Intro to Javascript

Web Design 2 - Week 1

How does Javascript work?

- JS is a scripting language most often used for client-side development
- It's an implementation of ECMAscript standard
- Browser engines are responsible to compile and run JS in browsers
- Google's V8 is the most popular JS engine and supports Chrome, Edge, Brave

Setting up VSCode

Download from code.visualstudio.com

Extensions

- Prettier (to format code)
- ESLint (to analyze, find and fix syntax errors in the code)
- Code Spell Checker

- We can also write Javascript in the browser console
- On Firefox and Chrome - Menu -> More Tools -> Web Developer Tools

Also recommended to use Firefox or Chrome for development.

ES6

ECMAScript 2015 (also known as ES6) is one of the biggest changes to the language recently.

Why do we care?

- New features
- Updated syntax
- Makes our lives easier

Variables

- `const` for values constant values
- `let` for values that may change

`const` vs `let`

Now what's `var`?

Moving on to Types

Primitive Types

- Numbers
- Strings
- Boolean
- Null
- Undefined
- (and `BigInt` and `Symbol`)

References Types

- Objects
- Arrays
- Functions

Types Examples

`x = 100` Number

`y = 20.56` Number

`name = "Jason Bourne"` String

`isPresent = true` Boolean

`inputData = null` null

`upcomingHolidays = undefined` undefined

String literals

Using variables inside a string

```
let x = 100;
console.log(`the value of x is ${x}`); //
prints "the value of x is 100"
```

JS Functions

- Functions help in structuring and maintaining clean code
- Writing reusable code

```
function myFunctionName() {
  let x = some_calculation;
  return x;
}
```

Example Function

```
function addTwoNumbers(x, y) {
  const answer = x + y;
  return answer;
}
```

Arrow Functions (ES6 syntax for functions)

```
const myFunctionName = () => {
  let x = some_calculation;
  return x;
};
```

Adding math and complexity

If `something is greater than 10`, only then return `someValue`

Conditions

`if...`

`if...else...`

`if...else if...else`

Example

```
// Check if a number is even
if (x % 2 == 0) {
    console.log("EVEN NUMBER");
else {
    console.log("ODD NUMBER");
}
```

Loops

- Loops offer a quick way to do a task repeatedly.
- As we start working with more complex data, loops also help perform iterable tasks with ease.

For loop

- Do something `n` number of times

  `for` loop structure:

  ```
  initialiser; condition; logic;
  ```

```
// print values from 0 to 19;
for (let i = 0; i < 20; i++) {
  console.log(i);
}
```

While loop

– Do something `while` something is true (or false)

```
while (x > 20) {
   console.log(x);
   x++; // incrementing the value
}
```

Reference Types

Arrays

- Arrays are a list of values (can be numbers, strings, or any primitive type value)

```
let myArray = [1, 20, "cats", "plants", false]
```

  (But it's generally a good idea to stick to one data type for 1D arrays for now! We'll explore more nested data in the coming weeks)

- How to find out how many items an array has though?
- Easy! `myArray.length`

Objects

– Objects are entities with names (`keys`) and values (`values`)

```
let cat = {
   name: "jorts",
   age: 2,
   color: "orange",
};
```

Nested Arrays and Objects

- Both objects and arrays can be nested
- An array of arrays, an object inside an object, an array of objects- any combination of data types can be nested

```
let today = {
  date: {
day: 26,
month: "January",
year: 2023,
  },
  weather: ["cloudy", "rainy", "light snow"],
  location: "London",
};
```

Looping through Arrays

```
let myPets = ["cat", "dog", "snake",
"mouse"];

for (let I = 0; I < myPets.length - 1; I++)
{
  console.log(myPets[I]);
}
```

New ES6 Syntax

```
myPets.forEach((pet) => {
    console.log(pet);
}
```

Looping through Objects

```
const object = { a: 1, b: 2, c: 3 };

for (const property in object) {
  console.log(`${property}:
${object[property]}`);
}
```

Variable Scopes

Block scopes

Local scopes

Global scopes

Block scopes

Variables declared within one block cannot be used outside the block;

Example

```
// x cannot be used here
{
   let x = 10;
}
// or here
```

Local scopes

Variables declared within a function cannot be accessed outside of it

```
function myFunction() {
  let x = 10;
  return x * 2;
}
// x is not accessible here
```

Global scopes

Variables declared outside blocks/functions become global

```
let x = 100;

function myFunction() {
  console.log(x); // prints 100
}
```

Additional

- Boolean coercion
- Javascript can coerce a boolean result out of most values
  - `0` false
  - `1` true
  - `null` false
  - `"cat"` true
  - `undefined` false

Additional

`==` vs `===`

- In JavaScript `==` and `===` can work differently.
- `===` is strict equality [^1]
- `0 == false` true
- `0 === false` false

[^1] More on MDN Documentation

JS Referencing & Documentation

- javascript.info `// comprehensive js course with exercises`
- MDN Web Docs `// js documentation`
- W3Schools `// js documentation`

Tasks

1. Try code exercises at - https://www.w3schools.com/js/exercise_js.asp
2. Setup Code Editor - VSCode with extensions
3. Create a `fileName.js` file in your code editor and link it with your `index.html` file using `<script src="fileName.js" />`

– Define a function that takes two integers, and a maths operation as an input, and returns the result as a string Eg. `myFunction(8, 5, “+”) = "The result is 13"`

– Define a function that takes as an array of integers as input and returns a new array with the squared value of each integer (using both `for` loop and `.forEach` method)