Programming fundamentals

Learning Objectives

- Know what a programming language is
- Understand what is a variable, the different datatypes in JS, and control flow
- Be able to run JS code using the terminal and VS Code

What Is a Programming Language?

Our task as programmers is to communicate thoughts/ideas/structures to a computer.

Imagine living in a world with no programming languages—you'd have to talk to your computer in machine code, like `0010101001010`. This would be very difficult to write and understand. Expressing high-level ideas would be nearly impossible.

Thankfully, humans have created programming languages and interpreters that allow us to speak to the computer in a more understandable and maintainable way.

The programmer can then write in high-level languages like JavaScript, Python, Java, Ruby, C, C++, C#, Swift, Go, Pascal, Scala, etc.

JavaScript

JavaScript is a versatile scripting language primarily known for web development but also used for server-side programming, game development, automation, and more. It was created by Brendan Eich in 1995 and has become one of the most popular languages today. It was specifically developed for web development, since before JS, there was no language that could be universally used in every browser.

JavaScript enables you to add interactive features and dynamic content to websites, and it's widely supported across browsers and platforms.

Working with JavaScript

Most developers write JavaScript code directly in web pages using `<script>` tags, or in separate `.js` files loaded into HTML. You can run JavaScript in browsers or with environments like Node.js. We will use Node.js that we already have installed for the time being.

To run JavaScript code from the terminal, you can use Node.js directly from the terminal:

```
node --version
```

Type node to start the interactive REPL (Read-Evaluate-Print Loop), and try running the following lines of code, then hit enter:

```
42 + 8
"hello".toUpperCase()
```

> If you are getting stuck in the Node REPL, remember, exit by pressing CTRL + D, multiple times if needed!

This environment is great for experimenting with JavaScript snippets outside larger programs.

Our first JavaScript Program

You can write your first JavaScript code in a .js file or directly in the console.

We will stick to writing files, as this enables us to debug and modify our code later.

First, get the `.run - Code Runner` extension from the extensions in VS Code!

Then, let's start with a classic, "Hello world"!

Create a folder in your desktop, naming it `couch_to_coder_2025`, then create a `day_1` folder in there.

In the `day_1` folder, create a file (not folder!) called `intro.js`.

In order to display any text in the terminal, we need to use the built-in method `console.log()`:

```
console.log("Hello world")
```

Don't forget the double qoutes!

Congratulations, we are officially JS developers now!

Variables

In JavaScript, variables are like labeled containers that hold different types of data, such as numbers, text, or more complex structures. They provide a way to store and manage information within your programs. Before you can use a variable, you need to declare it using keywords like `let`, `const`, or `var`.

> Technically you could create a variable without declaring it with the above keywords - please never do that, it will have a negative impact later on!

To retain text that we used or changed, we can store them in these variables. For now, always use `const` or `let` to declare a variable - `const` for values that we never want to change, and `let` if we want to modify them.

```
const firstName = "Zsolt"
const lastName = " Podoba-Szalai" // note the whitespace
const fullName = firstName + lastName
let greeting = "Hello"
greeting = "hi there!"
```

When it comes to naming your variables, you want something that's easy to remember and makes sense, right? So, the usual style is to use `camelCase` – start with a lowercase letter and then capitalize each new word, like `myCoolVariable`. Try to stick with letters, numbers, and underscores, and definitely start with a letter! While JavaScript might let you get away with some crazy names, using these tips will make your code way easier for others (and future you!) to understand when they're reading it. Naming things is half the battle when it comes to writing good code!

Now that we used our first data type - strings - in JS, let's explore the rest! The following data types are available for us:

- String: Represents text (e.g., "Hello, world!").
- Number: Represents numeric values (e.g., 42, 3.14).
- Boolean: Represents true or false values (e.g., true, false).
- Undefined: Represents a variable that has been declared but not assigned a value.
- Null: Represents the intentional absence of any object value.
- Object: Represents a collection of key-value pairs (e.g., { name: "John", age: 30 }).
- Array: Represents an ordered list of values (e.g., [1, 2, 3, "apple"]).

We will worry about the last two in the next lesson.

In JavaScript, the Number data type represents both integer and floating-point numbers. Unlike some other languages, JavaScript doesn't have separate types for integers and decimals; all numeric values are stored as double-precision floating-point numbers. This means you can perform arithmetic operations, store whole numbers, and represent fractional values all within the Number data type. JavaScript also provides special numeric values like Infinity, -Infinity, and NaN (Not-a-Number) to handle edge cases and errors that can arise during numerical operations.

Try out the following:

```
console.log(3 + 4);
console.log(9 - 64);
console.log(5 * 4);
console.log(6 / 0);
```

Yes, you can even divide with 0 with JavaScript...

The `Boolean` data type in JavaScript represents a logical value, which can be either `true` or `false`. Booleans are fundamental for making decisions in your code, controlling program flow with conditional statements (if...else), and evaluating logical expressions. They are often the result of comparisons (e.g., `5 > 3` evaluates to `true`) or logical operations (&& (AND) || (OR)). Understanding how to use booleans is essential for creating programs that can respond to different conditions and make decisions based on data.

We will take a look at these soon!

A note about `null` and `undefined`

In JavaScript, both null and undefined represent the absence of a value, but they signify different things. undefined typically means that a variable has been declared but has not yet been assigned a value. It's JavaScript's default state for uninitialized variables. On the other hand, null is an assignment value. It means a variable has been explicitly assigned the value of null, representing the intentional absence of a value. Essentially, undefined is JavaScript saying "no value has been given yet," while null is the programmer saying "there is intentionally no value here."

```
let firstName;
console.log(firstName);
```

Conditionals & Control Flow

Out of these datatypes, probably booleans are the ones that are the least obvious when it comes to how to use them. But if we imagine ourselves when we are making decisions, or asking ourselves strictly yes-no questions, then we find ourselves in a situsation where we are able to use them.

Every day we make decisions which effect what we do. If we didn't then we'd do exactly the same thing every day, and in every situation, and life would get really repetitive.

Often, these decisions we make in our daily lives are made based on a 'yes' or 'no' question. When we think about these questions and come to a yes / no answer we are evaluating them. We take in a statement about the world and then give a yes / no answer. We then make a decision as to what to do based on that answer.

Computer programs work in exactly the same way. We can give them a "statement" just like what we have been doing, and it will "evaluate" it to true or false (yes or no). We can also make sure that certain parameters are set before running our code, like confirming that our user is logged in or not, or that our database is connected or not. Our programs can then take different paths depending on the condition - this is why such statements are called *conditional statements*. If we could not do this, then all our programs would behave in exactly the same way, every time we run them.

We can try to convert question statements to boolean expressions. (true or false statements) Open-ended questions cannot be converted to booleans, we need to be making a yes or no decision.

Checking for Equality

One form of conditional statement in programming is to check if two values are the same. If they are the same, the result is true, otherwise it is false.

In JavaScript we use a special operator to check if things are equal. We cannot use the `=` operator for this - if you remember, `=` on its own *assigns* a variable. Three `===` together checks if two things are equal.

There is also the possibility of using `==` two equal signs, like in most other programming languages, but this one can be unpredictable. Stick to three for now!

Here are some examples of boolean expressions:

```
console.log(4 === 2 + 2) // true

console.log(4 === 2 + 3) // false

console.log("apple" === "apple") // true

console.log("apple" === "orange") // false
```

Conditional Operators

The equality or `===` operator belongs to a group of operators which are used for checking conditions. These are collectively known as conditional operators.

`>` - Is Greater Than

We use this operator to evaluate if the object on the left is greater than the object on the right.

```
5 > 4 // true
9 > 100 // false
```

`<` - Is Less Than

We use this operator to evaluate if the object on the left is less than the object on the right.

```
99 < 100000 // true

100 < 50 // false
```

`>==` and `<==`

These two operators, "greater than or equal to" (`>==`) and "less than or equal to" (`<==`) are variants of the greater than and less than operators, which also evaluate to `true` if the values are equal, unlike the greater than and less than operators themselves.

```
100 < 100 // false

100 <== 100 // true

99 <== 100 // true
```

A note on pseudocode

We might not want to always jump into solving problems - thinking it throught first always results in better outcomes! For this, we could use a technique called pseudocoding.

Pseudocode is like writing out your plan for a program in plain English (or whatever language you prefer!) before you actually start coding. It's a way to think through the logic of your program step-by-step, without worrying about the specific syntax of a programming language. It helps you organize your thoughts and identify potential problems before you get bogged down in the details of writing real code. Think of it as a rough draft for your program.

For example, I might want to write code that checks if a number is even or odd. Before frantically starting typing I might collect my thought process like so (you can write this as comments in your javascriot

```
// Program: Check if a number is even or odd

// Input: A number

// Steps:
// 1. Get the number from the user.
// 2. Calculate the remainder when the number is divided by 2.
// 3. If the remainder is 0, then the number is even.
// 4. Otherwise, the number is odd.
// 5. Display the result to the user.

// Output: "The number is even." or "The number is odd."
```

Not mandatory to use for the simplest of cases - but it will serve you well for more complex applications.

Remember, solve one step at a time!

Control Flow

Similarly, we might make decisions based on whether something is true or false.

If my car is running on fumes, I fill up my tank.

If I'm hungry, I'll find myself something to eat.

Similarly, our app might decide to let a user see someone else's posta only if they are friends. Or if they are logged in

You might see a pattern here - we are using the `if` word quite a lot here!

`if` Statement

Let's see - what happens if (ha!) we want to check if our user is over 18? We could use it to check if a registered user can enroll in a university without parental consent.

```
const age = 18

if(age >== 18) {
    console.log("Allowed to enroll")
}
```

Try running it, then change the number to something lower than 18 and try again!

The `else` Clause

So our program now does something if our condition is true, but we might want to do something different if our condition is false. It's like saying:

```
IF the condition is true THEN do X
OTHERWISE do Y
```

In our case, there is no feedback for when a user is not over the age limit. Let's change that with the `else` keyword!

```
const age = 18

if(age >== 18) {
    console.log("Allowed to enroll")
} else {
    console.log("Need parental consent")
}
```

You can even chain together multiple else statements if you more than one condition!

```
const age = 18

if(age >== 18) {
    console.log("Allowed to enroll")
} else if(age <== 10 {
    console.log("Need to wait a few more years, sorry")
} else {
    console.log("Need parental consent")
}
```

Checking Multiple Conditions

Sometimes we want to check if one or more conditons are true. To do this we can use **logical** operators.

We have the logical AND `&&` and the logical OR `||` operators.

The `&&` operator needs a boolean `true` expression on both sides (remember ,you can use brackets to group up conditions if needed)

```
const age = 18;
const haveParentalConsent = true;

if(age >== 18){
    console.log("Allowed to enroll")
} else if (age <== 18 && haveParentalConsent){
    console.log("Allowed to enroll with parental consent")
}
```

The logical OR operator needs at least one true value on either side of the operator

```
const age = 18;
const haveParentalConsent = true;

if(age >== 18 || haveParentalConsent){
    console.log("Can enroll")
}
```

> Note that data types are important - just because the number 23 and the string "23" have, at least for the human eye, the same numeric value, JavaScript will not interpret it that way - one is a number, one is a string. This is where using 3 === operators is important, because 2 might make it pass
>
> ```
> console.log(23 == "23") //true
> console.log(23 === "23") //false
> ```