CS 571 Building User Interfaces

JavaScript
An Introduction

Prof. Yuhang Zhao
Computer Sciences, UW-Madison

adapted from Prof. Bilge Mutlu's slides

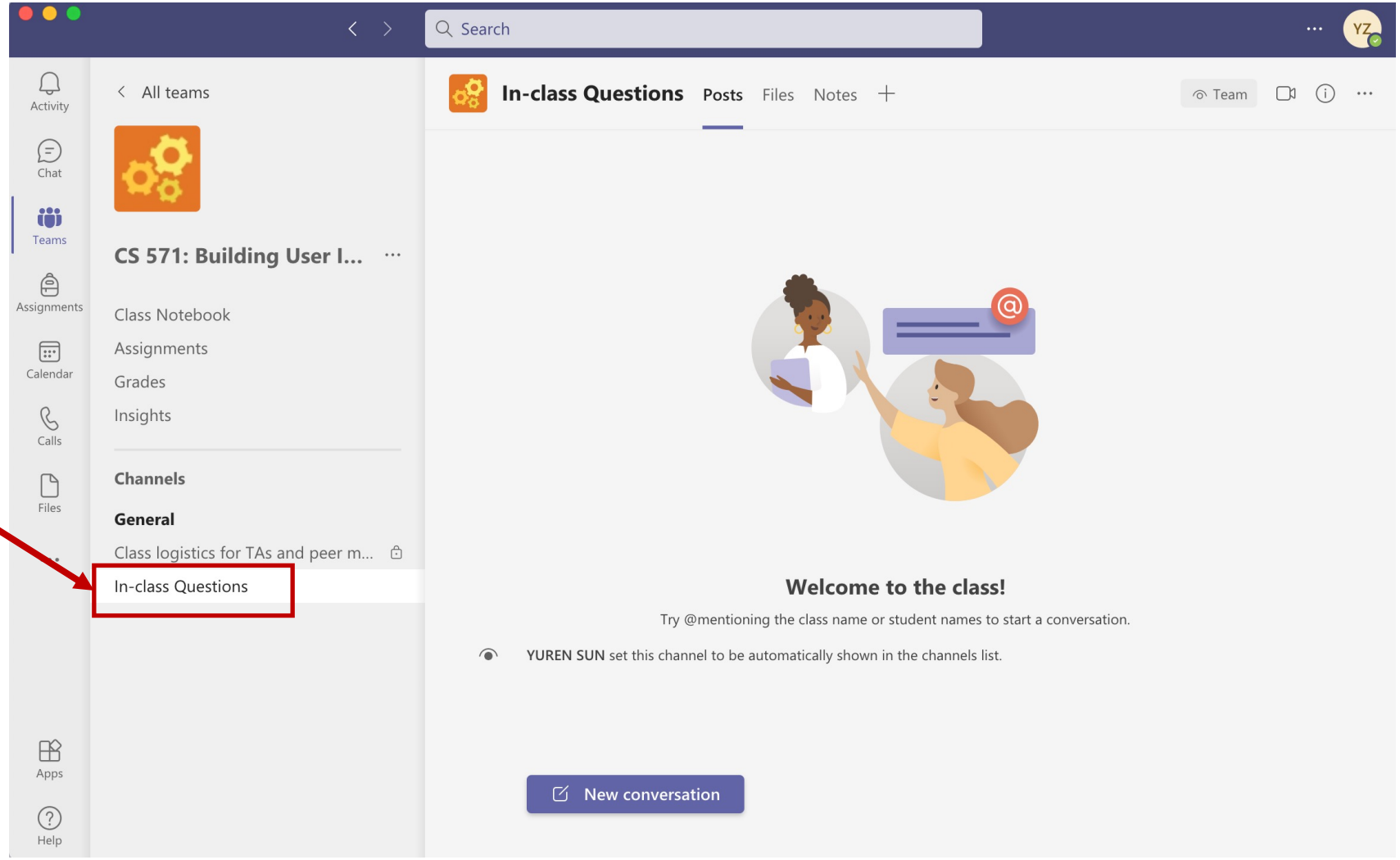
Disclaimer

- This is not a comprehensive introduction to JS, so below are links to great additional resources:
  - MDN Web Docs
  - DevDocs
  - W3 Schools
  - FreeCodeCamp

What will we learn today?

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

Live Q&A Reminder



What will you need?

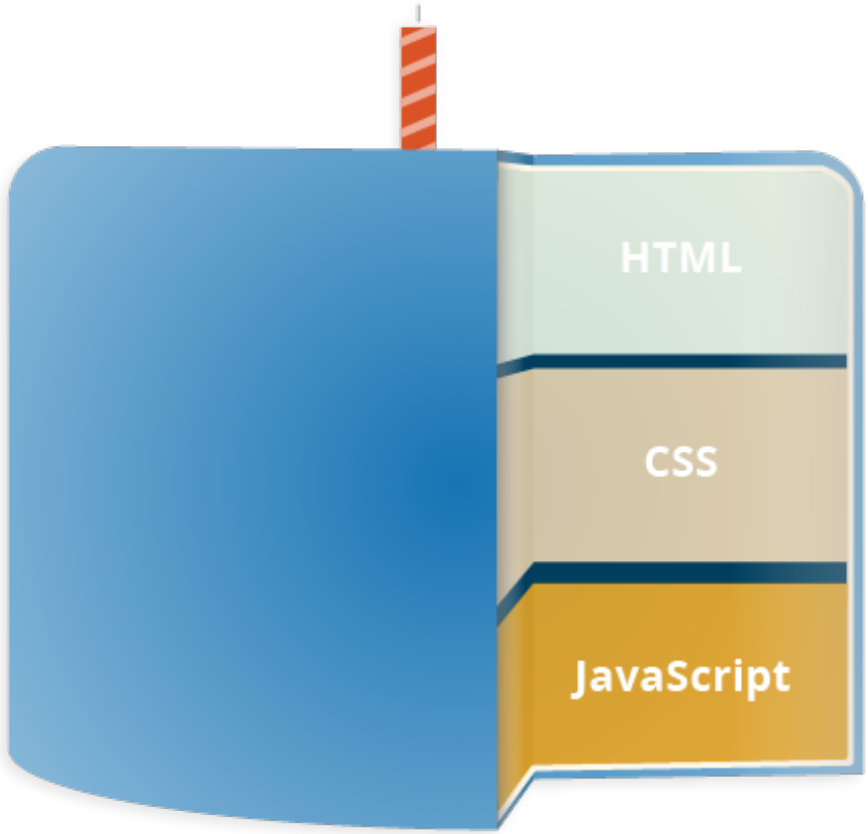
- A modern web browser (developer tools enabled)
- A source-code editor (e.g., Visual Studio Code, Atom, Sublime Text)

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 Andreesen*
- Mocha > LiveScript > JavaScript > 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:
  - HTML: Base cake layer
  - CSS: Icing
  - JS: Clown hidden in the cake



Source: The three layers of designing for the web

Let’s see an example

Consider the following very simple HTML page

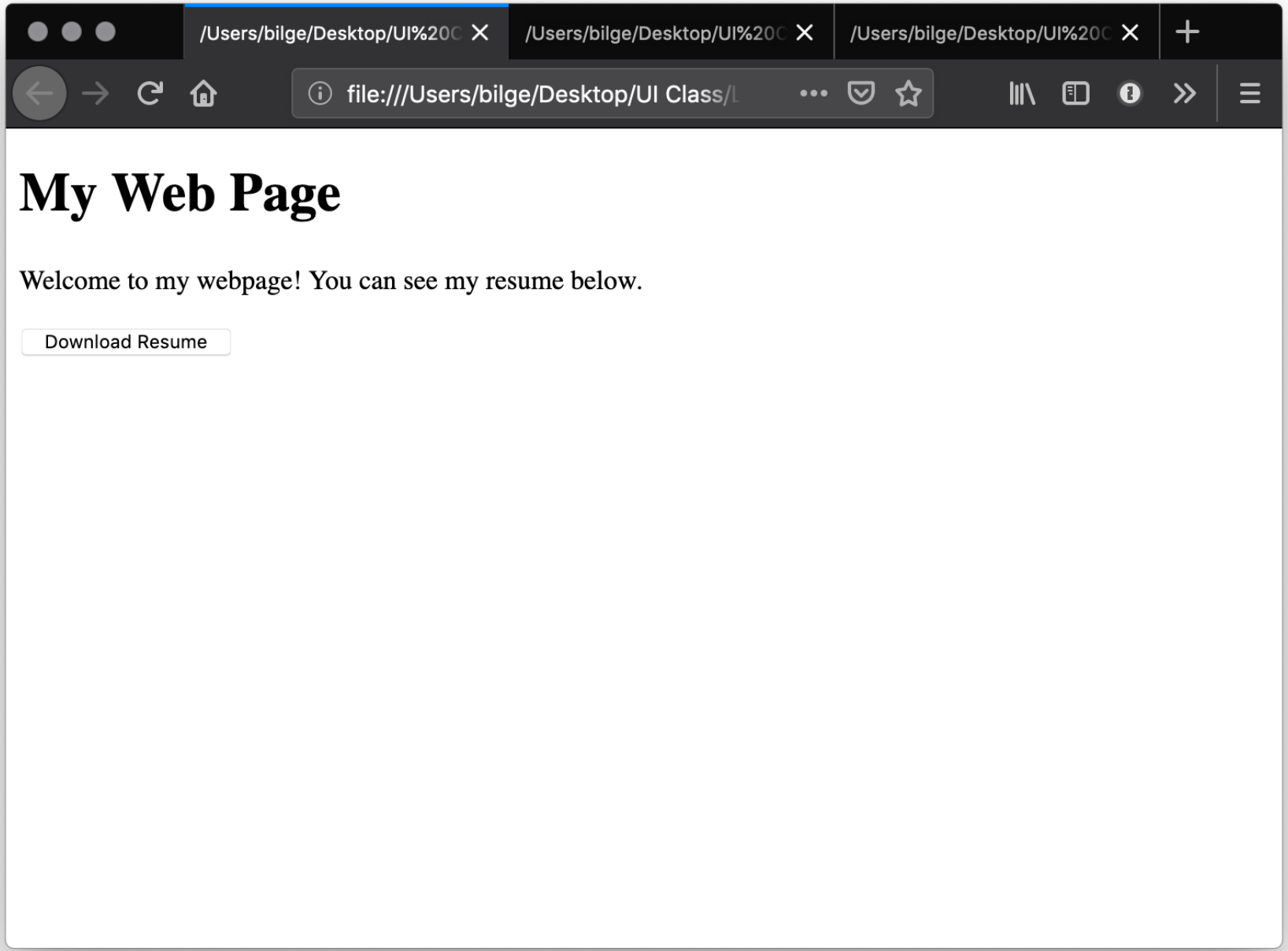
```
<!DOCTYPE html>
<html>
<head>
</head>
<body>

<h1>My Web Page</h1>

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

<button>Download Resume</button>

</body>
</html>
```

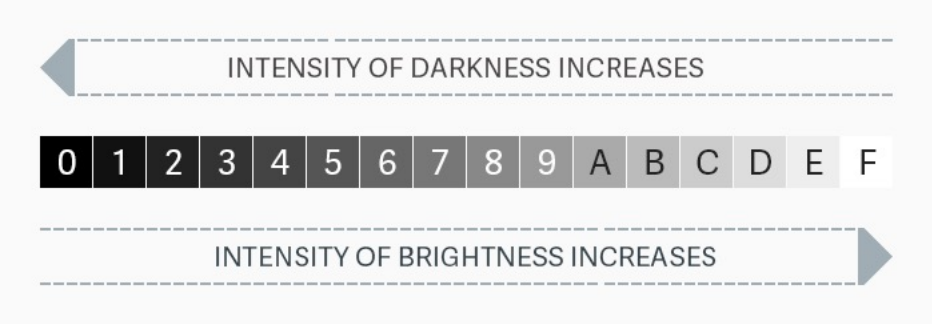

/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
file:///Users/bilge/Desktop/UI Class/
My Web Page
Welcome to my webpage! You can see my resume below.
Download Resume

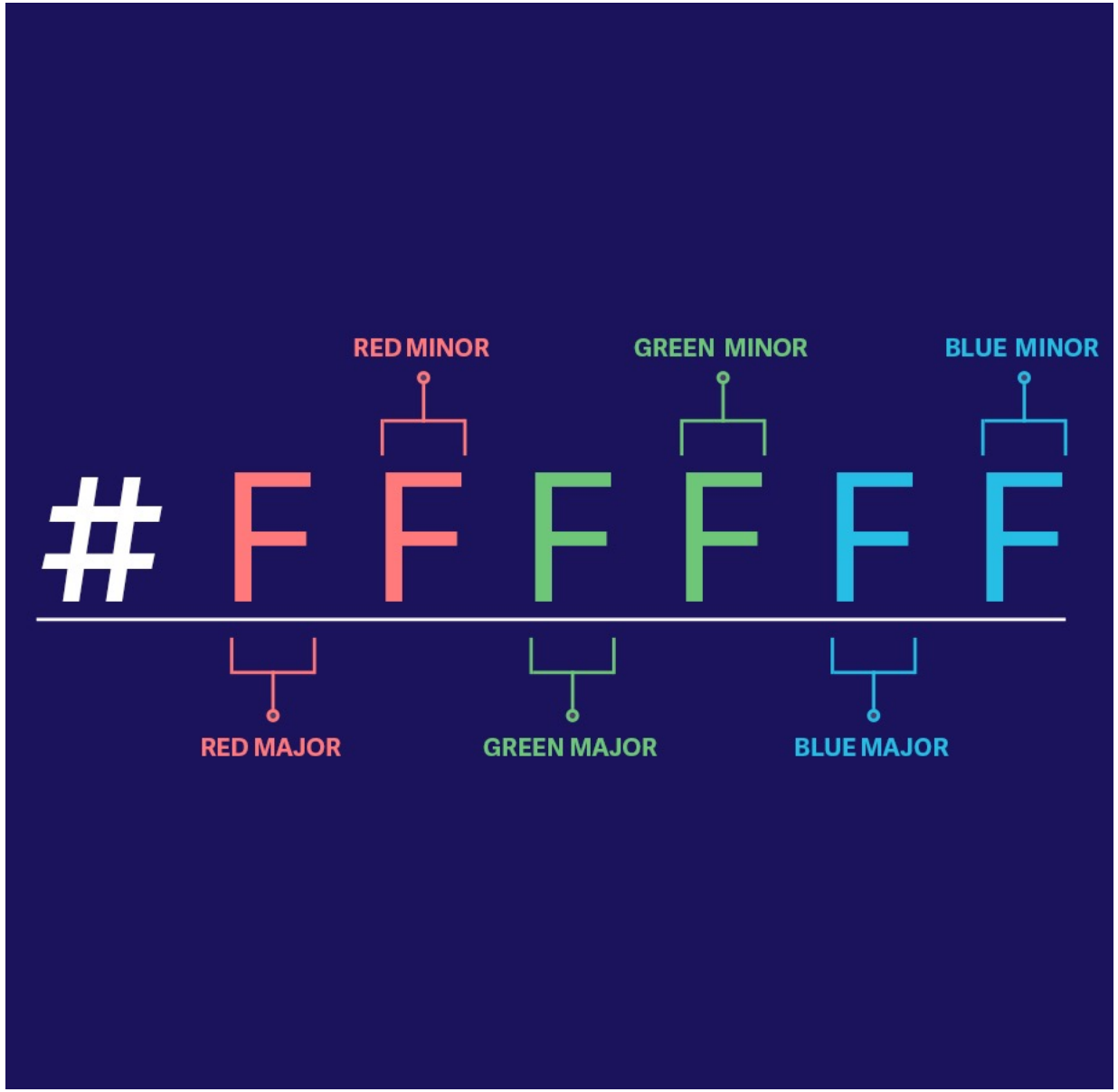
Let's improve its appearance. Within head then style:

```
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}
```

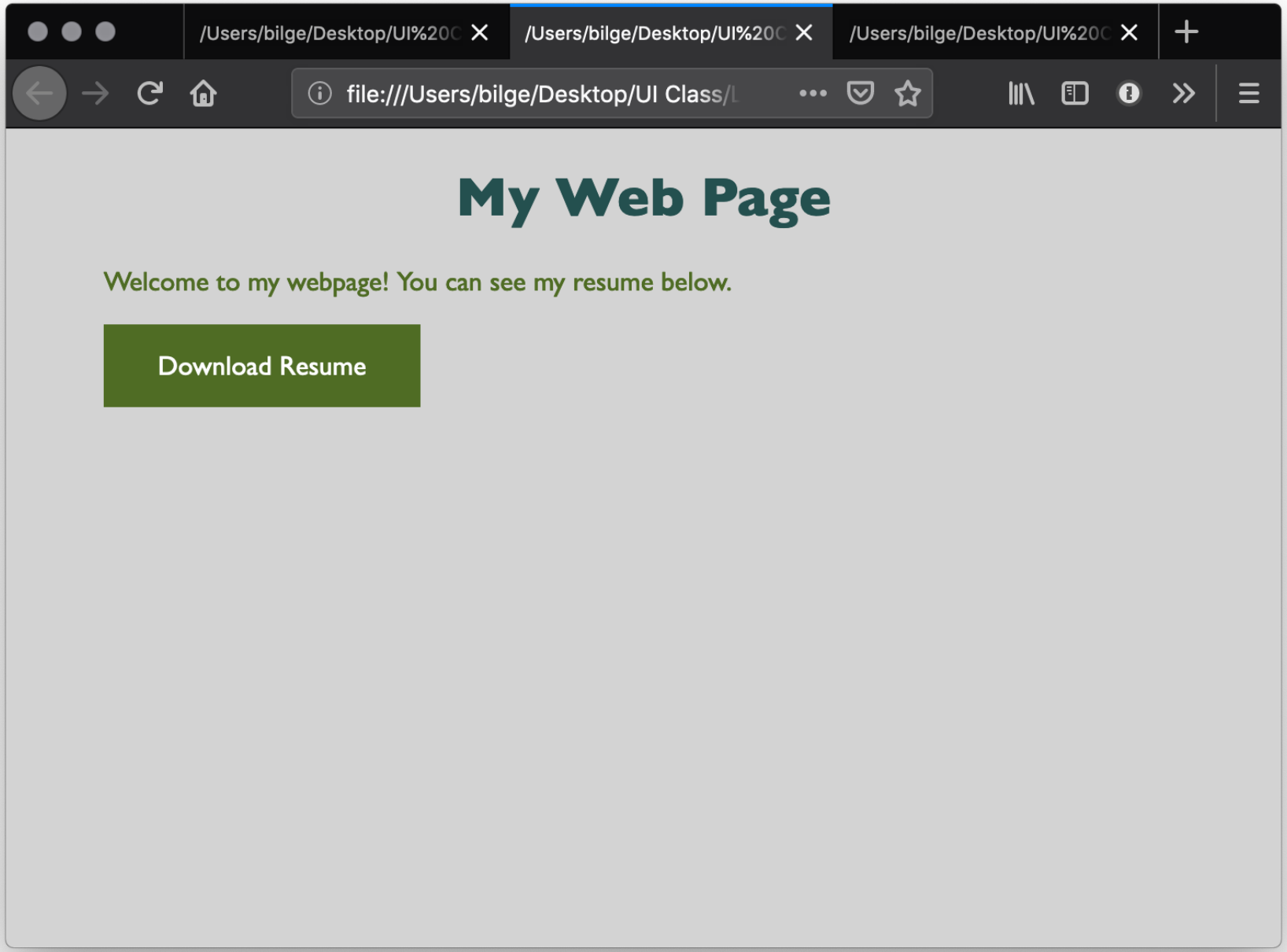
Detour: Specifying Color

- HEX triplet, RGB triplet
- Majors -> tone; minors -> shade
- Values 0-9-A-F (16 values)
- Search for “hex color”





Source: [Nitish Khagwal]

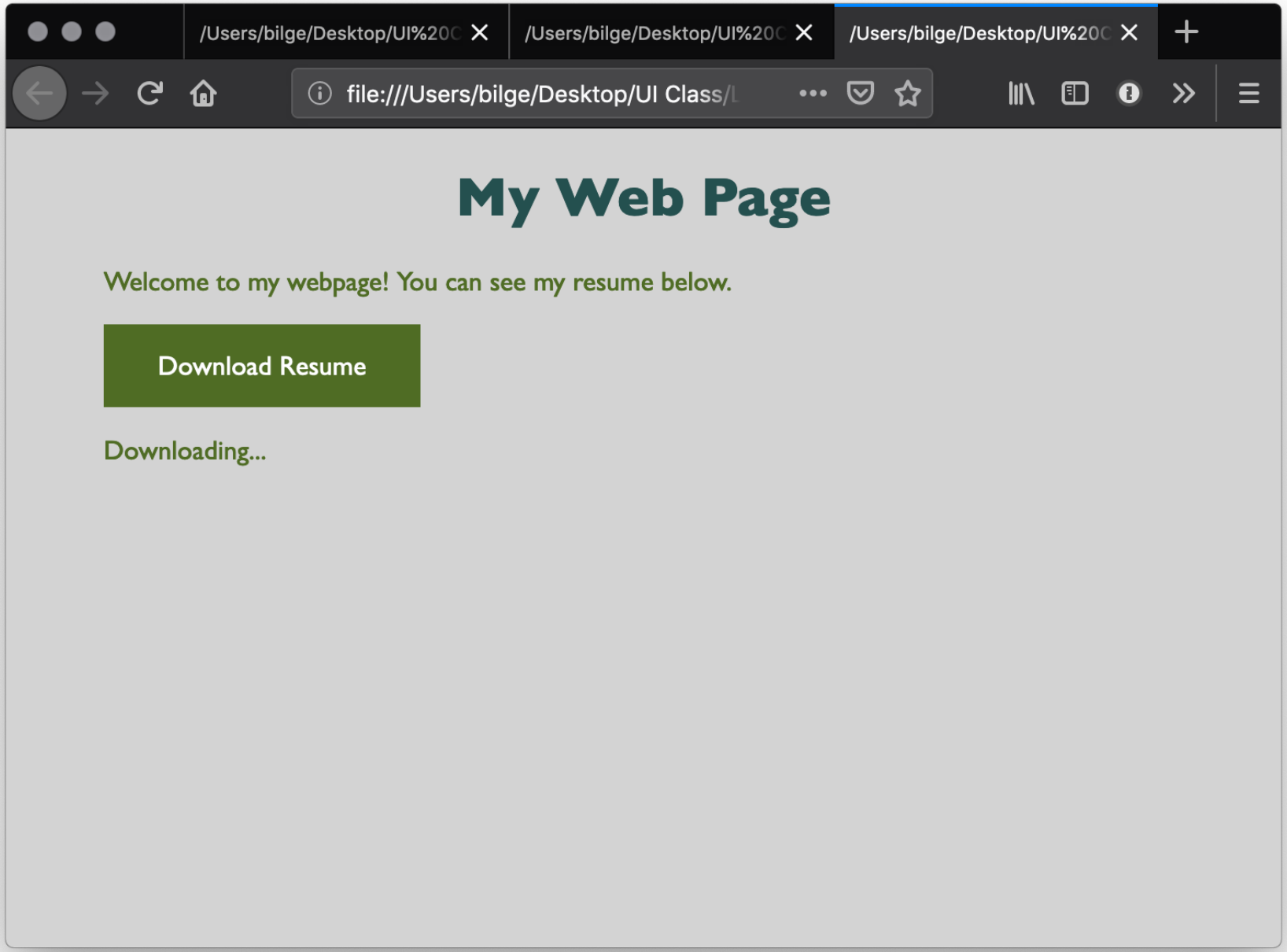

/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
/Users/bilge/Desktop/UI%20C
file:///Users/bilge/Desktop/UI Class/
My Web Page
Welcome to my webpage! You can see my resume below.
Download Resume

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/
My Web Page
Welcome to my webpage! You can see my resume below.
Download Resume
Downloading...

How does JS interact with the page?

- Internal JS
- External JS
- Inline JS handler

Internal JS

- Internal JS is included within the HTML inside <script> tags.

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

External JS

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

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

Here, defer indicates that script.js should be executed *after* the page is parsed.

Inline JS handlers

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

Pro Tips 1: In general, inline JS handlers results in inefficient and unorganized code.

Pro Tips 2: Different loading strategies are used for internal JS (listening for DOMContentLoaded event; including script after the page content) and external JS (defer and async attributes).

How is JS interpreted?

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

Source: List of ECMAScript engines; Node.js

How do I start JS development?

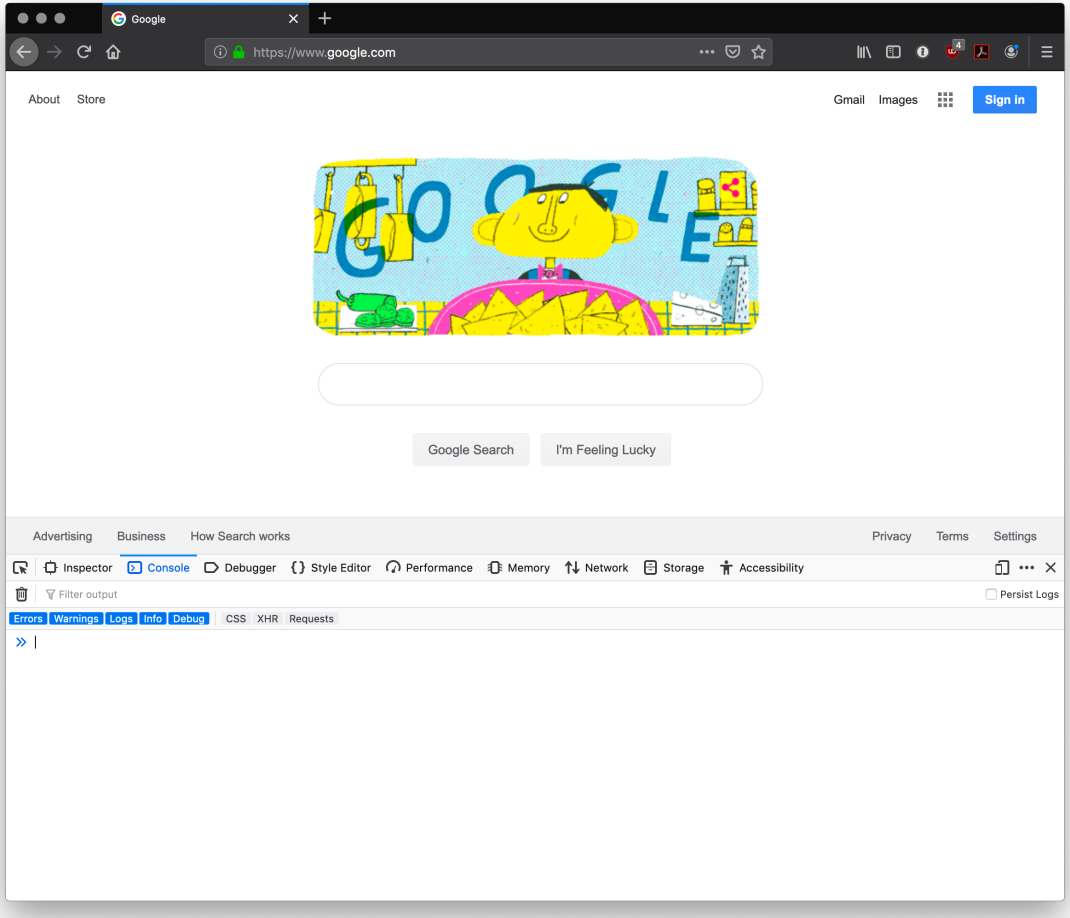
- In the **browser** – best for testing ideas, code, etc.
- In a **coding environment** – best for application development

Running JS in the browser

Ctrl-Shift-J **or** Command-Option-J

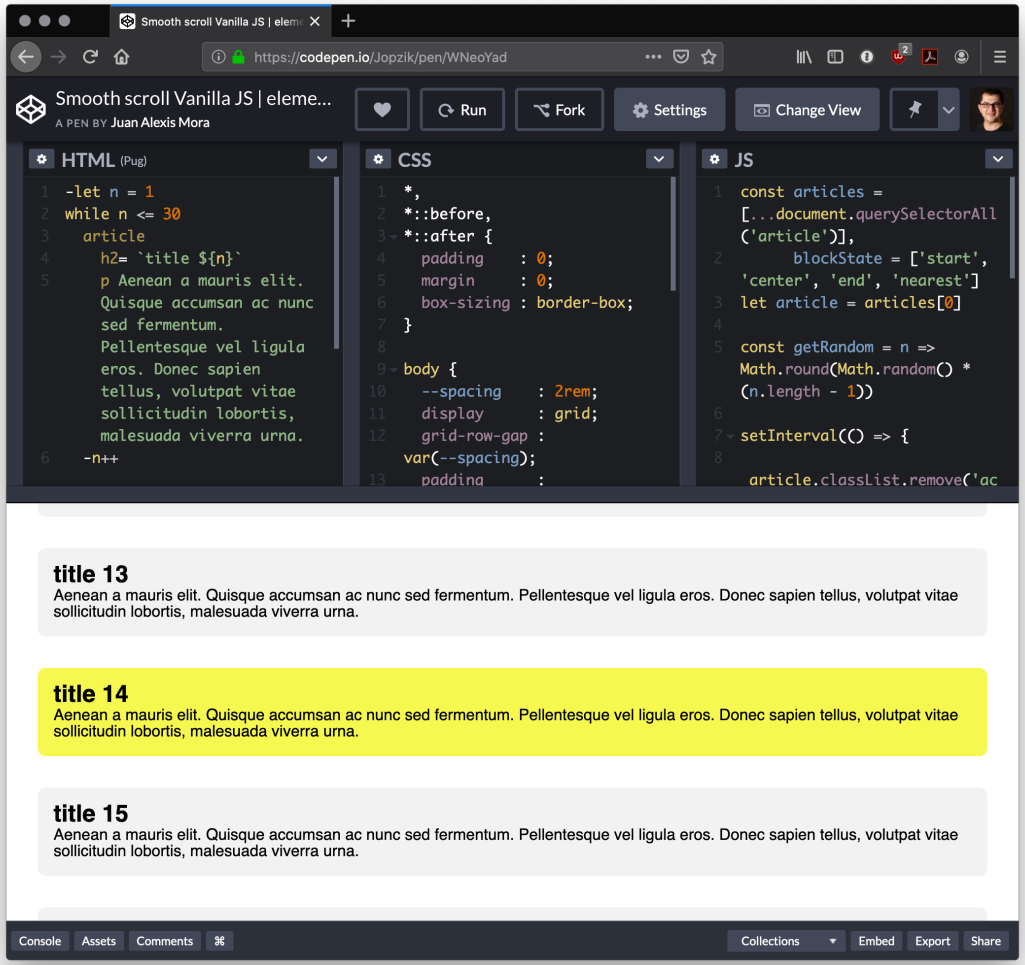
Try out:

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/

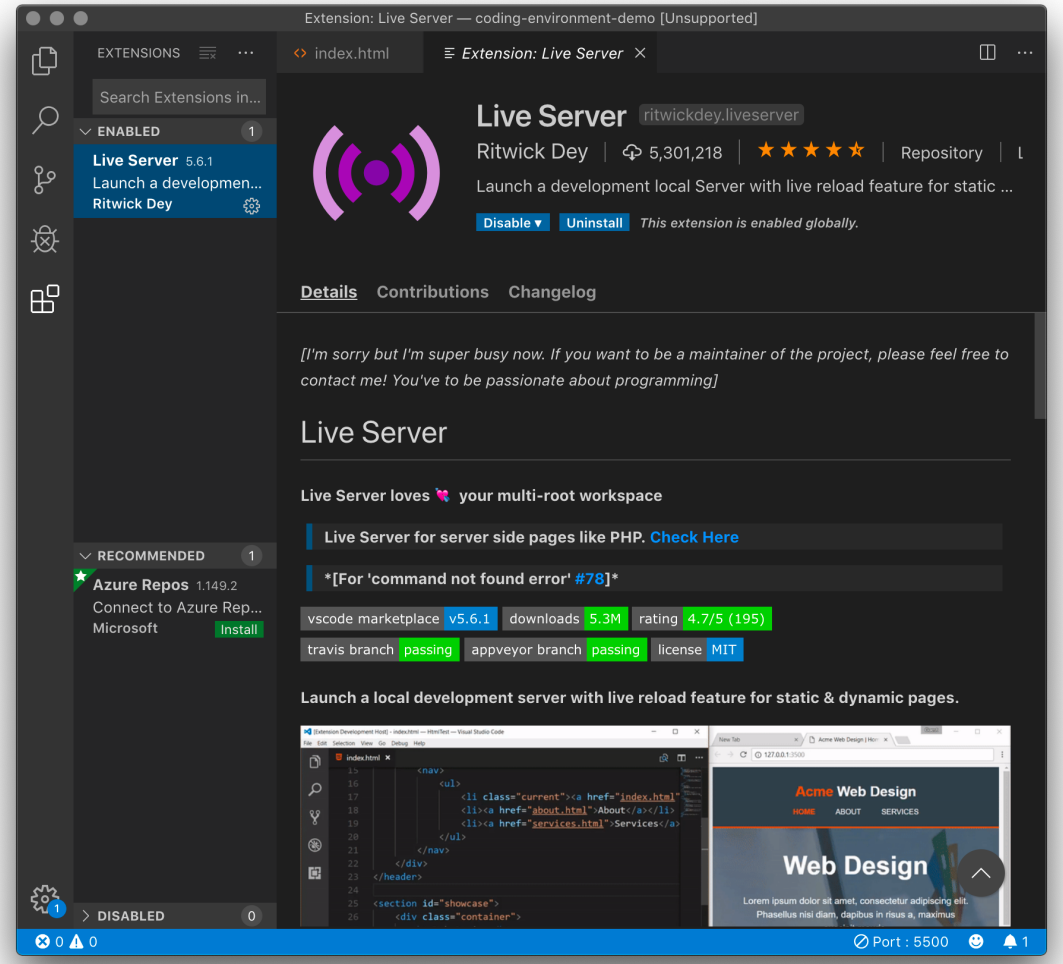


Running JS in a coding environment

- 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



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* (ensuring type safety).
- **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[] = [‘Yuren’, ‘Ilkyu’]
var firstPeerMentor: string = peerMentors[0];
```

Compiles into JS code:

```
var peerMentors = [‘Yuren’, ‘Ilkyu’];
var firstPeerMentor = peerMentors[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

Consider the following 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

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

```
var firstName = “Cole”, lastName = “Nelson”, age = 26;
var firstName = ”Cole”,
lastName = “Nelson”,
age = 26;
var fullName = firstName + “ “ + lastName;
```

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

```
typeof firstName;
“string”
```

Objects

- Definition: Objects are unordered collections of related data of primitive or reference types – defined using key: value statements

```
var teachingAssistant = {
        firstName: “John”,
        lastName: “Balis”,
        age: 24
}
teachingAssistant;
> {firstName: “John”, lastName: “Balis”, age: 24}
```

Object Properties

- Different notations to access object properties:

```
teachingAssistant.lastName;
```

> “Balis”

```
teachingAssistant[”lastName”];
```

> “Balis”

```
let userFocus = “lastName”;
teachingAssistant[userFocus];
```

>”Balis”

Arrays

- Definition: An array is a variable that contains multiple elements.
  - Like variables, arrays are also dynamically types.
  - 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

- **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
```

Source: Functions

Functions can also serve as methods associated with objects.

```
var lastestWeatherReport = {
        temperature: 77,
        humidity: 64,
        wind: 6,
        celcius: function() {
                return (this.temperature – 32) * 5/9;
        }
}
latestWeatherReport.temperature;
> 77
latestWeatherReport.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):

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

Source: Zen Dev

Declared vs. Anonymous

- Advantages of *declared* and *anonymous* functions are:

Named	Anonymous
Debugging	Scope
Recursion	Brevity

Source: 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 *strict equality* comparison:

```
var ta1 = 1;
var ta2 = “1”
console.log(ta1 === ta2);
>false
```

Example *abstract equality* comparison:

```
var ta1 = 1;
var ta2 = “1”;
console.log(ta1 == ta2);
>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’;
}
```

>Sign up

We don’t need to explicitly specify === true.

if … else **statements**

```
<select id=“sign”>
   <option value=“ “>--Make a choice--</option>
   <option value=”Illinois”>Illinois</option>
   <option value=“Indiana”>Indiana</option>
…

var select = document.querySelector(‘select’);
var para = document.querySelector(‘p’);

select.addEventListener(‘change’, setSign);

function setSign() {
   var choice = select.value;
   var messageText = ‘Current mortgage loan rate is ’;
// Data from https://www.astrology.com/horoscope/daily.html
   if (choice === ‘illinois’) {
      para.textContent = messageText + 4.50 + ‘%’;
   } else if (choice === ‘Indiana’) {
      para.textContent = messageText + 3.50 + ‘%’;
…
```

See in CodePen

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

select.addEventListener('change', setSign);

function setSign() {
    var choice = select.value;
    var messageText = 'Current mortgage loan rate is ';
    if (choice === 'illinois') {
        para.textContent = messageText + 4.50 + '%';
    } else if (choice === 'Indiana') {
        para.textContent = messageText + 3.50 + '%';
...
```

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
}
```

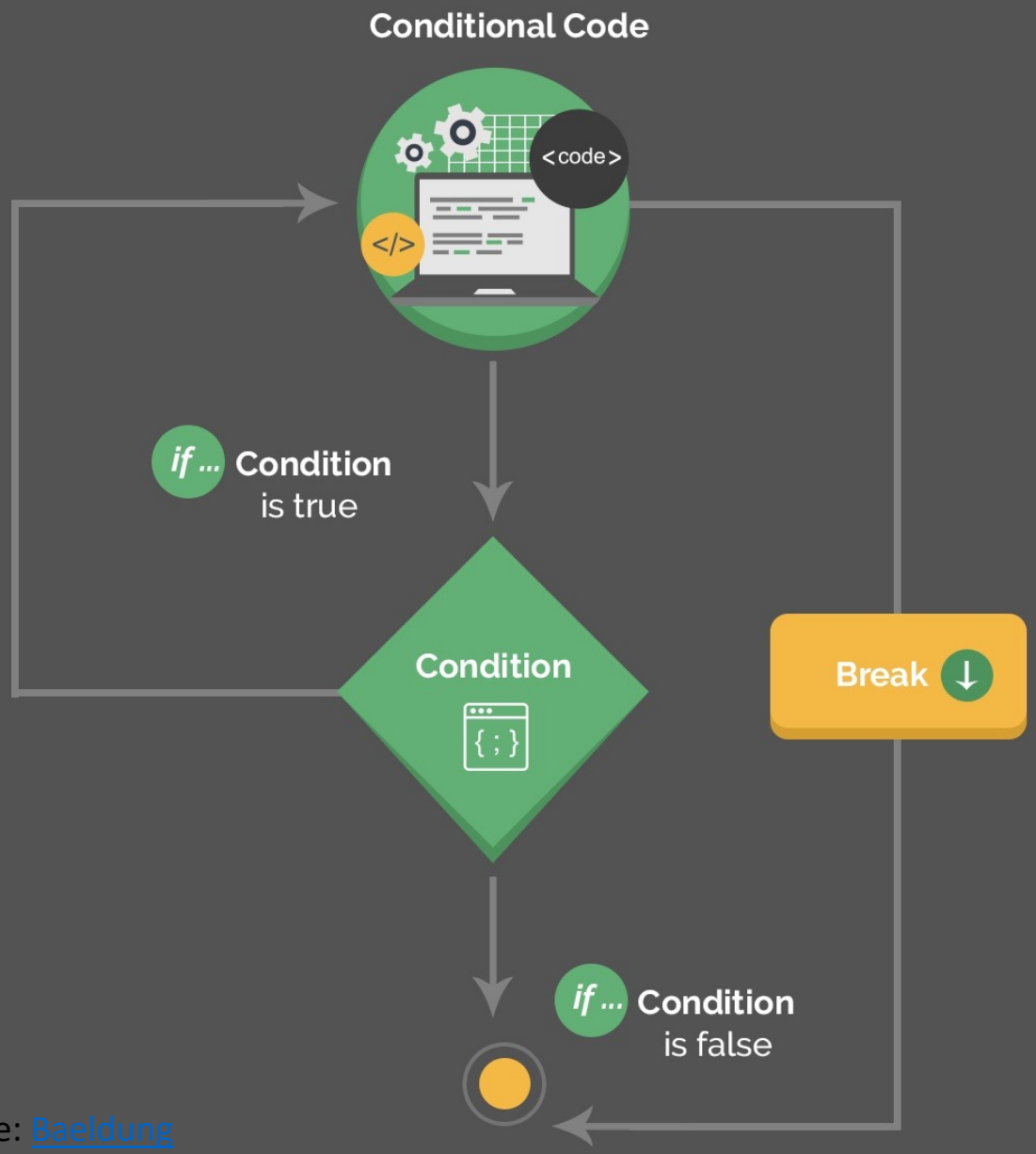
While and do … while loops:

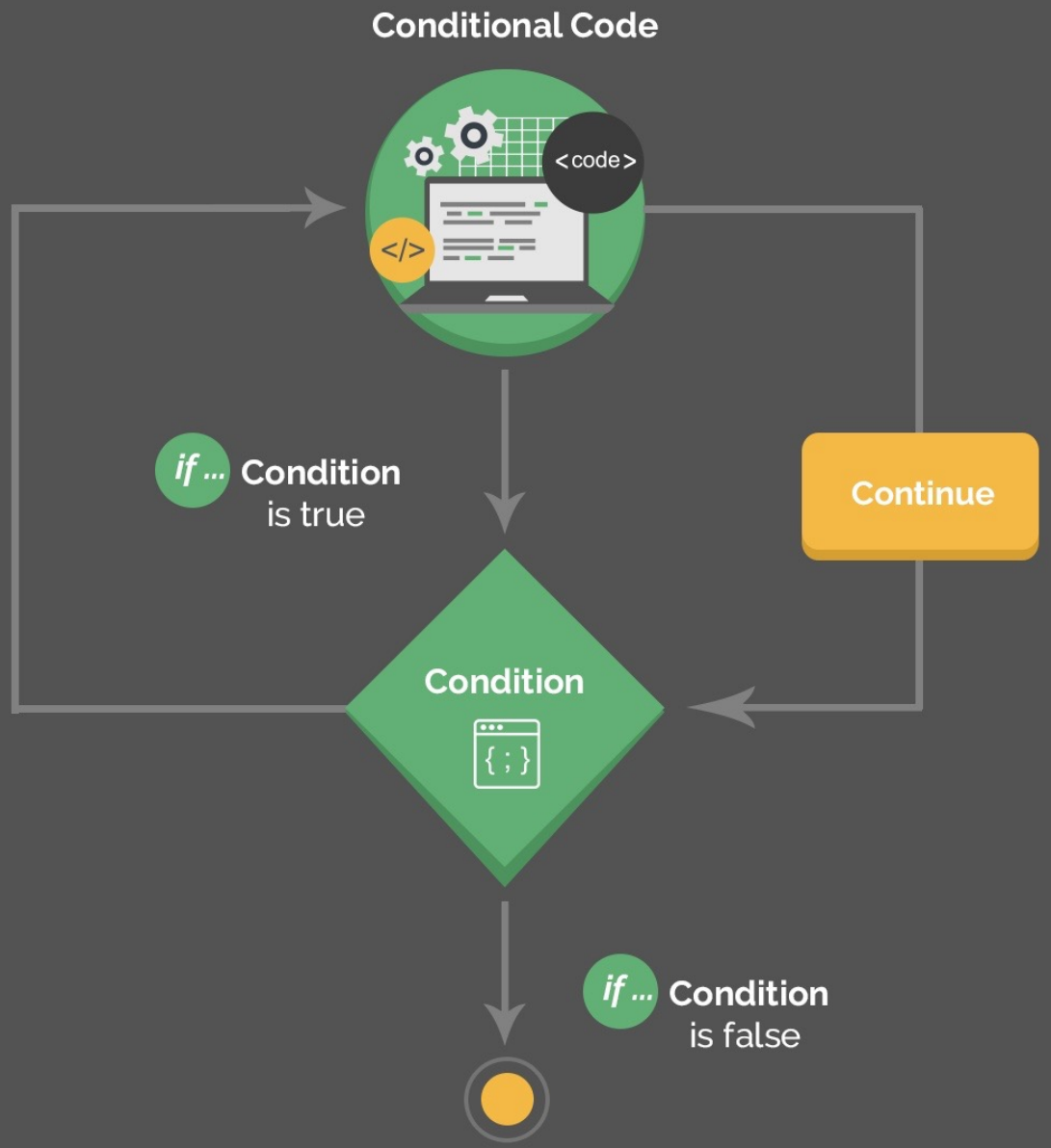
```
initializer
While (exit-condition) {
        //statement
        final-expression
}

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
}
```



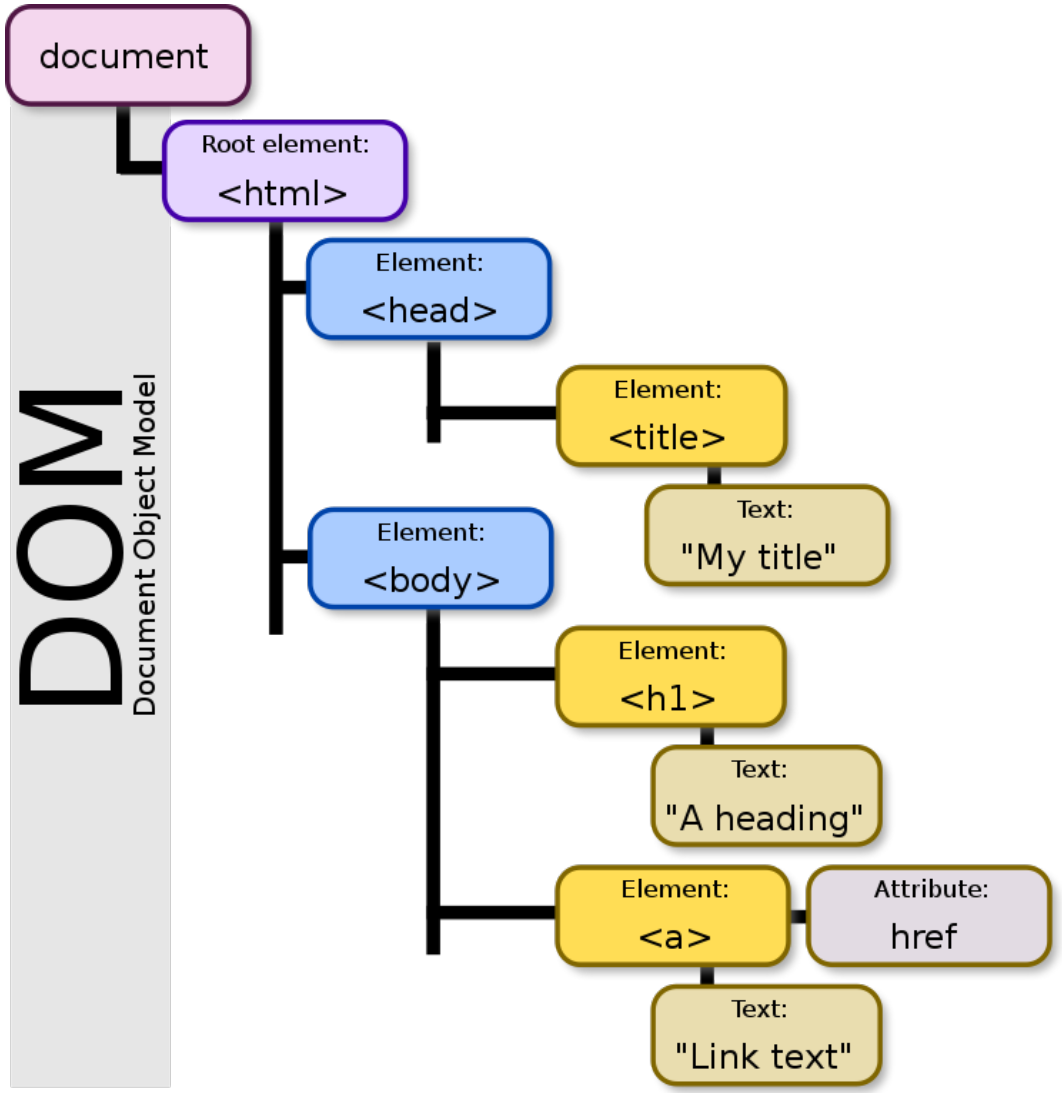


Source: Baeldung

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.



Wikipedia: DOM

DOM Programming Interface

- Objects: HTML elements, such as a paragraph of text.
- Property: Value that 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 = "Cole Nelson";
</script>
```

We can also find elements using tag name, class name, CSS selectors.

Manipulating HTML elements

- Changing content:

```
document.getElementById(“userName”).innerHTML = “Cole Nelson”;
```

- 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: onclick, onload, onunload, onchange, onmouseover, onmouseout, onmousedown, onmouseup, formaction.
- Three ways of registering functions to events:
  - Inline event handlers
  - DOM on-event handlers
  - Event listeners

Inline Event Handlers

- Prototype

```
<button id=”id-name” onclick=“function()”>Button name</button>
```

- Example:

```
<p id=“currentTemp”>7</p>
<button id=”convertButton” onclick=“function()”>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”>7</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">7</p>
<button id="convertButton">Convert to Celcius</button>
<script>
        document.getElementById("convertButton").addEventListener("click", 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();});
```

or

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

Pro Tip: Listeners are the most efficient way to manage events

```
<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>
```

See in CodePen
Eloquent JavaScript

What did we learn today?

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