Before we begin...

Download → VS Code

Download → Google Chrome

Welcome!

This Course

Course Requirements

- At least 80% attendance
- Please have your videos on as much as possible
- 80% of homework completed
- 100% projects completed
- Make this as interactive as possible! Ask and answer
- Be on time
- Don't do the bare minimum! Do far more than is asked
- We have a Code of Conduct
  - This works much better if we all are nice and work as a team

How we work at GA

- We start from the beginning
- We adapt as necessary
- We teach industry standards and best practices We
- appreciate your feedback
- We get you to base camp

Course Overview

This is JavaScript, from the ground up. We will start from the beginning and get you pro£cient in the most popular programming language in the world.

There are 4 main parts:

- JavaScript Overview
- JavaScript and the Browser
- APIs, AJAX and Persisting Data
- Front-End Framework (React)

But really...

- Learning JavaScript as a language
- Getting you excited about what you can do with it
- Introducing you to lots of new concepts
- Teaching you how to learn (in a coding context)
- Having fun and working as a team
- Providing you with a platform to continue from

A few thank yous!

- General Assembly
- Public Bank

A few thank yous!

- General Assembly
- Public Bank
- And all of you!

Introductions

Your Lead Instructor: Avinnaash

- Front End Developer || Data Scientist
- Currently...
- Previously...
- I have been teaching for a long time
  - HRDC TTT Certified
  - Research Scientist at University of Malaya
- Email and LinkedIn
- Outside of coding...

How I teach

- The more questions the better
- Interruptions and rabbit holes are always welcome
- I make lots of bad jokes
- I'm not afraid to make it awkward
  - If I ask a question, I will wait until I get an answer
- I'll try to keep it interesting
- I always want to keep learning
- I add more into the slides than we will typically get through

Your TA: Aiman

Take it away, Aiman!

Who are you?

- What is your name?
- What is your coding experience?
- What do you want to get out of this course?
- What do you enjoy? Hobbies, interests etc.

Thank you!

This Class

A Typical Class

- Agenda
- Warmup
- Recap
- Class content
- Break
- Class content

Agenda

- ~~Welcome~~
- ~~Introductions~~
- Class Overview
- Web Development Overview
- JavaScript's Role and History
- JavaScript's Data Types and Variables
- JavaScript's Operators and Conditionals
- JavaScript's Loops
- Review

The Tools We Will Use

- Google Chrome - our browser
- VS Code - our text editor
- Google Chat - for chatting and sharing info
- GitHub - for accessing my code
- Power Point Online - for viewing my slides

How comfortable are you with...

Programming (in years)

How the web works

HTML & CSS

JavaScript

0 (Not at all) 10 (Very)

Is there anything you are particularly interested in?

Annotate anyway!

A Bit of Advice

Getting the most out of it

- Take it slowly
- Pretend you are a computer
- Find parallels and look for real-world applications Fail
- early and often
- Do more than is asked
- Embrace your inner nerd
- Enjoy the little wins
- Solve the problem before you start coding
- Consciously read code
- Work together

The Pitfalls

- Getting too advanced too quickly
- Judging yourself
- Comparing yourself to others
- Forgetting about the user
- Not taking advantage of what you have here
  - Your classmates
  - Your TAs
  - 1-on-1s

Resources you can rely on

- MDN
- You Don't Know JS book series
- JavaScript.info
- Eloquent JavaScript
- Dash
- Codecademy
- FrontEnd Masters (paid)
- Egghead (paid)

Overview of the Web

What is the web?

Billions of connected devices through a series of networks

Is the internet and the web the same thing?

What is the web?

Billions of connected devices through a series of networks

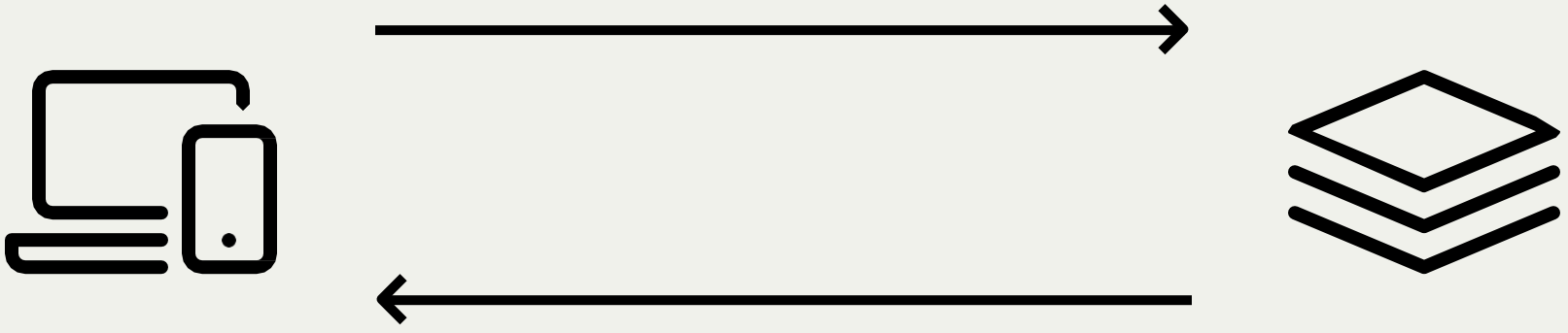
Is the internet and the web the same thing?

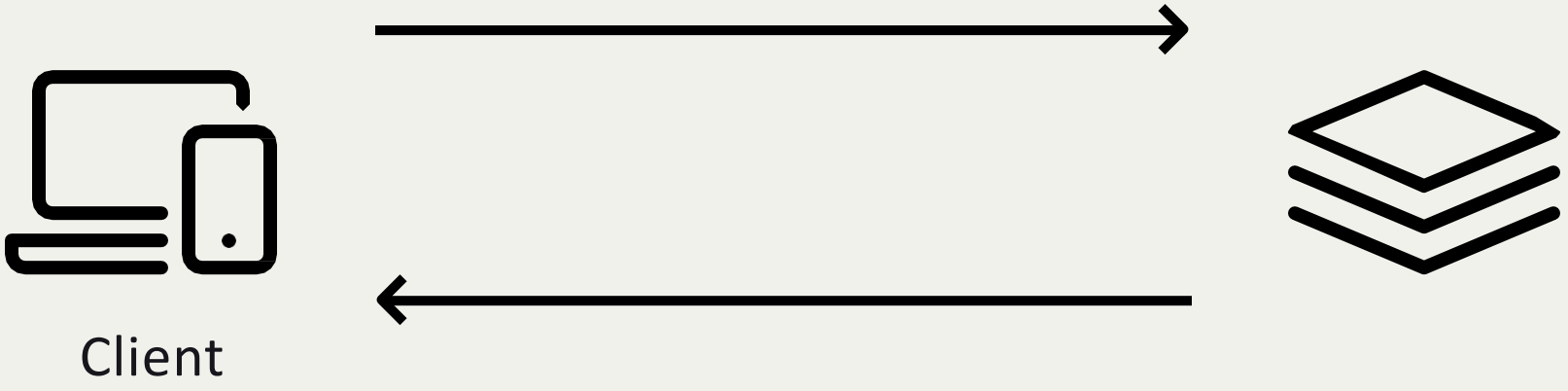
No! The web is the physical network of devices, the internet is the virtual network of information

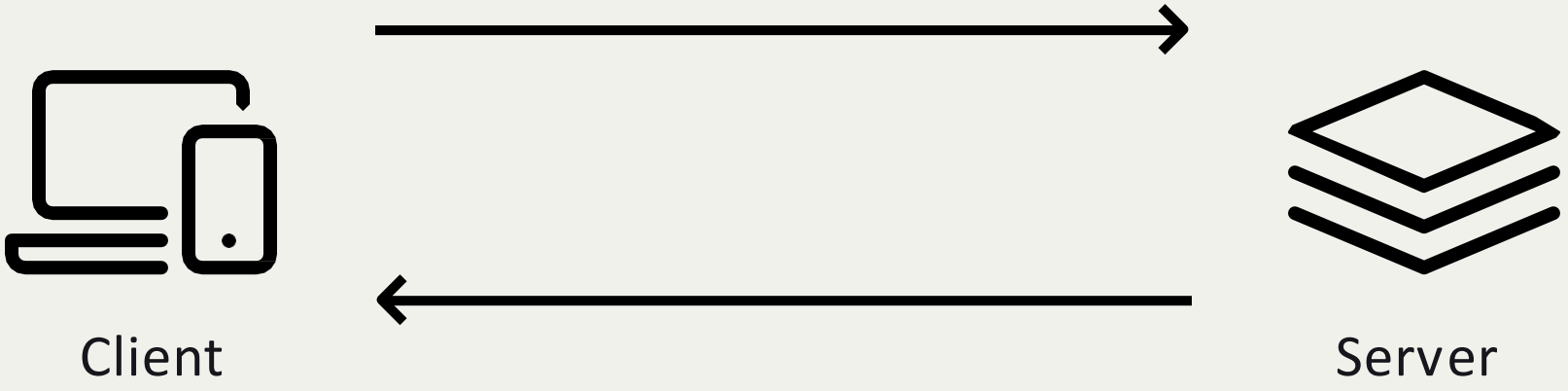
A bit of history...

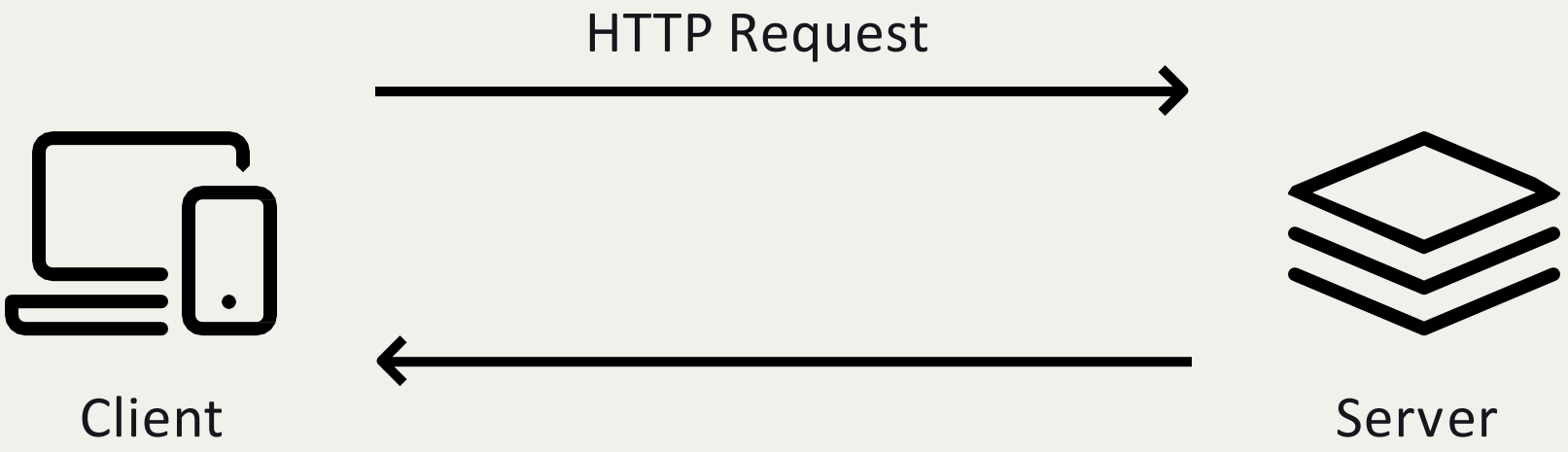
- J.C.R. Licklider came up with 'The Galactic Network' (Aug. 1962)
- ARPAnet in the 1960's for the US Government
- Vint Cerf and Bob Kahn invented TCP/IP in 1974
- Then, Tim Berners-Lee released the "World Wide Web" in 1991, with this site

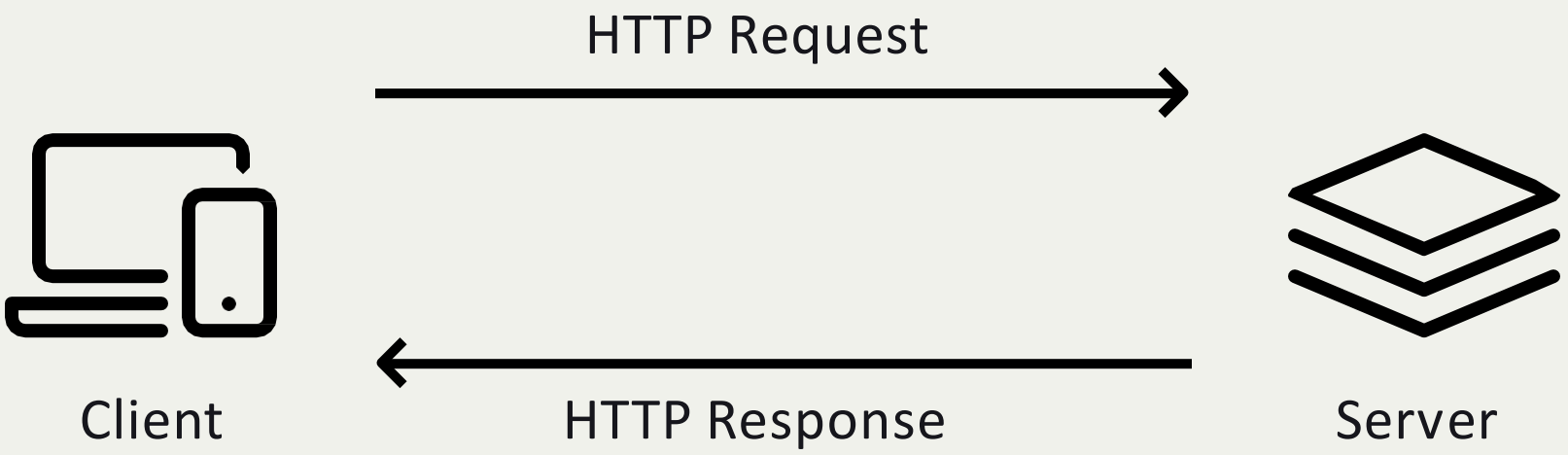
How does it work?

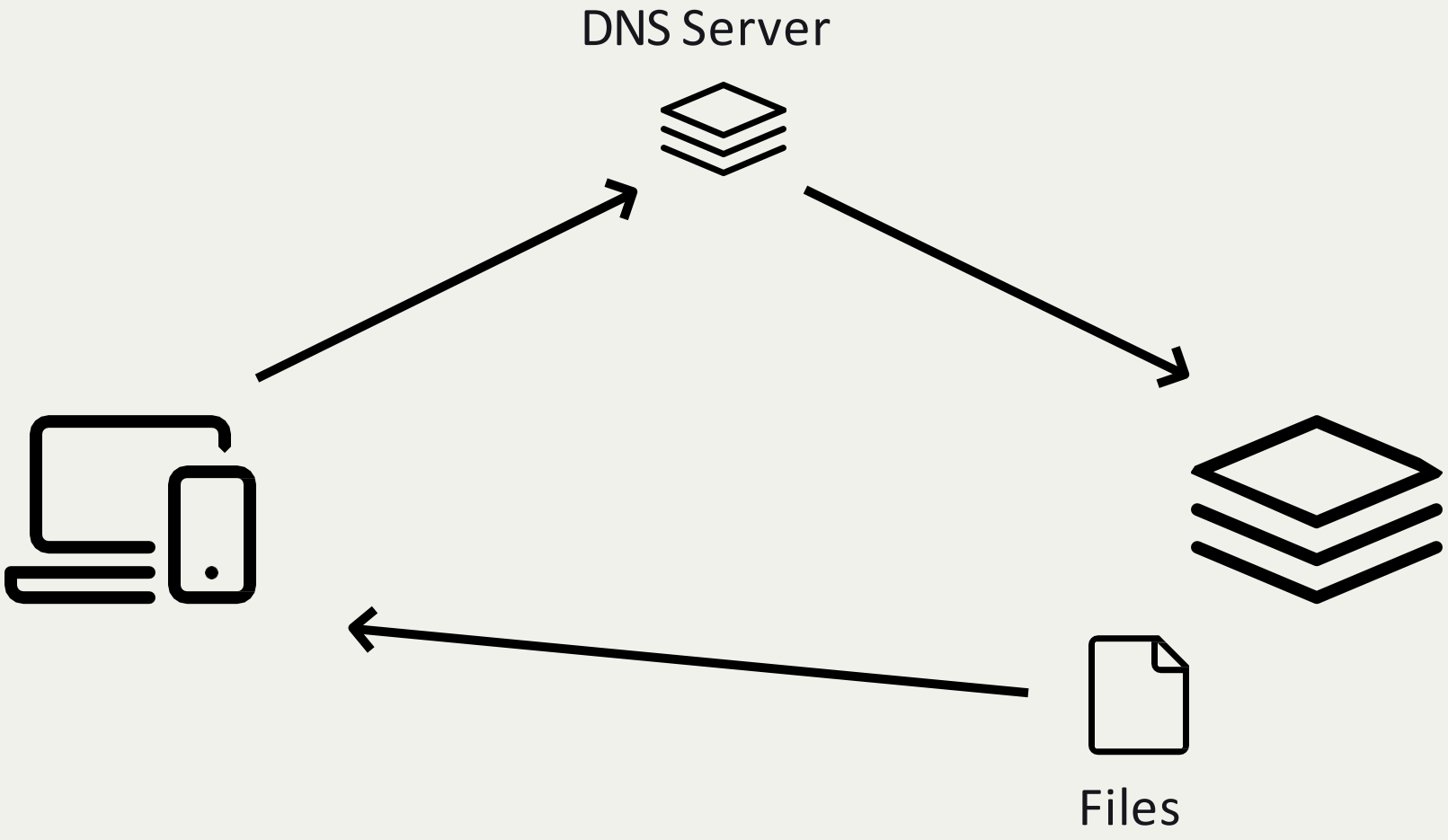


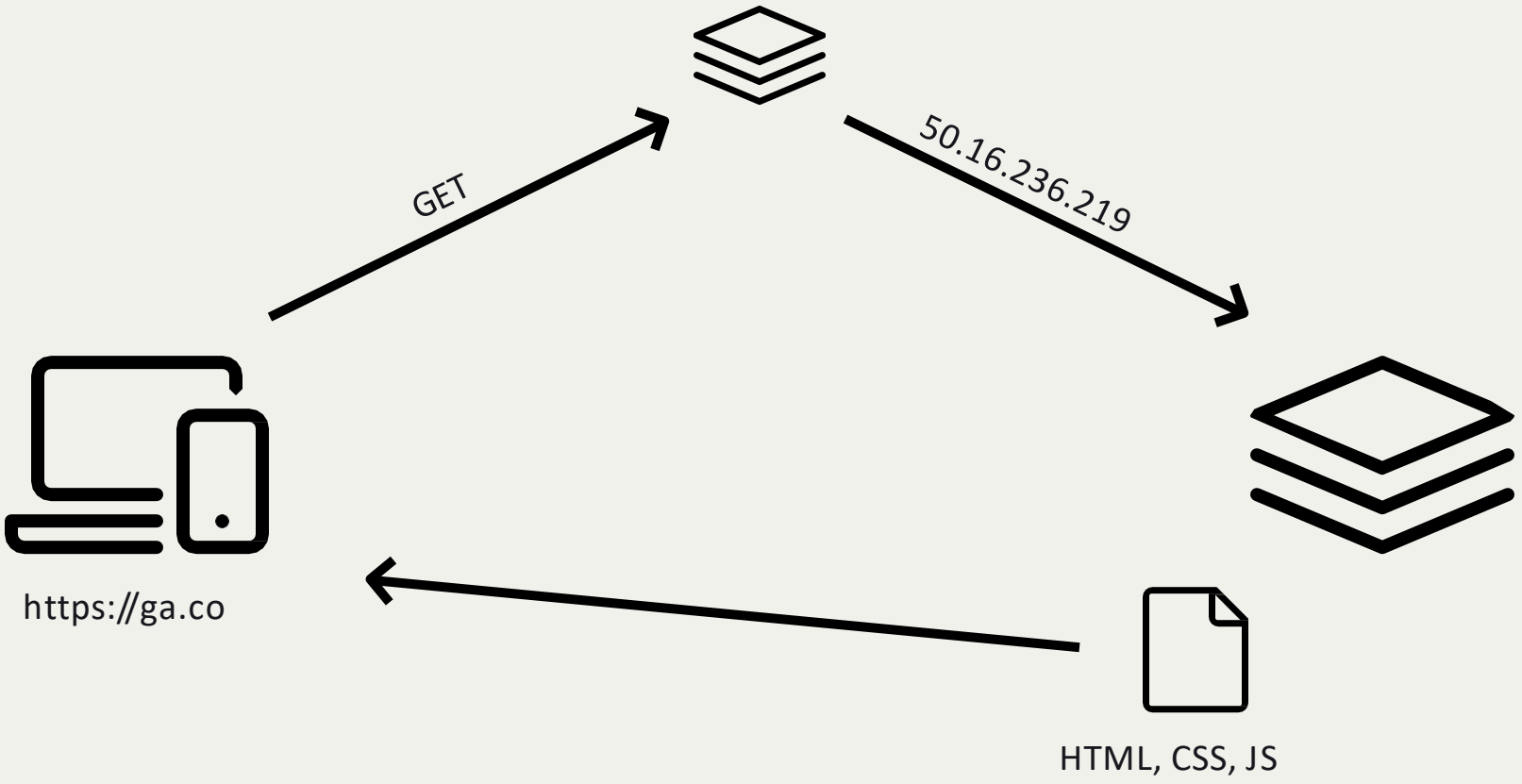

Client


Client
Server


HTTP Request
Client
Server


HTTP Request
Client
HTTP Response
Server


DNS Server
Files


GET
50.16.236.219
https://ga.co
HTML, CSS, JS

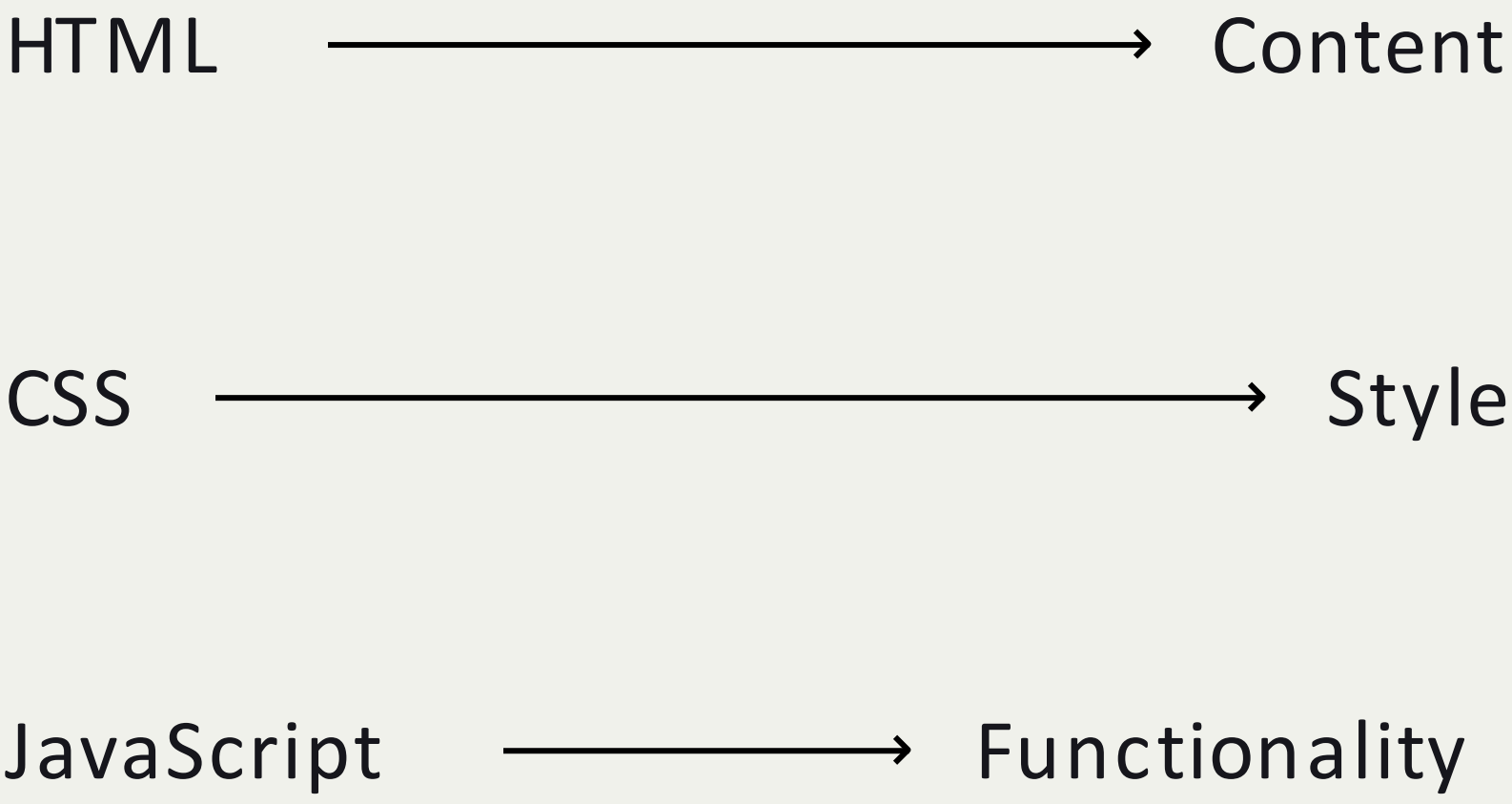
Web Development

What is it?

- The creation and management of websites and web applications
- Made up of the **Front-End** and the **Back-End**
- Most developers will try to convince you that it's magic

The Front-End

- What the user sees
- It powers the visuals and interactions of the web
- Made up of three languages:
  - HTML
  - CSS
  - JS

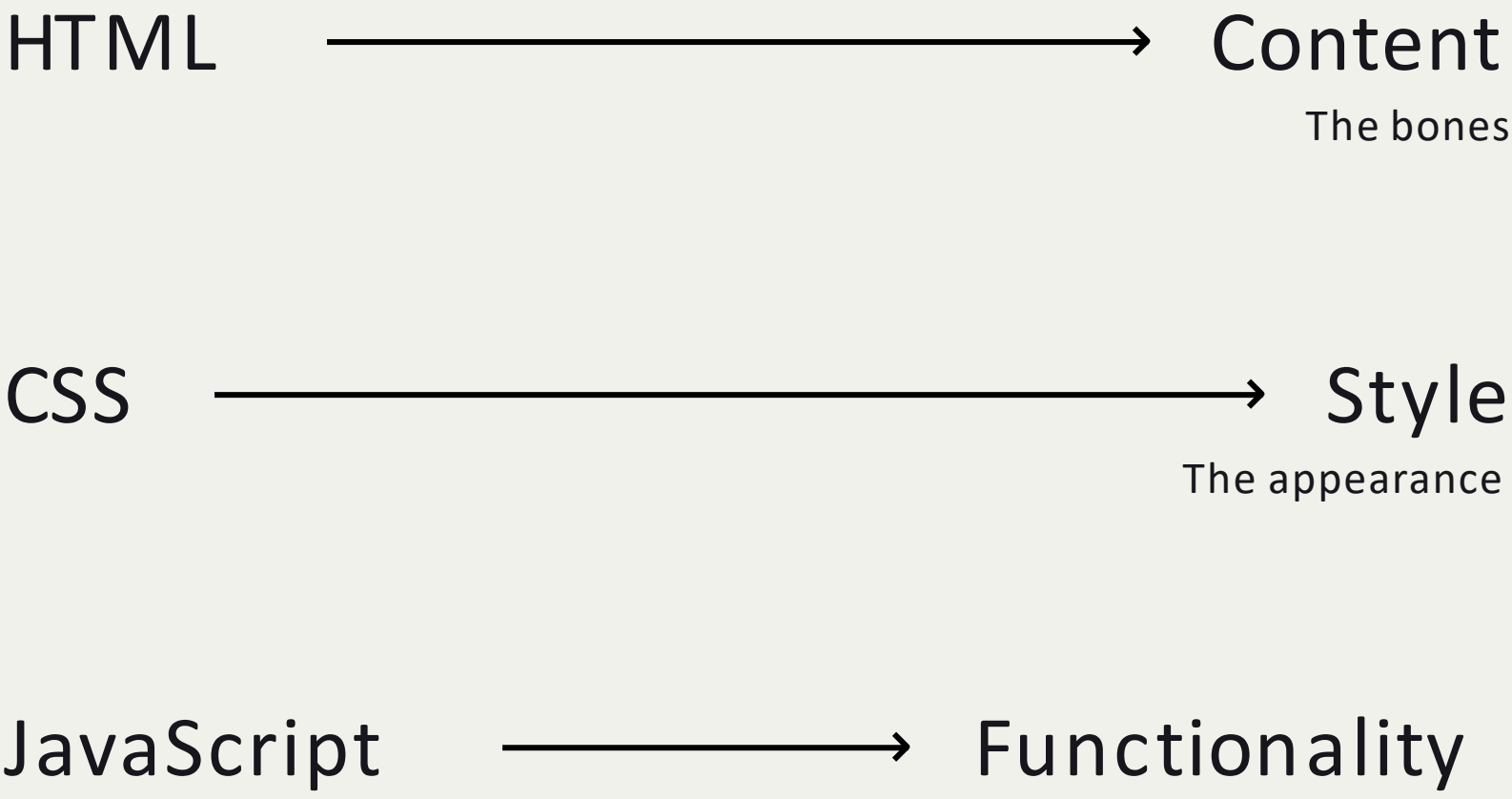

HTML
Content
CSS
Style
JavaScript
Functionality

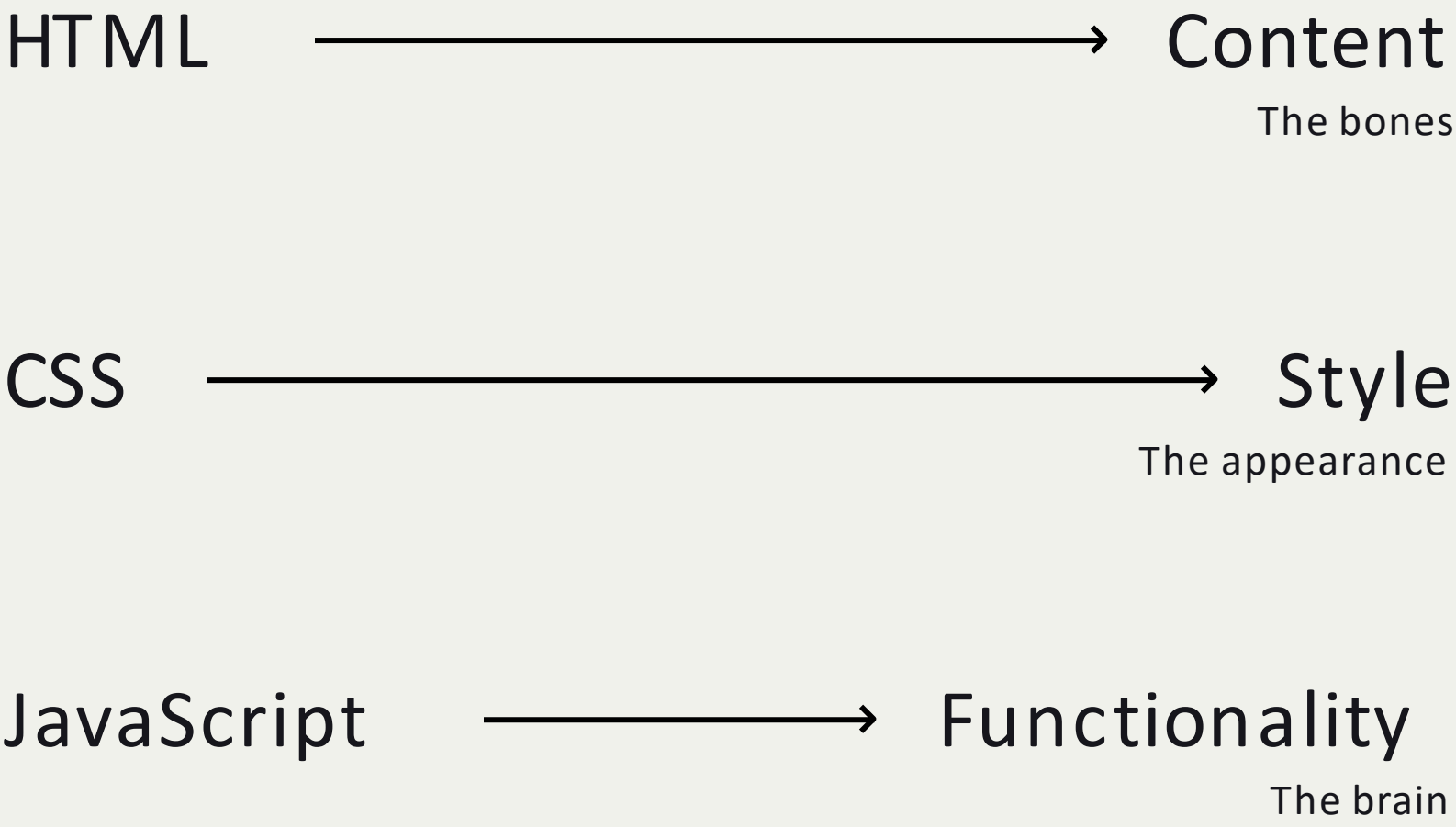
HTML → Content

The bones

CSS → Style

JavaScript → Functionality


HTML
Content
The bones
CSS
Style
The appearance
JavaScript
Functionality


HTML
Content
The bones
CSS
Style
The appearance
JavaScript
Functionality
The brain

The Back-End

- What goes on behind the scenes
- Consists of databases, servers, processes etc.
- Can make payments, send emails, store assets etc.
- Lots of different programming languages
  - Node.js, C++, Python, Ruby, Elixir, ASP.net etc.
- We will see this a little bit (but likely not very much)

JavaScript Overview

What is it?

- The most popular programming language (according to Stack Overflow, GitHub and GitHut)
- Originally created to make websites alive
- It is a very flexible programming language
  - Runs in the front-end, in the back-end and other places too!
- The programs in JS are called scripts. They are just ordinary files - no preparation or compilation needed
- It is a loosely typed and dynamic language (we will go into this later)

What can it do?

- Validating information
- Change a page's HTML and CSS
- Request information from APIs
- Adding interactivity and animations (e.g. TweenMax)
- Internet of Things and Hardware
- Visualise data (e.g. D3.js, DeckGL)
- Can be used for art (e.g. P5.js, PaperJS)
- 3D (e.g. ThreeJS, ReGL), Games (e.g. Phaser), AI, Augmented/Virtual Reality (e.g. Aframe, AR.js, MozVR)
- Plus more!

How will we run/test it?

- Through our own files
  - We will have an HTML file that references a JS file
- Through a REPL (useful for debugging and learning)
  - A Read, Evaluate, Print Loop
- Through the Command Line (eventually)

Chrome Developer Tools

- This is the REPL we will be using
- Exceptionally useful for learning and debugging
  - Not for writing projects in!
- To access it:
  - Mac: CMND + OPTION + J
  - Windows/Linux: CTRL + SHIFT + J
  - OR: View -> Developer -> JavaScript Console
- Let's take a look

Our Own Files

- Create a folder and open it in VS Code
- Create two files, an index.htmland an index.js
- Set up your HTML page (hint: !then tab)
  - Add a script tag and link up your index.js file
- In your JS file, write some code:

```
console.log("Hello World");
```

- Save both files, then open the HTML file in Chrome (and open the Dev Tools)

console.log

```
console.log("Hello World");
```

This is one of the ways we debug and test applications. Anything that is provided to the console.log function will appear in the Developer Tools (and nowhere else!). It's useful for leaving ourselves notes, seeing the order of how things execute, seeing the current value of variables etc.

Let's have a break!

See you in 10 minutes!

JavaScript Basics

What does JavaScript give us?

- A syntax
  - The skeleton of our code (what characters go where)
- Data Types
  - What we use to build
- Ways to save, access and manipulate data
  - Variables - saving things for later on!
- Ways to work with APIs

Data Types

What are Data Types?

- The building blocks of every script
- The types of information the language provides
- Any value in JavaScript has a certain data type
  - e.g. a number or a string
- There are 9 types in JavaScript, broken down into 3 categories:
  - Primitive Types
  - Structural Types
  - Structural Root Primitive

What are Primitive Data Types?

They are the basic building blocks of the language. They are ***immutable*** - this means that they cannot be changed.

What primitive types are there?

Number

Boolean

String

undefined

BigInt*

Symbol*

* Very new, and rarely used. We will ignore them for the time being

What are Structural Data Types?

Data types that are ***mutable*** - this means that they can be changed. They are constructed by combining one or more other primitive or structural data types. They aren't really data - they just provide structure to data.

There are technically only two structural data types:

- Objects*
- Functions

* This includes things like Arrays, Objects, Dates etc.

But for our purposes...

Structural (or composite) data types can take one of three forms:

- Arrays
- Objects*
- Functions

We will learn more down the track though!

* Remember that technically all structural types other than functions are objects

Primitive Data Types

Numbers

Numbers

The number type in JavaScript represents both **integers** and **floats** (decimals or oating points)

```
42;

124152152152;

-15;

-101245125;

1.52;

-18.92;
```

Numbers

We can perform operations such as addition (+), subtraction (-), multiplication (*) and division (/)

```
// Addition
15 + 6;

// Subtraction
25.12 - 13.25;

// Multiplication
18 * 7;

// Division
24 / 4;
```

Did you notice the...

Semicolon at the end of the line? This is how we end **most** lines in a JavaScript file.

```
14 + 29;
```

The lines that started with // ? This is one of the ways we **write comments** in JavaScript. The computer ignores these lines!

```
// Very useful for writing notes to yourself!
```

Booleans

Booleans

The logical type in JavaScript. It comes in two forms: `true` and `false`. It's commonly used to make decisions about what should occur. `true` means "yes, correct, continue" and `false` means "no, incorrect, skip or stop".

```
true;

false;
```

Strings

Strings

Used to represent textual data. It can be empty, contain one character or have many characters.

They must be surrounded by quotation marks!

```
"I am a string"; // Double Quotes

'I am also a string'; // Single Quotes

`I am a very fancy string - more on me later`; // Back Ticks
```

The different types of quotation marks are ***mostly*** interchangable.

Strings

```
"I'm double quoted, so this is fine";

'She said "This, here, is single quoted"';
```

To **escape** special characters in strings, use the back slash!

```
"But he said \"What if I need to escape the quote?\"";

'What\'ll I do if I need to escape?';

`I said: "I don't have these problems, plus I can interpolate"`;
// More on this later...
```

Concatenation

This is how we combine multiple strings. If you want to add something in that isn't a string, that's okay (most of the time) too - JavaScript will try to convert it to a string for you!

```
"Hello " + "World";

"We are learning " + "JavaScript.";

"The result of 1 + 1 is " + 2;

// It would be useful if these were dynamic, right?
//        That's where variables come into play! You'll see them soon
```

Properties and Methods

When you create something from a JavaScript data type, ***properties*** and ***methods*** get attached to the data.

Properties are pieces of information about the data

```
"Hello".length;
```

Methods are operations you can ask JavaScript to perform on the data.

```
"my string".toUpperCase(); // Notice the parentheses (round brackets)!
```

undefined

undefined

It means that there is nothing here! Either the value does not exist yet, or it does not exist anymore. I treat this as implicitly empty and mostly just let JS use it, rather than me using it directly..

```
undefined;
```

null

null

Again, it means that there is nothing here! But this, to me, is explicitly empty. I directly use this regularly if I want to say that something absolutely has no value.

```
null;
```

*nullis technically a structural root primitive

Structural Data Types

Arrays

Arrays

An ordered collection of any information, where you can access data with a zero-based index (the first item is at index 0, the second is at index 1 etc.).

They are able to be iterated, or looped, through

```
[]; // An empty array

[ "a", "b", "c" ];

[ 1, "", null, undefined ]; // Primitives we have seen

[ "First Todo", "Second Todo" ];
```

Objects

Objects

An unordered collection of key-value pairs (like a word and definition in a dictionary) that can store any data.

```
{}; // An empty object

{ name: "Avinnaash" }; // An object with one key and value

{
    course: "JavaScript Development",
    provider: "General Assembly",
    topics: [ "JavaScript", "React", "Firebase" ],
    numberOfHours: 60
};
```

Functions

Functions

The main building blocks of programs - they are reusable sections of code (almost like subprograms).

Useful for when we need to perform a single action in many places of the script, and for reducing repetition.

```
function showMessage() {
    alert( "Hello Everyone!" );
}
```

Notice how the function has a name? We can give names to all of our data!

Variables

What are variables?

- Named storage for storing, accessing, and manipulating data
- They are ways to store information in memory
  - You assign a name to a certain piece of data. Think of them as named buckets

To create one, we use the **let** keyword

```
let message;
let name;
```

let

Creating variables using let

To create a variable, we declare it with a name and then we assign a value to it.

```
let message; // Declaration

let animal;
animal = "Orca"; // Assignment

// Declaration and Assignment (suggested approach)
let course = "JavaScript Development";

let alphabet = [ "a", "b", "c" ];
```

Changing variable values

```
let animal = "Orca"; animal = "Wolf";

let favouriteNumber = 1; favouriteNumber = 42;

//        JavaScript is a dynamic programming language
//        Which means they can change types too!

let myVariable = null;


myVariable      = 10;
myVariable      = "String";
myVariable      = false;
myVariable      = [];
```

Variable Naming

- The name must contain only letters, digits, $ or _
- The first character must not be a digit and there can be no spaces
- Use *camelCase* when the name contains multiple words. It looks like this:

```
let myMultiWordVariable;
let thisCanGoOnAndOnAndOnAndOn;
let favouriteBook;
```

Some common gotchas

- There are some reserved words that cannot be used. See here for more details
- Names are case sensitive, meaning animaland Animalare not the same variable
- Declaring a variable with the same name twice in one scope will throw an error

```
let animal = "Orca";

let animal = "Wolf"; // SyntaxError: 'animal' has already been declared
```

Some good-to-follow rules

- Name your variables in a human-readable way
- Steer clear of abbreviations
- Be descriptive AND concise (if possible)
- Use camelCase

const

What is const?

It's very similar to `let`. The only difference is that it has an immutable binding (meaning the data it points to can't change). People often name these using `UPPER_SNAKE_CASE`.

```
const FAVOURITE_NUMBER = 42;

const COURSE = "JavaScript Development";

COURSE = "Marine Biology"; // This won't work - TypeError.
```

var

What is var?

`var` is very similar to `let` (essentially the older way of doing it), with a few slight differences. The main differences are that they:

- Have no block-scoping
- Tolerate re-declarations (no temporal dead zone)
- Are hoisted and can be declared below their use

```
var message = "Hello";
```

* You don't need to worry about all of this yet - we will come back to it soon

My (biased) advice?

- Only use `let` and `const`
- Use `let` for things that need to change, and `const` for things that don't need to change
- Only use letters in variable names (unless it really makes sense to add numbers, $ or _)
- Use camelCase for `let`
- Use UPPER_SNAKE_CASE for `const`

Loose vs. Strict Typing

Loose vs. Strict Typing

In strictly-typed programming languages when you create a variable, you must say what type it is too (and that can't ever change).

In loosely-typed programming languages, you don't have to explicitly say what type something is and the type can change.

Is JavaScript strictly typed?

typeof

typeof

Sometimes, you need to find out what type something is! `typeof` is often the solution. *

```
typeof("My String"); // "string"

typeof(42); // "number"

typeof(undefined); // "undefined"

let course = { name: "JavaScript Development" };

typeof(course); // "object"
```

* You will also need to use instanceofwith structural data types (e.g. arrays)

Type Conversion

GD!

Do the exercises here, please!

See you in 10 minutes!

Comparison Operators

What are comparison operators?

- Comparison operators compare two values
- They always return a boolean

==

Often called the "loose equality" operator, this will compare two values and return a boolean.

It'll try to make the types the same using type coercion

```
5 == 5; // true

"a" == "a"; // true

"First string" == "Second string"; // false

2 == "2"; // true
```

===

Often called the "strict equality" operator (or threequal), this will compare two values and return a boolean.

It cares about both the value and the type

```
26 === 26; // true

"String" === "String"; // true

"16" === 16; // false

2 === "2"; // false
```

My (biased) advice?

Always use the strict equality operator, and explicitly work with your types (make sure you know what types things are and convert them if necessary).

!=

!=

Often called the "loose inequality" operator, this will compare two values and return a boolean.

It will use type coercion, if necessary.

```
25 != 24; // true

"String" != "string"; // true

24 != 24; // false

19 != "19"; // false
```

!==

!==

Often called the "strict inequality" operator, this will compare two values and return a boolean.

It cares about both the value and the type

```
11 !== 13; // true

"Hello" !== "hello"; // true

38 !== "38"; // true


24 !== 24; // false
```

<

The less than operator, it'll always return a boolean.

```
4 < 10; // true

15.6 < 21.94; // true

25 < 25; // false

32 < 16; // false
```

<=

The less than or equals to operator, it'll always return a boolean.

```
11 <= 15; // true

13.1 <= 23.94; // true

25 <= 25; // true

31 <= 19; // false
```

>

The greater than operator, it'll always return a boolean.

```
24.124 > 12.522; // true

32 > 16; // true

15.6 > 21.94; // false

25 > 25; // false
```

>=

>=

The greater than or equals to operator, it'll always return a boolean.

```
25 >= 25; // true

31 >= 19; // true

11 >= 15; // false
```

13.1 >= 23.94; // false

Conditionals

The Typical Execution of a Script

```
let myVar = 10;  2
const IMPORTANT = true;  4
let myObject = {};  6
const data = takesTime();  8
const ALPHABET = [
    "a",
    "b",
    "c"
];

// End of program
```

JavaScript is an asynchronous programming language.

It runs line by line from the top, and starts the execution of each line in order (but won't wait for each line to complete).

What are conditionals?

- Conditionals make decisions with your code
  - Deciding what lines to execute and what lines to skip, based on the value of an expression
- They rely heavily on boolean(ish) values
- We use them to add logic, to make decisions, to allow data to change how our script runs and to make parts of our code optional

if

The if statement

The fundamental control statement that allows JavaScript to make decisions.

It is made up of the `if` keyword, parentheses wrapping a condition and curly brackets to identify what code should execute if the condition is truthy.

```
1  if ( true ) {
2      // Code to execute
3  }
```

The `if` statement

```
if ( 2 === 2 ) {
    // Maths seems to be working
}

let courseContent = "JS";

if ( courseContent === "JS" ) {
                // This is going to be fun!
}

if ( courseContent === "Ichthyology" ) {
    // This will be skipped!
}
```

if...else

The if else statement

```
if ( CONDITION ) {
    // Code to execute if the condition is truthy
} else {
    // Code to execute if the condition is falsey
}
```

The if else statement

```
if ( 2 === 2 ) {
    // Maths seems to be working
} else {
    // That's a real worry...
}

let number = 14;

if ( number > 0 ) {
    // The number is positive
} else {
    // The number is negative
}
```

if...else if...else

The `if else if else` statement

```
if ( FIRST_CONDITION ) {
    // Code to execute if the FIRST_CONDITION is truthy
} else if ( SECOND_CONDITION ) {
    // Code to execute if the SECOND_CONDITION is truthy
} else {
    // Code to execute if both conditions are falsey
}
```

- You can have as many else ifs as you need
- JavaScript will run the code in the only first passing condition's block (curly brackets). It will skip the rest

Logical Operators

What are Logical Operators?

- They are operators used to make more complex decisions or conditionals
- They can be ways to work with multiple pieces of data (or the opposite of a piece of data). For example:
  - This needs to be true AND that needs to be truthy
  - This needs to be true OR that needs to be truthy
  - NOT this (meaning this can't be true)
- You can have as many as you want of these in a row

||(OR)

The logical OR operator is represented by two vertical line symbols. As long as one item is truthy, it will pass.

```
true || true; // true

true || false; // true

false || true; // true

false || false; // false
```

|| (OR)

```
let lang = "JS";

if ( lang === "HTML" || lang === "CSS" || lang === "JS" ) {
    // You are talking about a front-end language
} else {
    // It is probably a back-end language
}
```

How does it work?

- It reads from left to right
- For each operand, it'll convert it to a boolean value.
  - If the result is true, it stops and will return that operand
- If it has gone through all the operands, and they were all falsy - it'll return the last operand

More or less, it'll return the first truthy value it finds, or the last value if nothing is truthy

&&

&&(AND)

The logical AND operator is represented by two ampersands. It will return the first truthy item it finds - meaning, both items need to be true for it to pass.

```
true && true; // true

true  &&  false;  //  false    false  &&

true; // false  false && false; // false
```

&&(AND)

```
let userLoggedIn = true;
let userIsAdmin = true;

if ( userLoggedIn && userIsAdmin ) {
// You have complete control
}

let validCreditCardDetails = true;
let balance = 100;
let itemCost = 80;

if ( validCreditCardDetails && balance >= itemCost ) {
    // Yes, you can purchase this item
 }
```

&&(AND)

```
let usedLoggedIn = true;
let userIsAdmin = true;

if ( userLoggedIn && userIsAdmin ) {
// You have complete control
 }

let validCreditCardDetails = true;
let balance = 100;
let itemCost = 80;

if ( validCreditCardDetails && balance >= itemCost ) {
 // Yes, you can purchase this item
 }
```

How does it work?

- It reads from left to right
- It converts every operand into a boolean
  - If that evaluates to false, it'll stop and return the value of that operand
- If all operands were truthy, it'll return the last operand

More or less, it returns the first falsey value it finds, or the last value if everything was truthy

!

!(NOT)

The logical NOT operator is represented by an exclamation point. It accepts a single argument, converts it to a boolean and then returns the opposite of that boolean.

```
!true; // false

!false; // true

!"Hello"; // false

!42; // false

!0; // true
```

!(NOT)

```
let hasAccount = false;

if ( !hasAccount ) {
    // You can create an account
} else {
// You already have an account
}

!!"Some string"; // What would this return?
```

SOLO!

Do the exercises here, please!

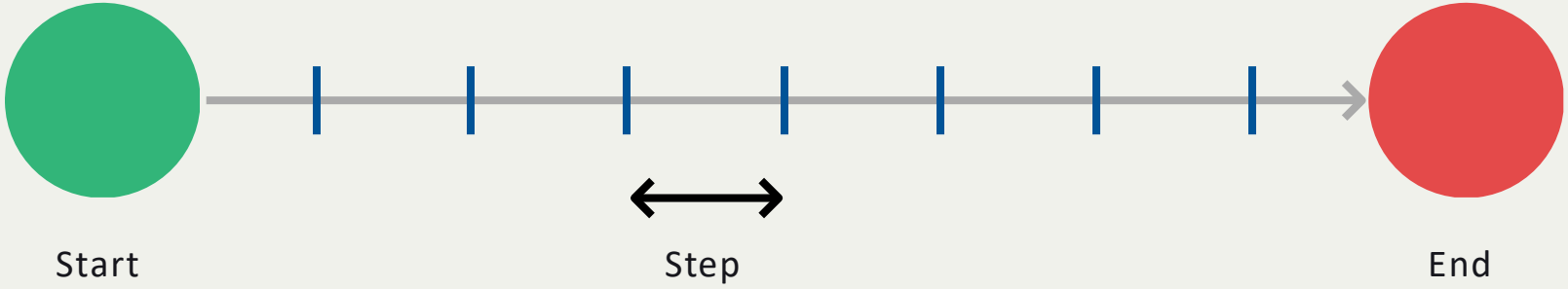
See you in 10 minutes!

Loops

What are loops?

- A block of code that can execute over and over again
- It consists of three must-have parts:
  - A starting point
  - An ending condition
  - An increment, or step
- You get access to the current value (e.g. how far it has gone in the loop)

Start, Step, End



Think of it as a running race. You start, you keep stepping until you reach 100 metres, and then you stop.

while

The while loop

```
1 let count = 0;                    // Start
2
3 while ( count < 10 ) {            // End
4       // Do something
5 count = count + 1;                // Step
6 }
```

The while loop

```
1 let count = 0;                    // Start
2
3 while ( count < 10 ) {            // End
4       // Do something
5 count = count + 1;                // Step
6 }
```

The while loop

```
1 let count = 0;                    // Start
2
3 while ( count < 10 ) {            // End
4       // Do something
5 count = count + 1;                // Step
6 }
```

for

The for loop

```
for ( let count = 1; count <= 10; count = count + 1 ) {
    // Do something
}

// Think of it as:
//
//        for ( start; end; step ) {                    }
```

My (biased) advice?

- Stick to using the for loop in the beginning:
  - It will ensure you have all the necessary parts (start, end, and step)
  - You can see all the important things in one place
- Double check all your loops before you run them!
  - Otherwise you run the risk of an infinite loop

SOLO!

Do the exercises here, please!

See you in 10 minutes!

That's all!

Homework

- Finish off in-class exercises
- Variables and Primitive Data Types homework
  - Note: It'll be worthwhile looking up interpolation and concatenation for this one!
- Conditionals homework
- Loops homework

A quick note on homework

- There is more than we expect you to get through
- As long as you have a serious go, that is okay!
  - We are here to answer questions too. Feel free to message us
- The biggest thing is that you are going to ***get out what you put in***

What's next?

- Variables
- Operators
- Conditionals
- Loops
- (Arrays)
- (Objects)
- (Functions)
- (Scope)

Thank you!