Objectives

- You should be able to create and use a variable
- You should be able to use basic JavaScript operators
- You should be able to add JavaScript into a HTML page

Adding JavaScript to a html file

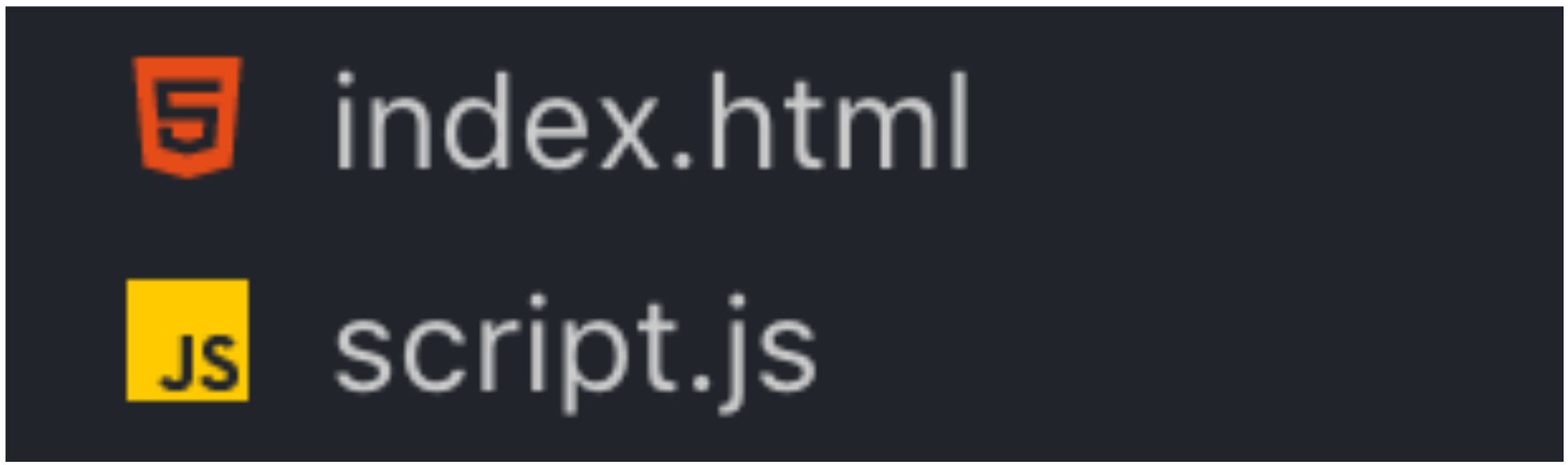
Way 1 →

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Lesson one</title>
</head>
<body>

<script type="application/javascript">
  console.log('First lesson');
</script>
</body>
</html>
```

Way 2 →

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Lesson one</title>
</head>
<body>

<script src="./script.js"></script>
</body>
</html>
```

Current file directory



How to add comments to JavaScript code?

```
5     // Comment only on one line
6
7     /*
8     |  Multi-line comment
9     */
10
```

Creating variables

```
1    var name; // is automatically assigned undefined
```

Variables are used to store values/function definitions which can be reused. For example, store a value to use for division.

When naming variables use camel case. E.g. firstName

But why is this useful?

Change in one place, it will change wherever it is used.

What can you start a variable name with?

- **A letter. E.g. ‘F’**
- **Dollar sign. E.g. ‘$’**
- **Underscore. E.g. ‘_’**

What can’t you start a variable name with?

Storing a value into a variable

```
1    var name; // is automatically assigned undefined
2
3    name = 'Francisco';
4
```

What can be stored in a variable

JavaScript is a dynamically typed language. This means data types are automatically assigned to variables based on the value stored.

There are 7 data types:

- **Number**
- **String**
- **Boolean**
- **Undefined**
- **Null**
- **Symbol**
- **Object**

How to join two or more data types together

```
1     var name = 'Francisco';
2     var age = 20;
3
4     console.log(name + ' is ' + age);
5
6
```

Note: In strings, you can escape character using ‘\’

How is it possible to compare/ join different data types

This is possible due to something called type coercion.

JavaScript can convert one data type to another for the required purpose. This is done behind the scene, or can be done explicitly.

Common operators which do type coercion are:

- **+**
- **==**

Reassign a new value to a variable

```
var name = 'Francisco';
var age = 20;

age = 25; // Can assign same data type

age = '25' // can assign a different data type

```

Operator precedence

Every operator in a Javascript has a precedence. This means that each operator has a default order in which it is evaluated.

On the following website you can find a table showing the precedence.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

Basic operators

- **Minus (-)**
- **Plus (+)**
- **Multiply (*)**
- **Divide (/)**

Logical operators

- **AND (&&)**
- **OR (||)**
- **Less than (<)**
- **More than (>)**

Check the data type of a variable

```
1    var name = 'Francisco';
2    var age = 20;
3
4    age = 25; // Can assign same data type
5
6    age = '25' // can assign a different data type
7
8    console.log(typeof age);
```

Coding challenge

Mark and John are trying to compare their BMI (Body Mass Index), which is calculated using the formula: BMI = mass / height^2 = mass / (height * height). (mass in kg and height in meter).

1. Store Mark's and John's mass and height in variables
2. Calculate both their BMIs
3. Create a boolean variable containing information about whether Mark has a higher BMI than John.
4. Print a string to the console containing the variable from step 3. (Something like "Is Mark's BMI higher than John's? true").

GOOD LUCK 😀

Extension

- **Use a math operator to multiply the two heights**
- **If you have used the escape character ‘\’. Try outputting without using the character, and vice versa**
- **Try outputting the final string using template literal**