front-end

DOM & Events

lab 2/8

today

I. Progressive Enhancement ideas

II. Recap (es6, arrows, methods)

III. Functions and Scope

IV. Document Object Model (DOM)

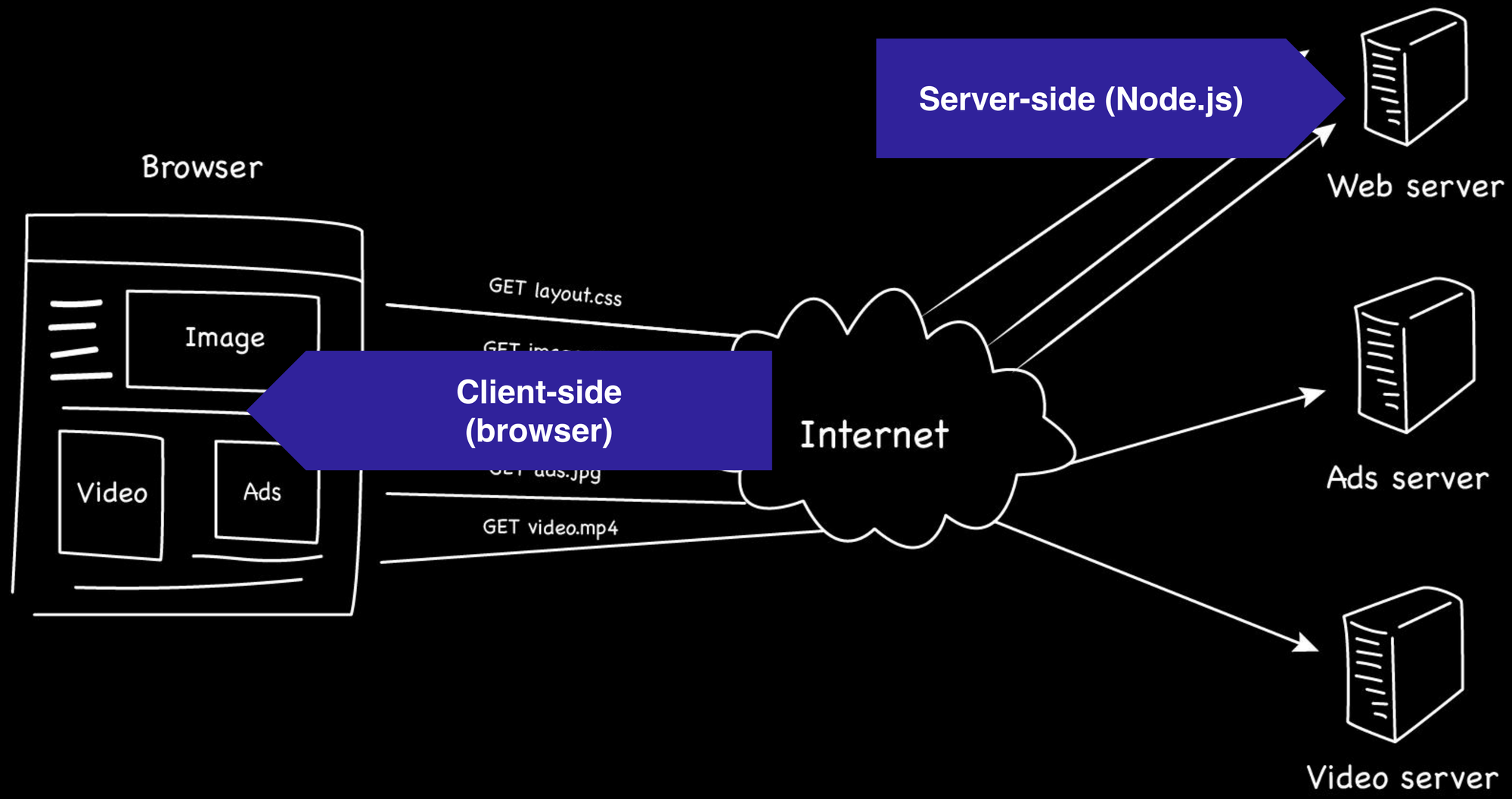
V. Web API’s

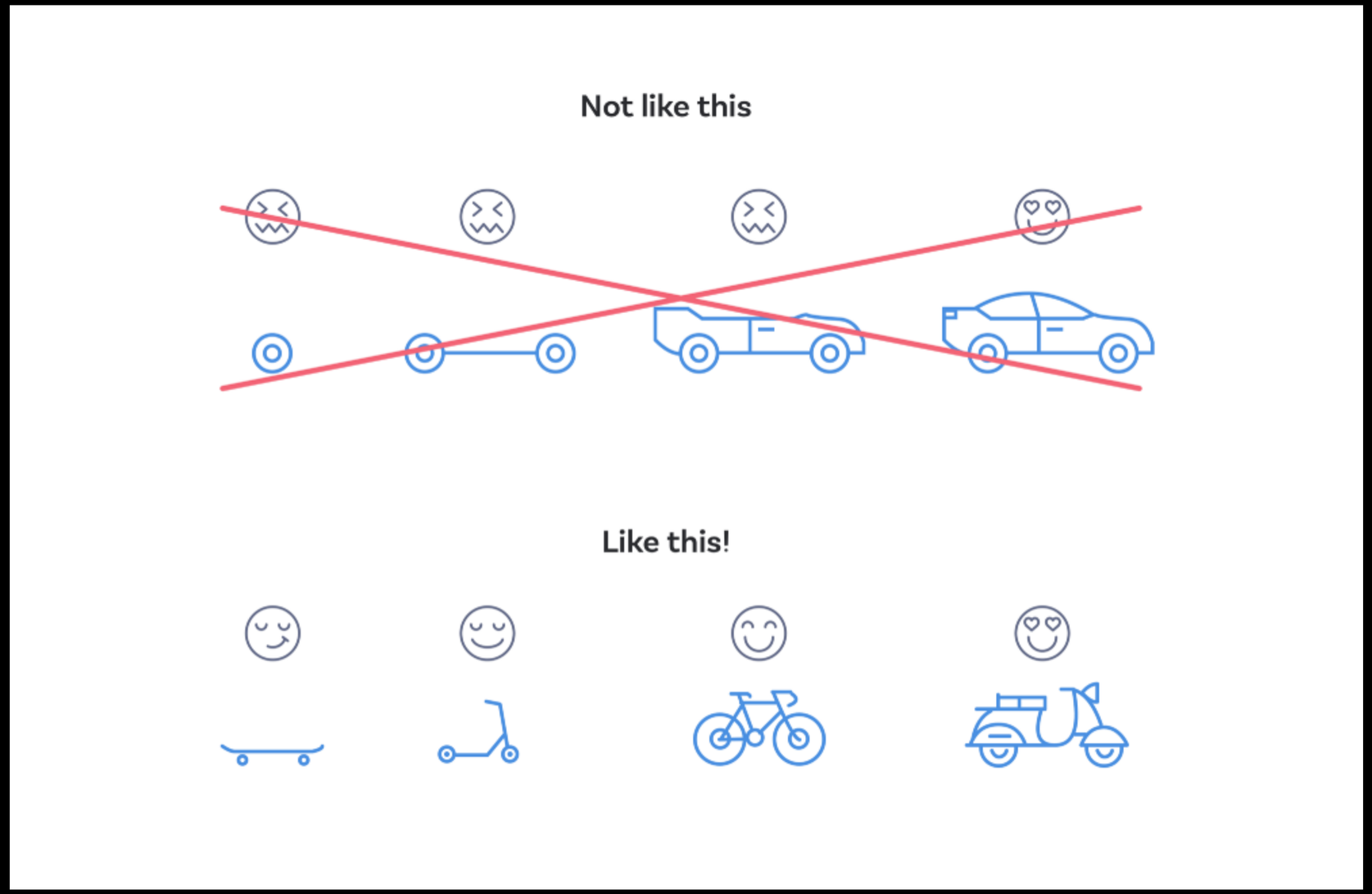
Assessment

description

For your A1 assessment, you're going to implement a *progressively enhanced component.* In short, you're going to enhance the client-side experience of the user by doing research, documenting patterns and implement the principle of *progressive enhancement* using JavaScript.

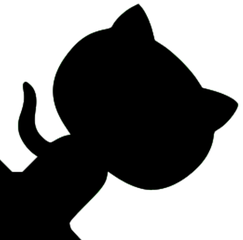
/readme.md


Server-side (Node.js)
Browser
Image
Video
Ads
Client-side
(browser)
GET layout.css
GET ads.jpg
GET video.mp4
Internet
Web server
Ads server
Video server



Andy Bell - The power of enhancement

Discuss (20m)



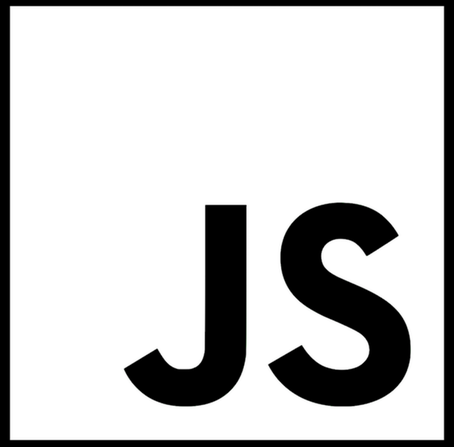
With your partner discuss the idea you have for your progressive enhancement. We'll pick a couple of good examples to discuss with the whole class.

- *What type of component is it (design pattern)*
- *How is it enhanced with CSS?*
- *What client-side JavaScript is used?*
- *Does it use a web api?*
- *Does it work with JavaScript disabled?*

Recap

ES5

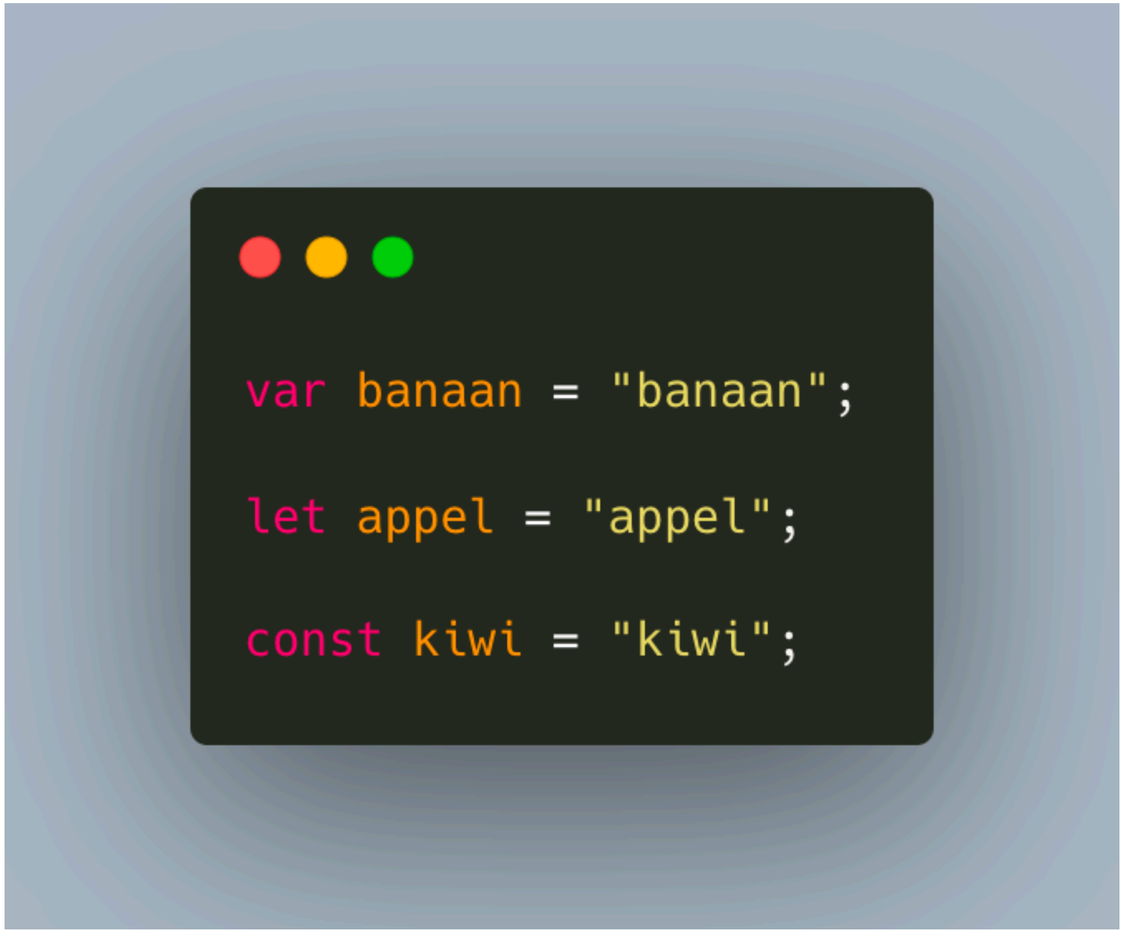
< 2015

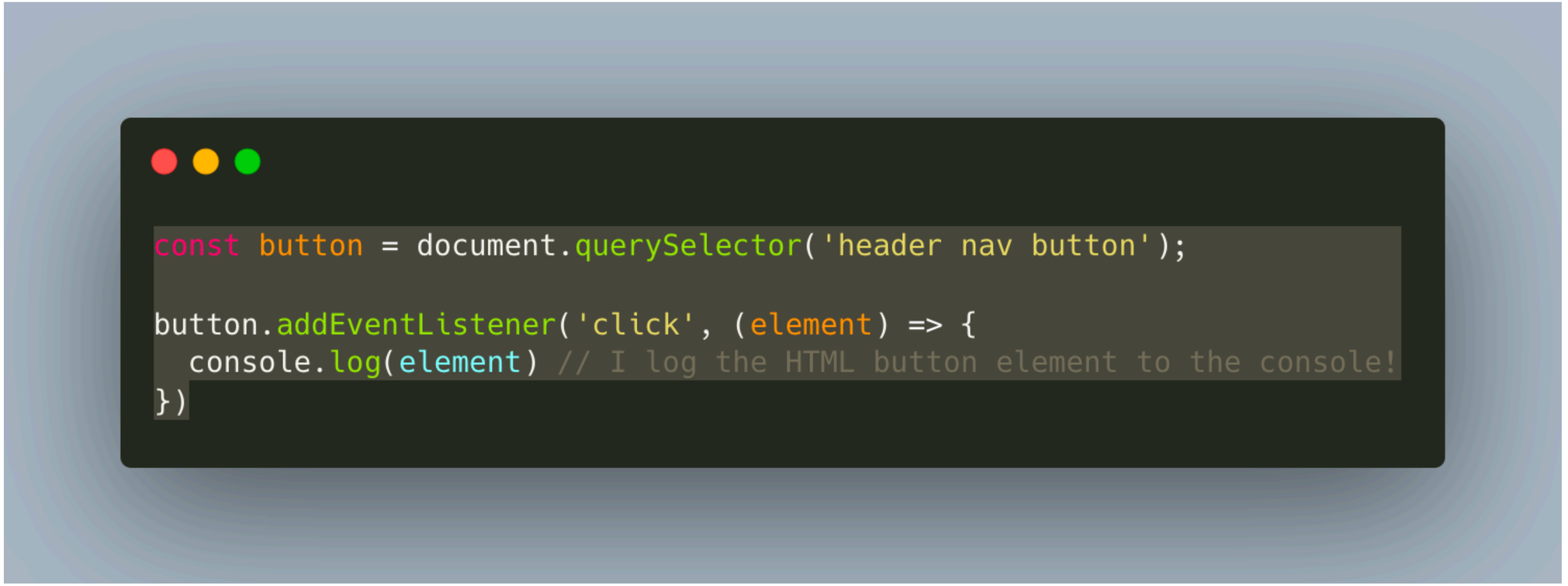


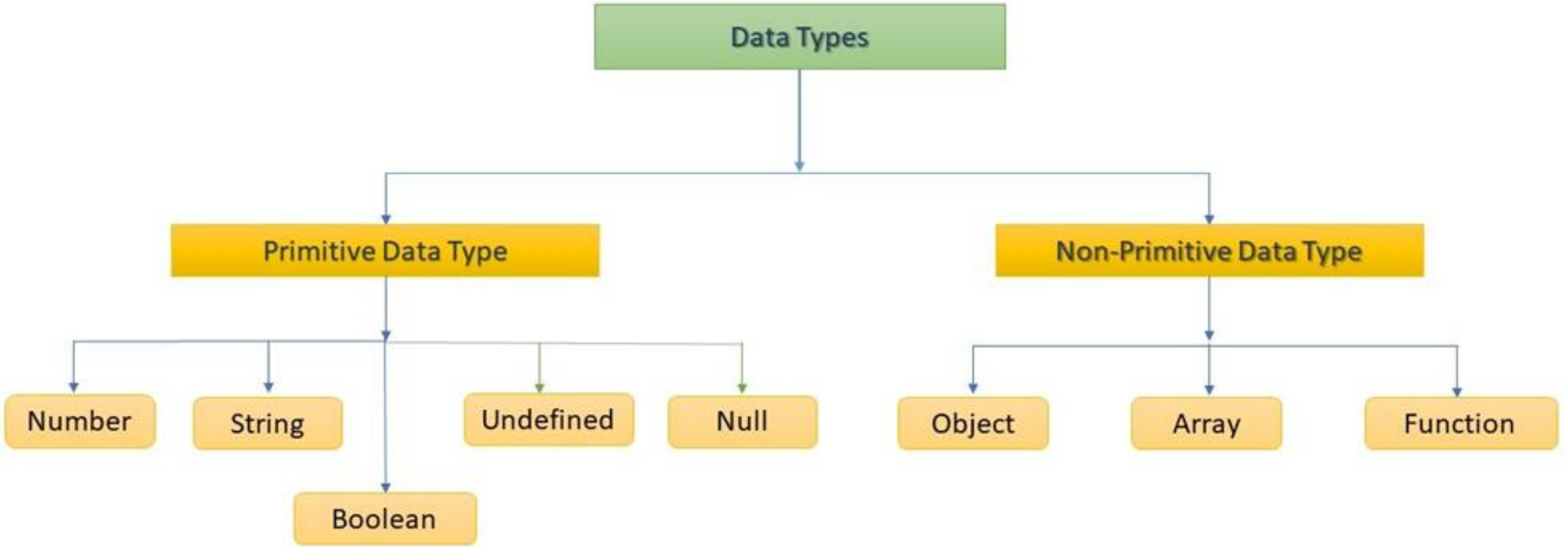
ES6

2015 >

ES6



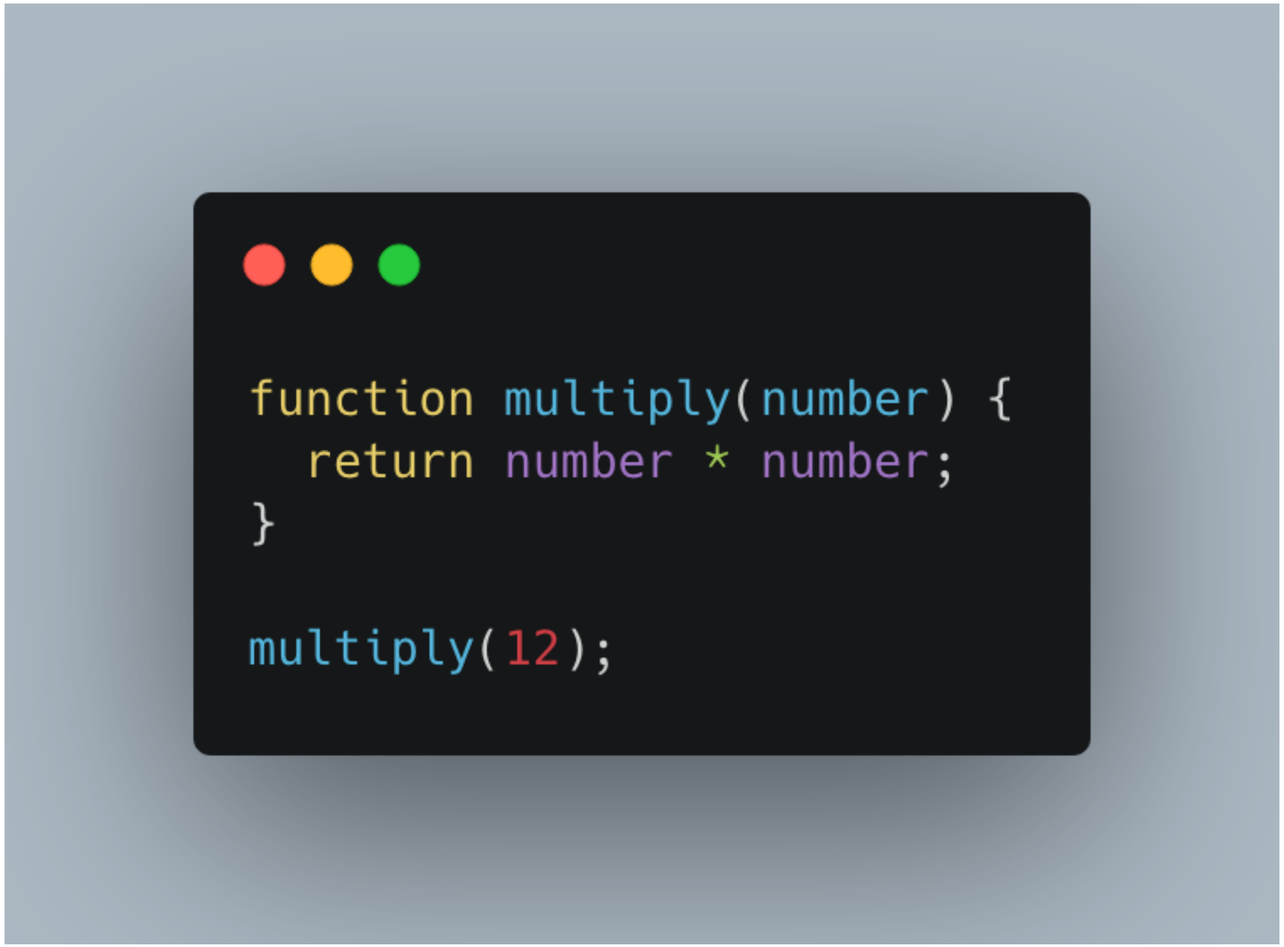

const button = document.querySelector('header nav button');
button.addEventListener('click', (element) => {
console.log(element) // I log the HTML button element to the console!
})

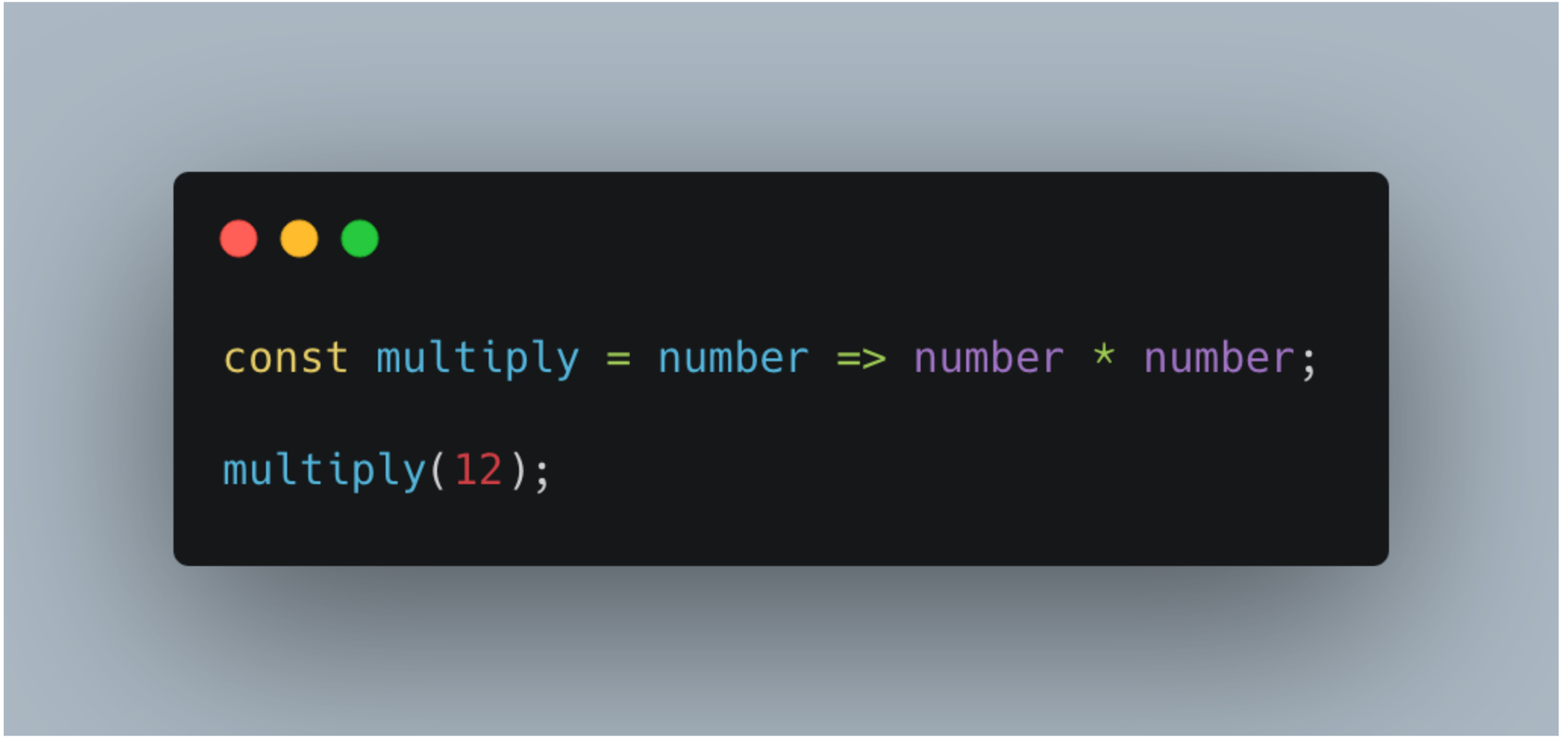

Data Types
Primitive Data Type
Non-Primitive Data Type
Number
String
Boolean
Undefined
Null
Object
Array
Function

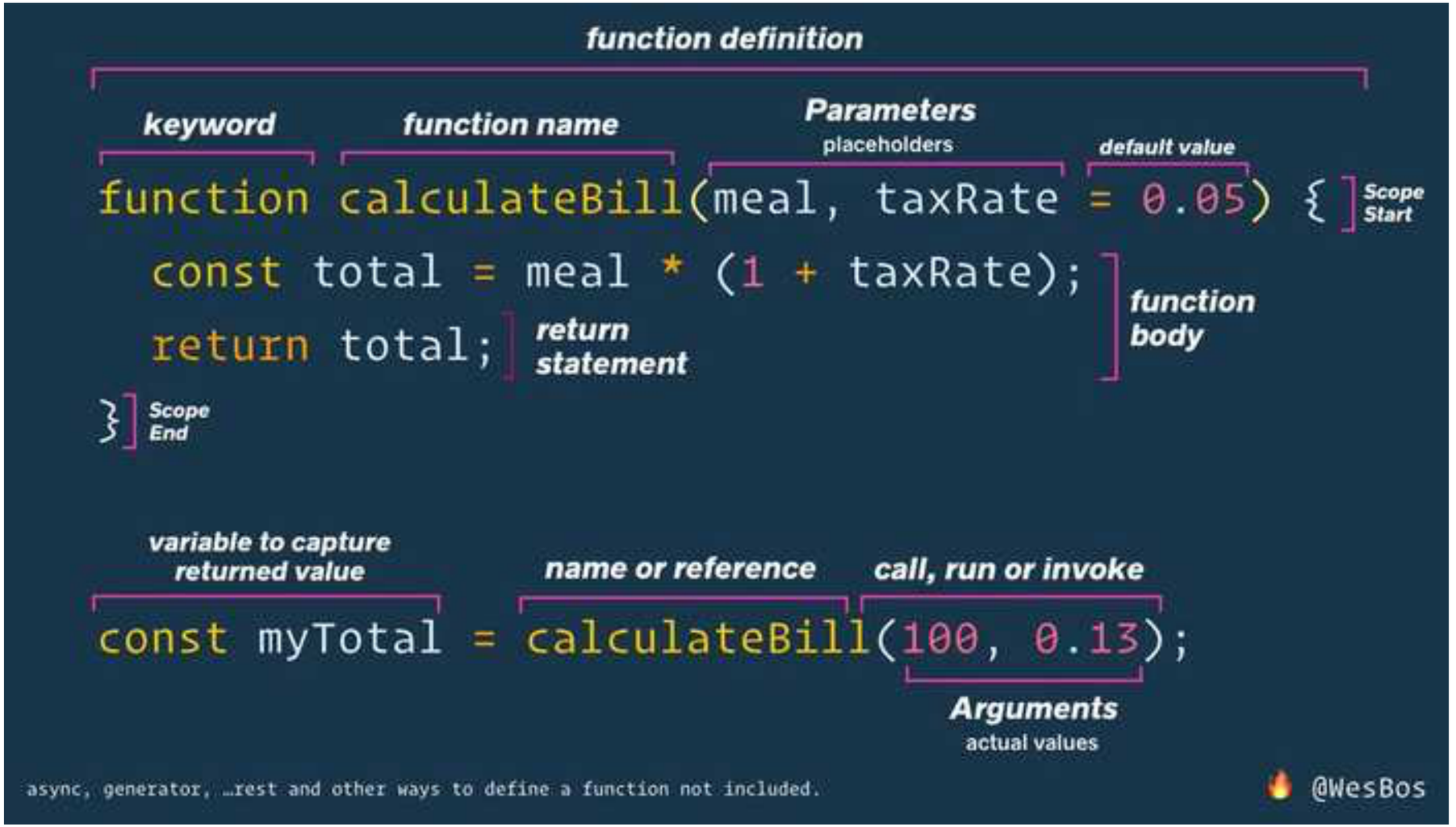

const teacher = {
name: "Robert",
age: 29,
location: ["52.359135", "4.909953"],
hasCar: false
}
console.log(teacher.name) // Yields "Robert"
console.log(teacher['age']) // Yields "29"

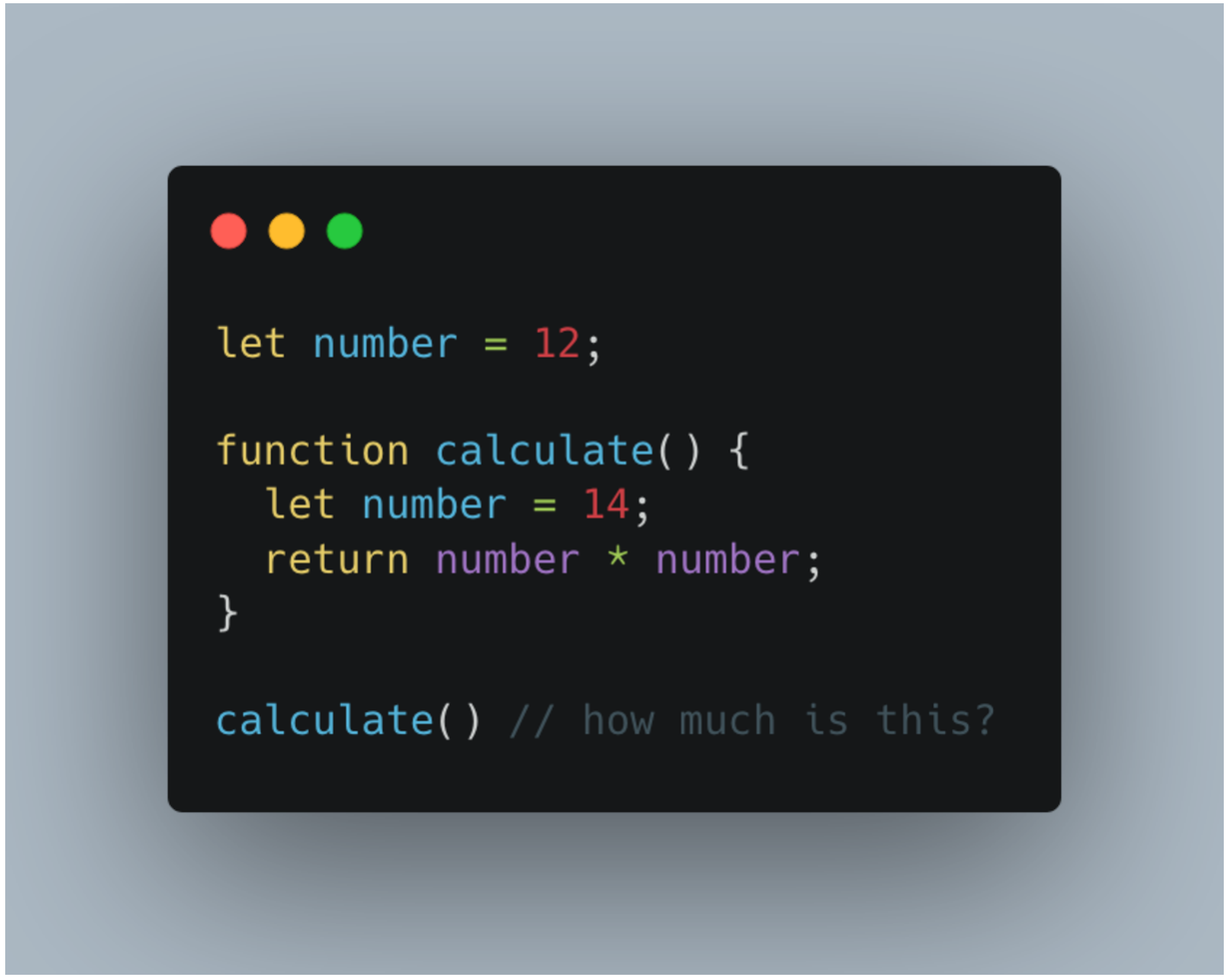
```
const teacher = {
  name: "Robert",
  age: 29,
  location: ["52.359135", "4.909953"],
  hasCar: false,
  sayHi: function() { // We use function() instead of => to preserve context!
    console.log(this) // "this" now points to this object as a keyword!
    console.log(`Hi ${this.name}!`)
  }
}

console.log(teacher.sayHi()) // Yields "Hi Robert!"
```

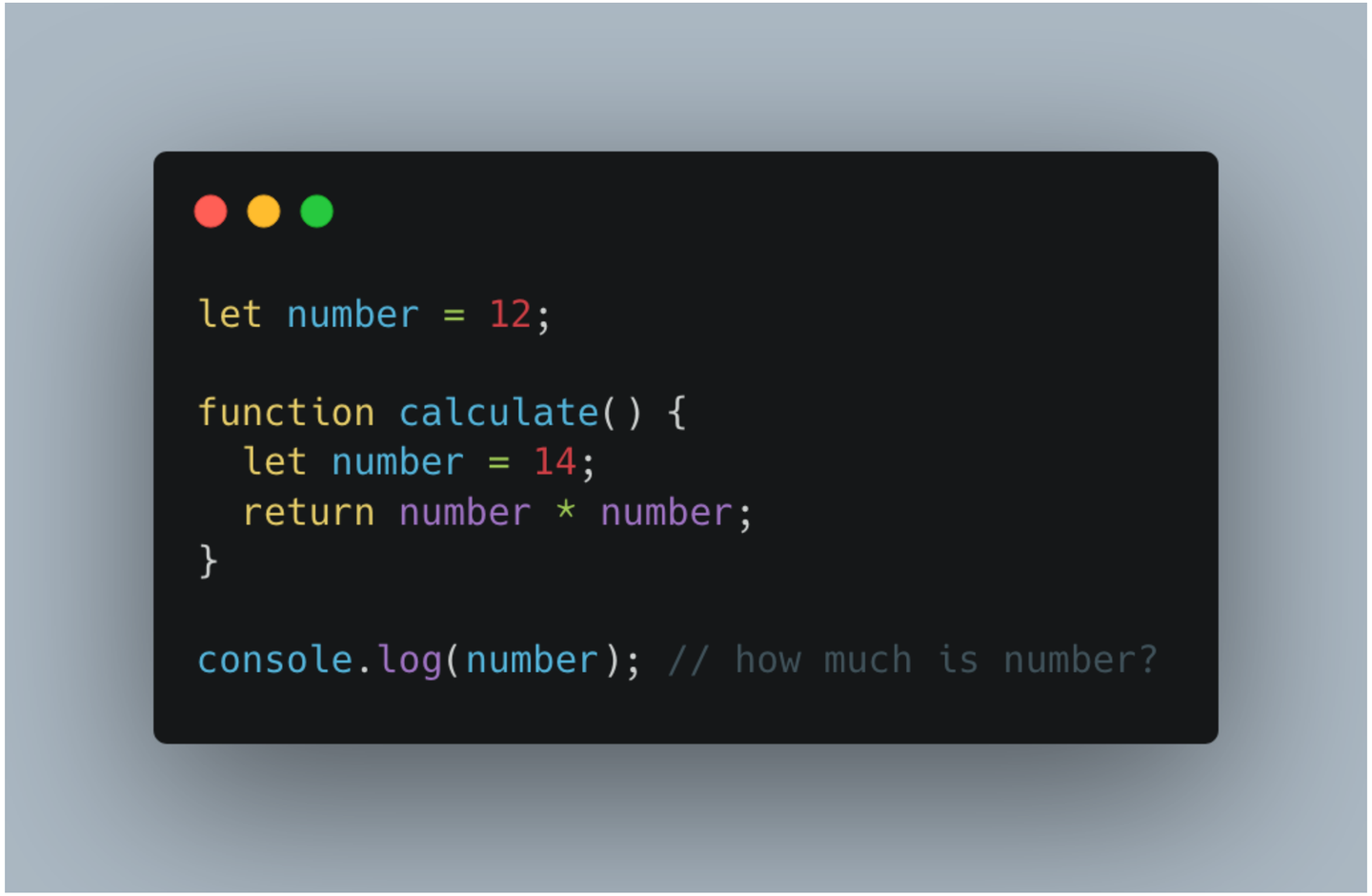
Functions and Scope

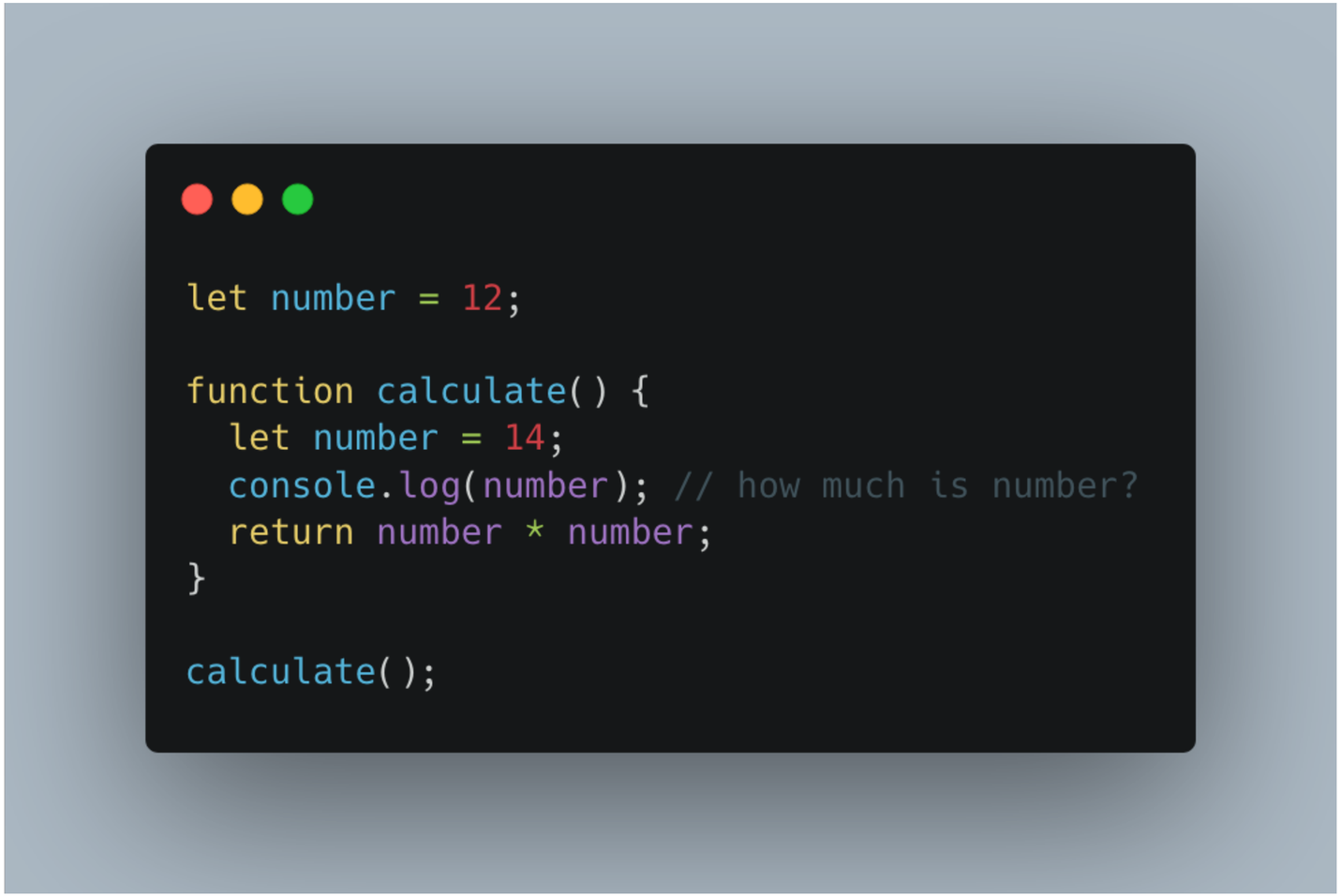

function multiply(number) {
return number * number;
}
multiply(12);

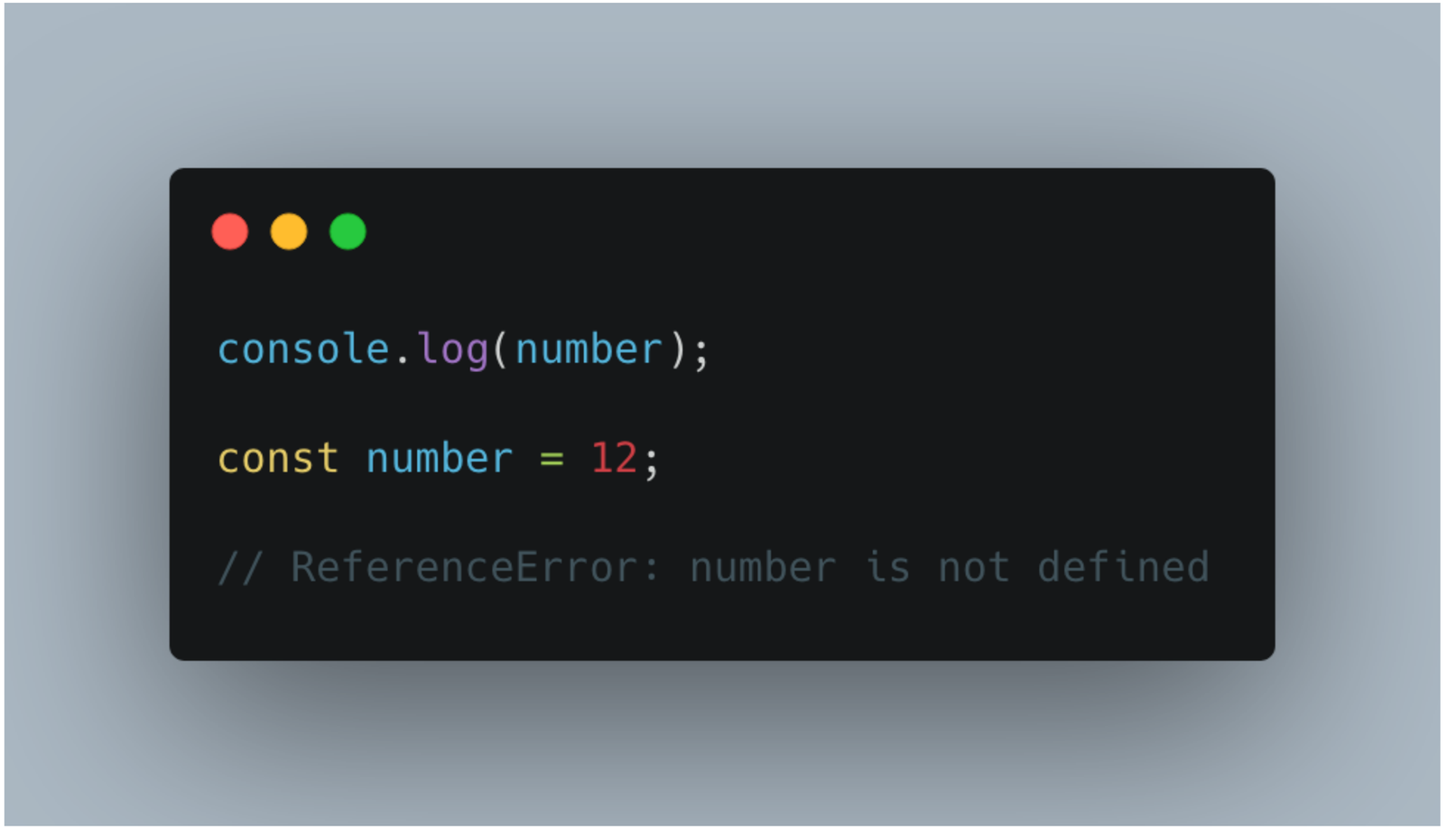

const multiply = number => number * number;
multiply(12);

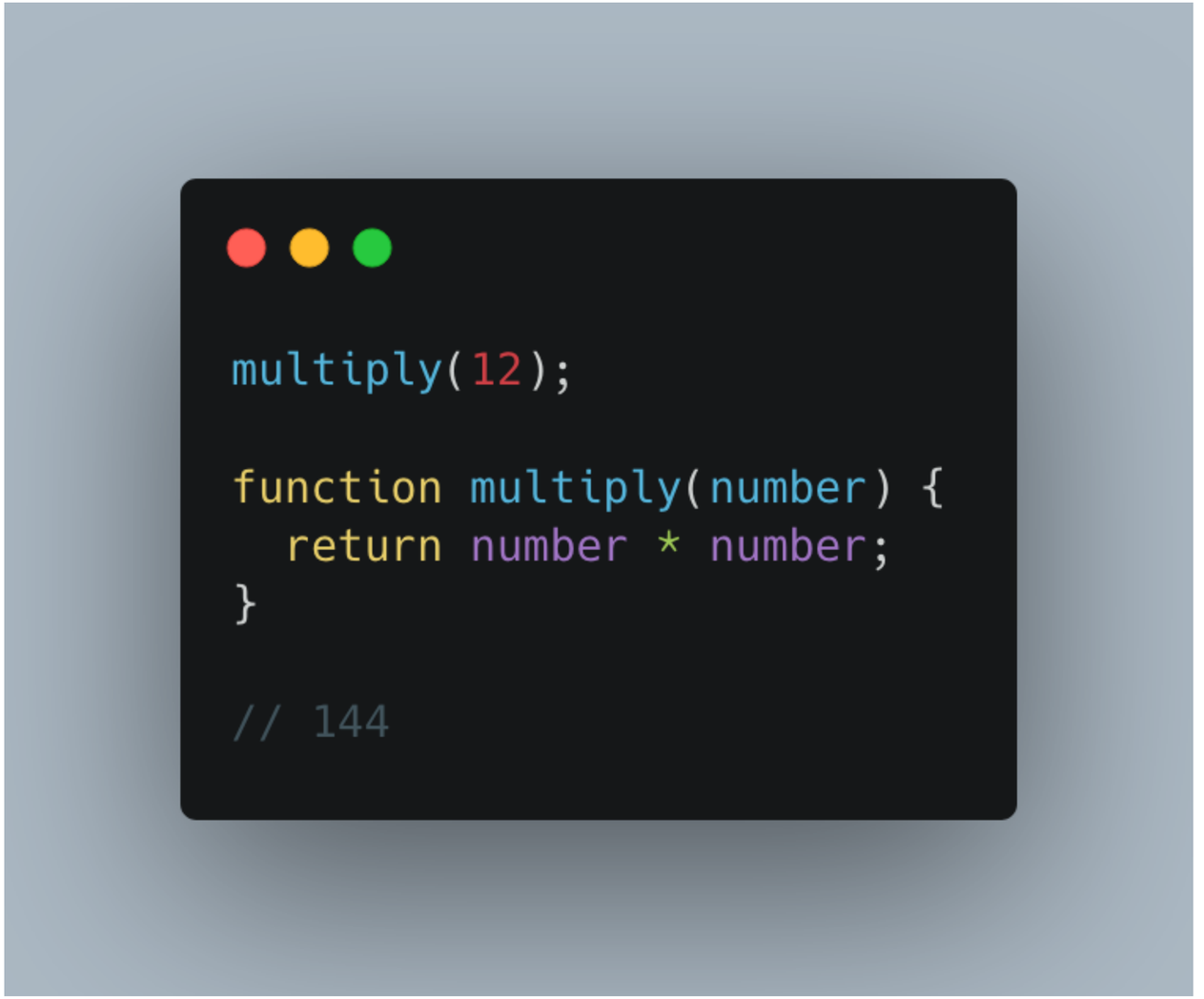

function definition
keyword
function name
Parameters
placeholders
default value
function calculateBill(meal, taxRate = 0.05) {
Scope Start
const total = meal * (1 + taxRate);
return total;
return statement
function body
}
Scope End
variable to capture returned value
name or reference
call, run or invoke
const myTotal = calculateBill(100, 0.13);
Arguments
actual values
async, generator, …rest and other ways to define a function not included.
@WesBos


let number = 12;
function calculate() {
let number = 14;
return number * number;
}
calculate() // how much is this?


let number = 12;
function calculate() {
let number = 14;
return number * number;
}
console.log(number); // how much is number?


let number = 12;
function calculate() {
let number = 14;
console.log(number); // how much is number?
return number * number;
}
calculate();


console.log(number);
const number = 12;
// ReferenceError: number is not defined


multiply(12);
function multiply(number) {
return number * number;
}
// 144

Break!

The DOM

DOM

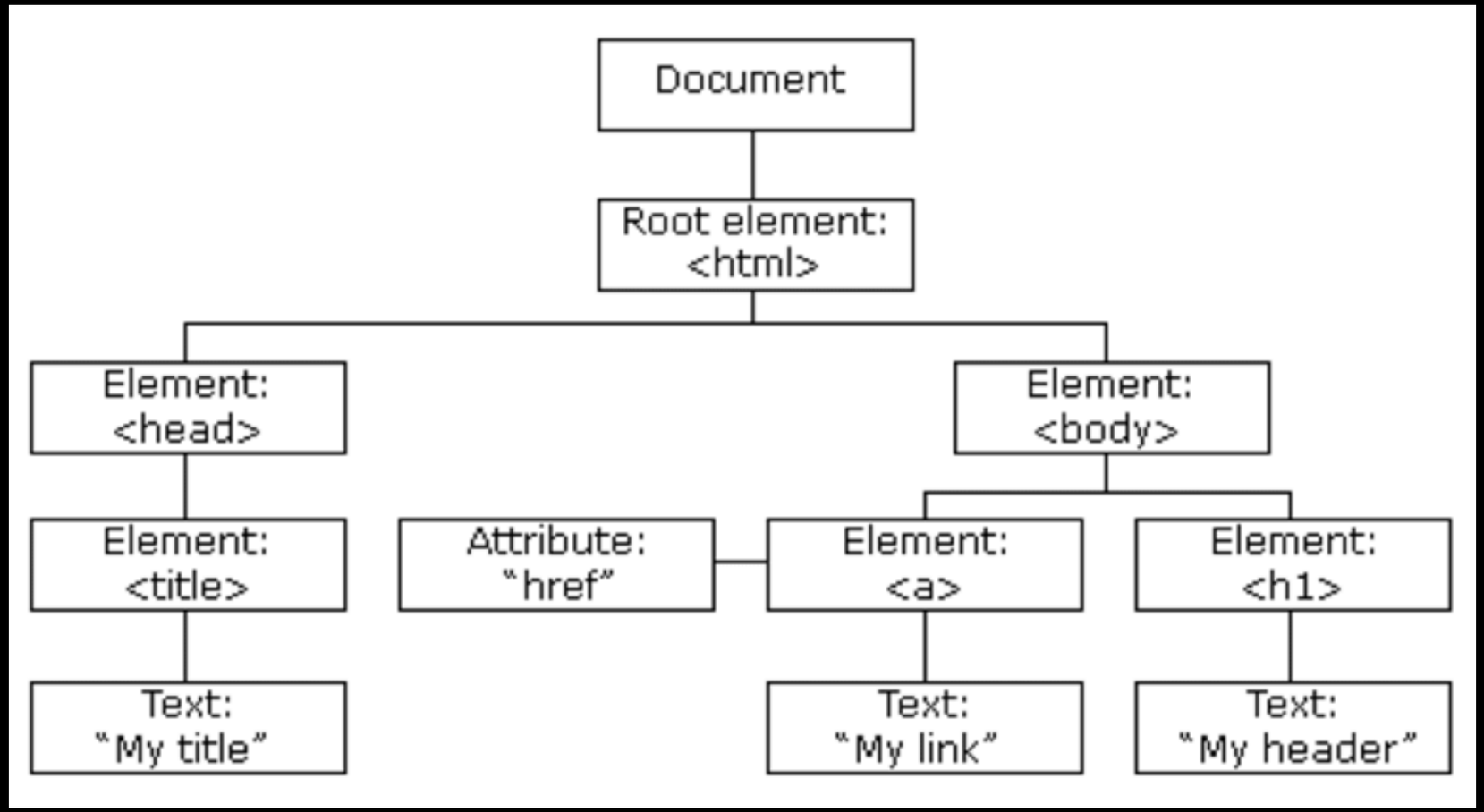
What?

The DOM or **Document Object Model** is the programming interface for web documents.

It represents the **HTML as nodes and objects,** enriched with attributes to provide context.

DOM

What?


Document
Root element:
<html>
Element:
<head>
Element:
<body>
Element:
<title>
Attribute:
“href”
Element:
<a>
Element:
<h1>
Text:
“My title”
Text:
“My link”
Text:
“My header”

DOM

What?

DOM

What?

```
const allElements = $$('*')

console.log(allElements);

> (285) [html, head, link, link, link, link, link, link, link, link,
link, link, link, link, link, link, link, link, link, link, link,
link, link, link, link, link, link, link, link, link, link, link,
meta, meta, meta, meta, meta, meta, meta, meta, meta, meta, meta,
meta, meta, meta, meta, title, link, link, link, link, link, link,
link, link, link, link, link, link, link, link, link, link, link,
link, link, link, link, link, link, link, link, link, link, link,
link, link, link, link, link, meta, noscript, script, script,
script, script, script, script, script, script, script, script,
script, script, script, script, script, script, script, …]
```

DOM

What?

```
// Selects the first <p> tag
let element = document.querySelector('p');

// Adds a click event to the DOM / HTML.
document.addEventListener('click', doSomething);

// Selects the body
let body = document.body;

// Creates a new element and appends it to the body.
let newElement = document.createElement('div')
body.appendChild(newElement);
```

DOM

Array / Nodelist

```
» const teachers = ['Danny', 'Robert', 'Janno',
  'Ivo', 'Sonja'];

  console.log(teachers);
```

```
▶ Array(5) [ "Danny",          debugger eval code:3:9
"Robert", "Janno", "Ivo",
"Sonja" ]
```

Assignment (25m)

Try out working with the DOM and nodeLists. Do the following steps;

- *Create a <ul> with 5 list items in HTML with your hobbies*
- *Append a 6th hobby dynamically with JavaScript*
- *Use queryselectorall to select all list items and loop over them*
- *Add an eventlistener to each listitem and log the hobby to the console*

Events

Events

What?

```
const button = document.querySelector('button');
const menu = document.querySelector('nav');

button.addEventListener('click', openMenu);

function openMenu(e) {
  e.preventDefault();

  nav.classList.add('is-open');
}
```

Events

What?

“Events are **fired** to notify code of "interesting changes" that may affect code execution. These can arise from **user interactions such as using a mouse or resizing a window, changes in the state of the underlying environment (e.g. low battery or media events from the operating system)**, and other causes.”

Events

Reference

Table of contents

Related Topics

- Introduction to events
- Creating and triggering events
- Detecting device orientation
- Event handling (overview)
- Orientation and motion data explained
- Using device orientation with 3D transforms

Event reference

Events are fired to notify code of "interesting changes" that may affect code execution. These can arise from user interactions such as using a mouse or resizing a window, changes in the state of the underlying environment (e.g. low battery or media events from the operating system), and other causes.

Each event is represented by an object that is based on the `Event` interface, and may have additional custom fields and/or functions to provide information about what happened. The documentation for every event has a table (near the top) that includes a link to the associated event interface, and other relevant information. A full list of the different event types is given in Event > Interfaces based on Event.

This topic provides an index to the main *sorts* of events you might be interested in (animation, clipboard, workers etc.) along with the main classes that implement those sorts of events. At the end is a flat list of all documented events.

> **Note:** This page lists many of the most common events you'll come across on the web. If you are searching for an event that isn't listed here, try searching for its name, topic area, or associated specification on the rest of MDN.

Event index

Event type	Description	Documentation
Animation	Events related to the Web Animation API. Used to respond to changes in animation status (e.g. when an animation starts or ends).	Animation events fired on `Document`, `Window`, `HTMLElement`.
Asynchronous data	Events related to the fetching data.	Events fired on `AbortSignal`, `XML-`

Assignment (25m)

Pick an event type from the MDN event reference and log and listen to that event. Examples;

- *Touch events (gestures)*
- *Keyboard events*
- *Document / window loading*
- *Mouse events*

Break!

Web API's

Web Api’s

Definition

A web API is an **application programming interface** for either a web server or a web browser. It is a web development concept, usually limited to a **web application's client-side**.

Web Api’s

Abstraction

Application Programming Interfaces (APIs) are constructs made available in programming languages to **allow developers to create complex functionality more easily.** They abstract more complex code away from you, providing some easier syntax to use in its place.

Web Api’s

Two types

* **Browser API’s;** native and build into your web browser to expose data from the browser and device.

* **Third Party;** usually used for retrieving data. Fetching data from a server.

DOM

Web API???

The DOM is a **Web API.**

It provides us with an interface we can utilise to achieve **interaction**

On the web, we use JavaScript to interact with the DOM[1]

1) However, we can also use languages like Python to access the DOM.

Web Api

Example

Table of contents

Related Topics

Intersection Observer API

The Intersection Observer API provides a way to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document's viewport.

Historically, detecting visibility of an element, or the relative visibility of two elements in relation to each other, has been a difficult task for which solutions have been unreliable and prone to causing the browser and the sites the user is accessing to become sluggish. As the web has matured, the need for this kind of information has grown. Intersection information is needed for many reasons, such as:

- Lazy-loading of images or other content as a page is scrolled.
- Implementing "infinite scrolling" web sites, where more and more content is loaded and rendered as you scroll, so that the user doesn't have to flip through pages.
- Reporting of visibility of advertisements in order to calculate ad revenues.
- Deciding whether or not to perform tasks or animation processes based on whether or not the user will see the result.

Implementing intersection detection in the past involved event handlers and loops calling methods like `Element.getBoundingClientRect()` to build up the needed information for every element affected. Since all this code runs on the main thread, even one of these can cause performance problems. When a site is loaded with these tests, things can get downright ugly.

Consider a web page that uses infinite scrolling. It uses a vendor-provided library to manage the advertisements placed periodically throughout the page, has animated graphics here and there, and uses a custom library that draws notification boxes and the like. Each of these has its own intersection detection routines, all running on the main thread. The author of the web site may not even realize this is happening, since they may know very little about the inner workings of the two libraries they are using. As the user scrolls the page, these intersection detection routines are firing constantly during the scroll handling code, resulting in an experience that leaves the user frustrated with the browser, the web site, and their computer.

Assignment (20m)

Search online (or on MDN) what web API's are supported natively in the browser and try to make a little example.

B

- Background Fetch API
- Background Tasks
- Barcode Detection API
- Battery API
- Beacon
- Bluetooth API
- Broadcast Channel API

C

- CSS Counter Styles
- CSS Font Loading API
- CSS Painting API
- CSS Typed Object Model API
- CSSOM

H

- HTML Sanitizer API
- High Resolution Time
- History API

I

- Image Capture API
- IndexedDB
- Intersection Observer API

L

- Layout Instability API
- Long Tasks API

M

- Media Capabilities API
- Media Capture and Streams

S

- Screen Orientation API
- Screen Wake Lock API
- Sensor API
- Server Sent Events
- Service Workers API
- Storage
- Storage Access API
- Streams

T

- Touch Events

U

- URL API
- URL Pattern API

V

Synopsis

- **Regular lesson**
- **Time:** 1:40h

Table of Contents

Practicum

work on **self-study week-2**

exit;

see you in **lab-3**!