Flexbox (CSS)

Today we're going to start by introducing you to flexbox. Developers use flex box in order to introduce a way of dynamically sizing their website. This makes it easier to create responsive designs, or to be able to create 1 website that can work on both a massive desktop monitor and a mobile screen. We use flexbox since it makes it easy to distribute space and align content on our site!

Next Slide

We'll need to define some syntax/words in order to work with flexbox easier. First we say that the entire flexbox item is within a container, and inside that container are some individual items.

VSCode

By using these terms we can begin to look at our first example of flexbox. If we open up `club1` we'll find an example file called `flex.html`, within it we've defined some simple HTML syntax and created a div with the class of *container* and some divs inside it with the class of *item*.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <link rel="stylesheet" href="style.css">
</head>

<body>
```

```
  <div class="container">
    <div class="item">Content!</div>
    <div class="item">Content!</div>
    <div class="item">Content!</div>
  </div>
</body>

</html>
```

Notice we've also defined some styling within the `style.css` file and imported them. The most important line of styling within this file is:

```
display: flex;
```

Which turns on the flexbox display type for that container.
If we open up this html file in our web browser we'll see our first example of flexbox. Notice how the items are placed beside eachother, with similar sizes. If we began to edit the content of one of the *items* to become much larger we can see the true benefits of flexbox:

```
<div class="item">Let's create some long form content and then
make it longer by copying and pasting it Let's create some long
form content and then make it longer by copying and pasting it
Let's create some long form content and then make it longer by
copying and pasting it Let's create some long form content and
then make it longer by copying and pasting it.</div>
```

What do you think will happen?

Notice how the flexbox automatically makes more space for the additional text. If we grab our browser window and make it larger or smaller, we will also see this adjust. This is where the true power of flexbox comes from.

flex-direction

We can define the flex-direction, or the direction that each *item* within the *container* is placed. The main options are:

1. row (default)
2. row-reverse
3. column
4. column-reverse
   So if we were to add a `flex-direction` item to our *container*, and provide it with a value of `column`, we'd see our items become vertical:

```
flex-direction: column;
```

flex-wrap

We can also allow our items to have the "wrap" ability. If you've ever used notepad this is similar to word wrap there. Essentially if one of the *items* within your container will fill up the entire width, it will place the other items on to a new line.

```
flex-wrap: wrap;
```

justify-content

We can define the `justify-content` item in ordeer to change how our flex items are spaced.

Show Slides

We can define the content as many values, some of the important ones are:

1. flex-end ⇒ `flex-end` won't immediately work due to that item with a massive amount of text, if we remove some of that text we'll see that the items are now placed at the end.
2. center ⇒ places the items in the center
3. space-between ⇒ items arae evenly distributed

```
justify-content: flex-end;
```

Let's see what each of these do to our current flexbox implementation.

align-items

We can think of `align-items` as a vertical version of `justify-content`. We can define them with some similar items, for example:

1. flex-start (default) ⇒ stretches to fit the container
2. center ⇒ items are centered vertically
3. strech ⇒ items are all streched to the same length

```
align-items: center;
```

How to Center a div!

We can finally answer the age old question of "how to center a div"! We can create a flexbox, and define the container to have:

```
justify-content: center;
align-items: center;
```

Yet we'll find with web development that this works *most* of the time, yet suddenly you'll have that one instance where it doesn't in which case more divs and more stylings is probably the solution!

Summary

You can do **so** much more with flexbox, but in the interest of not boring you we are going to stop here. Check out css-tricks, or flexboxfroggy if you would like to learn even more, or talk to one of the TAs. Flexbox, is well just like CSS in the way that we are going to be wanting to google how to do something with flexbox instead of memorizing them. Our goal with this flexbox section was to show you **why** it's so powerful, and why you as a developer should be using it as a solution to some of your problems.

Club Website

Now we're going to go back to our club website, and see an example of flexbox in design. If we open up `club2` and then `club.html`, we can then scroll down to our footer and notice that it is a flexbox!

```
<footer class="footer">
                <!-- Flex Container 1 -->
                <div class="footer-container">
                        <!-- Flex Container 2 (also Flex Container
1, Item 1 ) -->
                        <a id="footer-container-item"
href="https://instagram.com">Instagram</a>
                        <a id="footer-container-item"
href="https://linkedin.com">Linkedin</a>
                        <a id="footer-container-item"
href="email:contact@gmail.com">contact@gmail.com</a>
                        <!-- Flex Container 2, Item 1 -->
                        <a id="footer-container-item"
href="tel:123-456-7890">123-456-7890</a> <!-- Flex Container 2,
Item 2 -->
                </div>
                <p>This site was written by the members of the
Queen's People Watching Club.</p>
```

```
                <!-- Flex Container 1, Item 2 -->
                <p>copyright 2022&copy;</p> <!-- Flex Container 1,
Item 3 -->
        </footer>
```

We'll notice that the HTML is defined with `footer-container` and `footer-container-item` and we can check our CSS file to see that it is indeed using flexbox!

We have another example of flexbox on our site as well if you check out the human facts section on our `club.html` page.

Javascript

Switch to slides

Next we're going to have a look at Javascript - the leading and pretty much only programming language specifically for websites. It allows us to make a web page truly interactive and funnily enough isn't related to javascript at all.

Next slide

We will need to link our Javascript to our HTML file just as we had to link our CSS to our HTML page.

VSCode

Let's **create** a folder called `club3` and copy over our previous club pages. Then we'll create a new Javascript file called script.js, notice the `.js` file extension. We'll then place that script tag into our body tag in order to have the script execute.

From here we will actually write some javascript code. We'll create an **alert** by:

```
alert('hello this is an alert!')
```

We'll then open up our `club.html` page and we'll realize hey before our site actually loaded we have a little alert box. It seems that to create these alert boxes in our web browser, we use a javascript function called `alert`.

Variables

Switch to slides

Next we're going to go through some simple javascript syntax. If you've programmed in python before, you'll realize that javascript is quite similar.

VSCode

First we'll want to show how to define variables, there are three main ways:

```
var x = 42;
let y = 65.4;
const z = "69";
```

Javascript is similar to python in that it's a dynamically typed language - you do not need to provide the type of your variable before you define it. You may also think hey, what is the difference between these definitions. A `const` (or constant) is just that, it can only be defined once and can never be changed. Then the difference between `var` and `let` is based off of the *scope* of your code. Similar to python, `var` is

the equivilant of a global variable. If this is a bit confusing it will make some more sense once we do some conditional statements. For now let's just say that `var` and `let` allow you to change the value of your variable after the fact.

Conditional Statements

Switch to slides

We can use the basic math operators, and comparsion operators that are defined in Python in Javascript. We've got some new syntax here, first of all `===` in Javascript is similar to Python's double equals. Javascript technically has `==` and `===`, and the difference would be:

```
"1" == 1 // is true
"1" === 1 // is not true
```

Essentially, the triple equals sign means **exactly equal**, and it's the one you generally want to use.

Next slide

The main two conditional statements in Javascript are `if` and `switch`. These are very similar to Python. If you've never encountered a switch statement before, consider it as a bunch of if statements in a list that become a bit easier to write. For now, we'll focus mostly on if statements.

VSCode

First we'll write an example if statement.

```
if (x === y) {
        console.log("hello");
}
else {
        console.log("bye!");
}
```

We'll notice that `x === y` will evaluate to a boolean express of either true or false. From there if that value is true, we will enter "hello" into the console. We can think of the console as the terminal in python. If that statement is not true, we'll enter the else statement. Notice that if we open up our website, click through the alert screen, and then open up the console we'll see "bye" is in the console since x is not equal to y.

If you're getting confused at any point throughout this Javascript tutorial that's okay! We're aiming this as a overview of Javascript assuming that you know Python if that isn't you then please ask either myself or one of the TAs for a more indepth overview after the lesson, or there are a ton of online websites that will teach you Javascript from scratch!

Array

You can also define an array in Javascript. Define a variable like any other, and then encase your items in square brackets - just like python!

```
const fruit = ["apple", "banana", "orange"];
```

There are a **ton** of built-in Javascript functions to help you out when dealing with arrays. I highly suggest you check the documentation before you try to do anything with arrays. For some quick examples there are:

1. toString() ⇒ makes your array a string

2. join() ⇒ adds a string between each array item (good for making a list)
3. pop() ⇒ remove the last element and return that element
4. length ⇒ provides the length

Loops

Switch to slides

We can define a for loop as seen here, it's similar to a C loop and may look a little confusing if you've only done Python.

VSCode

We can also define a for loop as the following:

```
for (let step = 0; step < 5; step = step + 1) {
        console.log(step);
}
```

We define a variable step, we set it equal to 0. We check to see if step is less than 5. Then we complete the items in the loop (in this case printing out a value), and then we incremenet the value of step and start again.

For reference, this for loop would be the same as python's:

```
for i in range(5):
        print(i)
```

Functions

Javascript has a ton of ways for defining functions, for now we're not going to go too in depth with functions. For now, let's just show you a few ways of writing the same area function:

```
function area1(width, height) {
        return width * height;
}

const area2 = functiton areaOfSomething(width, height) {
        return width * height;
}

const area3 = (width, height) => {
        return width * height;
}
```

and we'll notice if we print out each of these values that these functions are the **exact** same.

We're also going to breifly mention that Javascript works with an object based system called JSON. We'll get into this so much more later when we use react (next week in fact), but for now let's just say it's very similar to a python dictionary or a key and value pair. You provide a key and that key links to a value, and then that value is provided back.

Why Javascript

Now your next question may be **why did we just learn all of this Javascript**, especially since it seems similar to Python, but let's actually show you how we can effect our HTML page by using Javascript.

We'll create a new folder called `club4`, and let's bring in our club web page from `club2`. Let's create a `script.js` file first, and then let's link it to our contact HTML page by using `<script defer src="script.js"></script>`. The `defer` value means it will wait to load the script until the

body of the site is loaded. This won't mean much until we actually define that script.

From here let's try to select some of the items from our HTML page. Remember earlier when we said hey you should define these ID values on your form so that you can select them later - well this is why!

We can get at these values by using `document.querySelector`. For example:

```
const form = document.querySelector('#contact-form')
```

Now that we've grabbed all of the elements of our form, we can start to check whether our form has proper input! Remember in the last session when we typed an incorrect email address, the CSS prompted us to enter a real email. Well, Javascript will let us do that on a more complicated level.

First, we need to find out when our form is actually submitted. We can do that by adding an event listener to our form.

```
form.addEventListener('submit', (event) => {}
```

We are grabbing our `form`, then adding an event listener by using the `addEventListener` function. Then we define what sort of event we want this to check for, in this case the submission of the form. Then our second argument is the function to call when this happens, which in this case is going to pass in the event that happens, and then do whatever is in the function!

From there we can actually get all of the values of our from from that submission! Remember when we defined the names in our HTML file for our form -- well we can use those names to get to the individual items in our form. We can use `event.target` in order to grab the object

that the form submitted (remember this is just a dictionary!). From there you can input the name of the element, for example `name` and then grab the value of that form input value.

```
const name = event.target.name.value
const email = event.target.email.value
const message = event.target.message.value
const experience = event.target.experience.value
```

Next, we want to display a message on our site somewhere when an error actually happens! First, we'll need to grab the error element in the same way that we grabbed the form

```
const errorElement = document.querySelector('#error')
```

If we check out our HTML page we can see that we have an empty div in order to put these errors in that element.

Then we create an array called `messages` in order to hold all of our errors. We will define a variety of checks:

```
if (name === '' || name == null) {
    messages.push('Name is required')
  }

  // This will check that the Email input isnt empty, and contains
an @ and .
  if (email === '' || email == null) {
    messages.push('Email is required')
  }

  if (email.includes('@') === false || email.includes('.') ===
false) {
    messages.push('Email is missing "." or "@"')
  }
```

```
  // This will check that the Message input isnt empty
  if (message === '' || message == null) {
    messages.push('Message is required')
  }

  // This will check that at leaast one radio input is checked
  if (experience === '') {
    messages.push('Experience rating required')
  }
```

From here, we have checked for all the errors that we can think of. We now need to actually provide our site with this error message. We can check the length of our `messages` variable in order to see if it has any errors within it: `if (messages.length > 0)`. From there we will prevent the default form action using `event.preventDefault()` ⇒ remember last lesson when we defined our form, by default when we submit it, it will refresh the page. Yet we want the user to be able to edit their input, so we want to disable that. From there, we can set some styling on the errorElement, and set its text to the messages joined together with a comma. `.join` will take each value from the array, and place a comma and a space between each of them.

```
if (messages.length > 0) {
    event.preventDefault()
    errorElement.style.display = 'inline'
    errorElement.innerText = messages.join(', ')
  }
```

We can see a general example of `.map` here:

```
let temp = ["fruits", "oranges"]
console.log("There are many fruits:", fruits.join(", "))
```

From here, when we submit our form, if there are any errors they show at the top of the page, and we are able to edit our form values in order to fix this!

Going Forward

This week was a bit different from the previous weeks in that we didn't spend all that much time working on our club site. We wanted to introduce you to flexbox, which is a great tool for you going forward, as well as Javascript which is the language that React is based off of.

I once again highly suggest you try to implement some of these things on a personal site - or at least consider using flexbox in your next project!