", event.target.textContent);
    }
  });
</script>
```

[10] Eloquent JavaScript

[11] See in CodePen

What did we learn today?

- History and overview of web programming
- Syntax, JS for Java developers
- Interacting with user-facing elements