HTML, CSS, & JS Basics

Contents

- VS Code
- Chrome
- Demonstration of the basic app
- Basics of JS:
  - numbers and integers
  - strings
- Basic app in HTML and CSS
  - basics
  - emphasis on flexbox

Note that where you see a `>`, that is the symbol that indicates that I'm referring to a command in Terminal (which quite often has a `>` by default). When you see `<something>` it means replace this part with the appropriate argument. So:

`> cd <directory-name>` means that you will need to type:

`> cd coding-directory` or..

`> cd some-directory-you-have`

A few other thoughts on coding

We are here to learn code, but we are also here to teach you how to think like coders, which means becoming self-sufficient. Of course you will need guidance (even the best coders do), but we are also trying to get you to a point where you can learn on your own, and then come to help when you get stuck.This is hard, but hard work will pay off. It's important to understand that while this is a difficult set of skills to master, it rewards work. Intelligence is useful but overrated in coding. Grit and persistence are key aspects too.

Yesterday was hard because there was a lot to get through getting things set up, but things will be a little easier to follow from here on (not quite so many shiny new moving parts). Please feel free to ask questions. I will respond when I can.

VSCode

- This is a text editor, and the tool we will use to write our code.
- Open with `>` code .
- It is just a tool to help us write programs
- Often it can help by indenting things for us, and colouring things, both of which make the code easier to read.

Terminal

- `> ls` : list all the files and directories in the current directory
- `> mkdir <dir-name>`: make a new directory
- `> cd <dir-name>`: change location into this new directory
- `> touch <file-name>`: create a new file called file-name.
- `> New-Item <file-name>`: (Windows) create a new file called file-name.
- `> cd ..`: go back up the filesystem one level.
- `> rmdir <dir-name>`: remove a directory (directory much be empty).
- `> rm <file-name>`: remove a file (cannot be undone).

JavaScript

JS is a programming language that a browser knows how to understand. We will use this later to make our pages interactive. For now we are learning the basics, and will have it impact our pages in the future.

Numbers

- Integers (whole numbers, or counting numbers).
- Floats (decimals, or floating point numbers).
- Operations (including modulo %, which is the remainder).

Strings

- A string is the name for text in JS.
- Interpolation: where we use a variable as a stand in within a string.
- eg, ``` `Hello there, ${student1}` ```
- Notice the backticks (`` ` ``: far top left of your keyboard)
- We are assuming that we put a value into the variable `student1`

Variables

- Think of a variable as a named storage space for some data.
- `let <variable-name>`: used to define a local variable.
- We have learned how to store integers, floats, and strings in variables.
- We have briefly learned how to manipulate variables.
- We put all these skills together in our first JavaScript file, index.js.
- We connected it to our page with a script tag in the HTML.

HTML

- The browser understands how to read and interpret the structure of HTML
- HTML forms a tree structure, with a hierarchy of tags. Tags on the same level are siblings. Tags within tags are the children, and the enclosing tags are the parents.
- We connected our HTML to the browser by clicking ‘Go Live’ on VSCode, or by getting the path of our index.html file, and putting that into the browser.
- To get the HTML boilerplate we had an empty file, typed `!` and then pressed tab.
- The visible HTML goes within the `<body></body>` tags.

- We learned about heading tags (`<h1>`, etc), unordered list (`<ul>`) and our list elements (`<li>`).
- We connect the JS to the webpage using a `<script> tag and providing a path to the .js file (<script src="index.js" defer></script>`)

Chrome

- In Chrome we learned how to see behind the scenes and use the Chrome tools. To access these features we right-click on the browser and select `inspect`.
- The 'Elements' tab shows us the HTML. Feel free to have a look at the HTML of your favourite websites. You can also see here the tree structure of the HTML.
- The 'Console' tab shows the JS that is running on the page.
- Here in the console you can also write javascript directly.

CSS

- CSS (Cascading Style Sheets) is how we add colour and style to the page.
- We put our CSS in a file called styles.css (although it could be called anything - this is just convention), and we connected it to the page with a `<link>` tag (`<link rel="stylesheet" href="styles.css">`)
- We learned (briefly) how to change the background colour, and the text colour (which is the color attribute).
- In our case we changed the styles of the body, and the `h1`s on the page.

Booleans

- Yesterday we discussed a new datatype, booleans, to add to numbers and strings.
- Booleans take only two values: `true`, and `false`.
- Please note that these are different to the strings `'true'` and `'false'`. They are the words used for the concepts of true and false.
- This is linked to the idea of comparison operators, and we looked at:
  - `>` (greater than)
  - `<` (less than)
  - `>=` (greater than or equal to)
  - `<=` (less than or equal to)
  - `==` (equality), and
  - `!=` (not equal).
- We learned how to use these operators when comparing numbers and strings, and then also to do the same with variables (that contain numbers and strings, or even booleans).

Control Flow

- Here we started looking at how to change the functioning of our code depending on the value of a variable, or the changing of the input.
- To do this, we use an if statement.
- An if statement requires a condition.
- The condition will resolve to a boolean (`true` or `false`).
- `if` statements can exist on their own, or with `else if (condition)`, or with an `else` statement.
- The `else if` statements (if they are required) need to have a condition attached.

- The `else` statement does *not* have a condition. It is the catch-all that scoops up all conditions that are not caught earlier.

More JS

Arrays

- We have been thinking of variables as named storage spaces for data.
- Arrays are like having a whole warehouse of storage, or think of them as a series of storage buckets.
- They come pre-indexed, starting at 0, and then 1, then 2, and so on.
- We can put just a few things in, or many many things.
- We can create an array like this:

`const myPets = ["Garfield", "Odie", "Snoopy", "Woodstock", "Bugs", "Winnie"]`

- We can access the contents of the arrays we create by referencing the name of the array, and then the index, eg, `myPets[2]` would access the fourth element of the array that we called `myPets`, and that would be the string `"Snoopy"`.
- Once we have made our array, we have various tools at our disposal.
- We showed that we can search through our array using a `while` loop (and later we will show even simpler ways to loop through our array).
- We didn't discuss these much, but we can pop the last element off the end of the array: `let lastElement = myPets.pop()`.
- We can also add an element to the array: `myPets.push("Gromit")`
- Run the following code to see these in action:

```
const myPets = ["Garfield", "Odie", "Snoopy", "Woodstock", "Bugs",
"Winnie"]
pet = pets.pop()
console.log(pet)
pets.push("Snoopy")
console.log(pets)
```

- I would recommend having a look at the Mozilla docs on JS docs (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) for some more interesting functions that apply to arrays.

Loops

While Loop

- A while loop can be thought of as a repeating `if` statement.
- It comes with a condition, which is checked each time the loop is to run. If the condition resolves to `true`, then the loop runs the code between the containing `{ }`.
- It takes the form:

```
while(<condition>) {
    // do this code..
}
```

- We need to be careful that we have a way to break the loop.
- We saw a few different ways of doing this:
    - With a changing counter that eventually makes the condition `false`.
    - With a `break` statement.
    - By checking some user input.

More CSS

Flexbox

- Here we looked at a tool that we can use to align some items in one direction.
- We can use it to centre one item, or distribute several items.
- We needed to remember the concept of the tree structure (or hierarchy in HTML), and remember that the flex box commands are added to the *parent*.

So, if we have:

```
<div class="wrapper">
    <div class="child">A</div>
    <div class="child">B</div>
    <div class="child">C</div>
</div>
```

- Then all of the flexbox instructions go on the `.wrapper` class.
- We can set the direction:
    - `flex-direction: row` or
    - `flex-direction: column`
- The `justify-content` options set the distribution of the items in the ***same*** direction that we set our `flex-direction`.
- And the `align-items` attribute controls the perpendicular axis - guides items in the other direction.
- An example could be:

```
.wrapper {
    display: flex;
    justify-content: space-around;
    align-items: center;
    height: 200px;
    background-color: brown;
}
.child {
    background-color: chartreuse;
}
```

- We looked at *Flexbox Froggy* and *Flexbox Zombies*, two games that can help to get your skills up at flexbox.
- If you want to challenge yourself this week, I would have a go at creating a website, and trying to use some of the skills that we have learned so far.
- This is a great CSS resource: https://css-tricks.com

More CSS

More Flexbox

- At the start of the class we showed the importance of understanding that the flex commands are written as attributes of the parent class, and they move the children around within that class.
- We used this to help is with entering a child within the parent (and also to show how we could move this child around as we wished within the parent).
- We preferred this approach to shifting the children around using `padding` or `margin`.

```
* {
    margin: 0;
}

h1 {
    background-color: green;
    margin: 50px;
    /* height: 300px; */
    /* width: 100%; */
    /* margin-top: 50px; */
    /* padding-top: 180px; */
    /* padding-left: 400px; */
}

.wrapper {
    background-color: blue;
    height: 300px;
    display: flex;
    justify-content: flex-end;
    align-items: flex-end;
}
```

- `margin` and `padding` are very important, but in general should be used for the finer details, whereas Flexbox is good at providing thew structure, and making our pages *responsive* (meaning that they hold their structure well across several screen sizes.
- I have left the margin and padding aspects in the code above (but they are commented out between the `/* */`
- We also learned about the `*` selector, that we can use to override the default margin and padding values for CSS elements. `h1`s and over tags come with some defaults build in, that we might need,

but it sometimes helps to start from scratch.

More JS

Objects

- We talked about arrays, and how they are very useful for storing data that is similar. Arrays are simply a tool that we can use, and we need to understand why we do so. And as our code gets more complicated, we need to make use of these structures.
- We leaned a new one, *objects*. These are used to store data that falls under the same topic, or that should be stored in the same place, but has datatypes that are dissimilar.
- Objects consists of key and value pairs. You access the value via the key of the object.
- Here are some examples:

```
console.log("index.js is runnning in week 4")

let typesOfSewing = ["Embroidery", "Hand-stitch", "Cross-stitch"]
console.log(typesOfSewing)

let sewingType1 = typesOfSewing[1]
console.log(sewingType1)

let arrayOfNumbers = [7, 6, 9, 2, 4]

arrayOfNumbers[0]
arrayOfNumbers[4]
arrayOfNumbers[3] = 100

// NOT a very useful structure
// let famousMusicians = [
//     "Freddy Mercury",
//     "Queen",
//     "Night at the Opera",
//     15,
//     "Bruce Dickenson",
//     "Elvis"
// ]

let freddy = {
    name: "Freddy Mercury",
    band: "Queen",
    numOfAlbums: 15,
    living: false,
}

let johnLennon = {
    name: "John Lennon",
    band: "The Beatles",
    numOfAlbums: 7,
    living: false
}
```

```
console.log(freddy)

console.log(freddy.band)
console.log(freddy.numOfAlbums)

console.log(johnLennon)

console.log(johnLennon.living)
console.log(johnLennon.name)
```

Functions

- Functions are repeatable segments of code. We use functions when we have some small piece of logic that we need to use a few times, or when we want to make our code clearer and more organised.
- There are a few ways to define functions (so be aware when you look online that sometimes it might look different), but the way we started with was using the `function` keyword, then the name of the function, followed by its arguments:

```
function funcName(arg1, arg2) {
    // function code in here
}
```

- we *call* the function (run the function) by invoking its name, and passing in the arguments that we require:

```
funcName(4, 5) // function call
```

- Arguments are like the *inputs* into the function.
- The return value is the data that is *returned* out of the function.
- If we need to store the output (return value) of the function, we need to place it into a variable:

```
let funcResult = funcName(4, 5)
```

- Remember that when we have something to the right of an `=`, that this evaluated, and then *assigned* to the variable. So when you have a function call on the right, that function will be run, and the return value then stored in the variable that you are declaring.
- Functions are a very important part of JS, so getting comfortable with the basics will be very important.

Some example code:

```
function adder(num1, num2) {
    let result = num1 + num2
    return result
}
// console.log(adder)

let resultOfAdder1 = adder(6, 7)
console.log(resultOfAdder1)

let resultOfAdder2 = adder(1, 4)
console.log(resultOfAdder2)

let resultOfAdder3 = adder(100, "ggggggg")
console.log(resultOfAdder3)

function printSomething(str) {
    console.log(`printSomething ran and had argument ${str}`)
}

printSomething("Banana")
printSomething("Orange")
printSomething()
```

CHALLENGES

1. In the console, calculate: ◦ How many hours are in a year. ◦ How many minutes are in a decade? ◦ How many seconds old are you?
2. What do you think happens when you combine the following floats and integers? Try computing these in the console: is the result a float or an integer? ◦ 3.0 / 2 ◦ 3 / 2.0 ◦ 4 ** 2.0 ◦ 4.1 % 2
3. Why is 4.1 % 2 => 0.099. Look up in the ruby docs or google modulo
4. Put "Hello world!" onto the screen
5. Make a new variable called 'name', set its value to your name (as a string).

Beast Mode

1. Use interpolation to put your name on the screen.
2. Make a new variable called 'siblings', set its value to the number of siblings you have (integer).
3. Use code to put your number of siblings on the screen.
4. Use interpolation to pretty it up, E.g.: "Total Siblings: 3".
5. Use code to increase your number of siblings by one.
6. Use interpolation again to put your new number of siblings on the screen.

Terminal Challenges

Using only terminal, navigate your file system to:

1. Create a file structure that matches your family tree e.g. a root folder with 2.grandparents, 2 parent directories in each with child folders within those.
2. Create a folder called students. Inside the folder, create a few student text files 'jane.txt', etc.

3. Rename one of the student files to include full names ('jane doe.txt') - note what happens if you try to use a space. How can you get around this?
4. Delete one student.

Extra Terminal Challenges

One: Man —

1. Google some interesting commands.
2. Use the `man` command to read about the command you have found.
3. Experiment with this command, and its flags.

Two: Help —

1. Repeat the above process, but use the —-help flag to get info on the command.

HTML Challenge

Create a website about your favourite band using the HTML skills you have just learned. Elements you should include on your site and page(s):

1. A homepage (index.html) featuring a photo of the band and a brief description of the band (eg. how it got started, what genre of music, etc...).
2. A discography page showing an ordered list of 3 studio albums (in order of release date) with a photo (album cover), title and track listing for each.
3. A bio page showing an unordered list of the band members with photo, name and instrument/role of each.
4. A nav bar with links to the 3 pages.
5. A heading on each page to indicate which page it is. Homepage heading should be the name of the band, with a sub-heading saying "Tribute Site" or similar.
6. Use emphasis on keywords
7. A footer at the bottom of each page declaring your copyright on such a wonderful write-up, and make sure you use the copyright symbol!
8. Your page should validate in this at this link: W3 Validator https://validator.w3.org/#validate_by_input.
9. Keep your code readabable and maintanable for your future self and future contributors.