Week 08
JavaScript

- Is a modern programming language that is derived from the syntax of C;
- It has been around just about as long as Python, have been invented a few years later, in 1995. ♡
- JavaScript, HTML and CSS make up the three languages defining most of the web experience on the web.
- To start writing JavaScript, open up a file with the .js file extension.
- No need for any code delimiters like you may be familiar with if you've used a language like PHP. our website will know that our file is JavaScript because we'll explicitly tell it as much in an HTML tag.

.js

- Unlike Python which runs server-side, JavaScript applications run client-side, on your own machine.
- Including JavaScript in your HTML;
- Just like CSS with <style> tags, you can directly write your JavaScript between <script> tags;
- Just like CSS with <link> tags, you can write your JavaScript in separate files and link them in by using the src attribute of the <script> tag;

Variables

- JavaScript variables are SIMILAR to PYTHON variables. → tipada
  - ↳ No type specifier;
  - ↳ when a local variable is 1º declared, prefece with the ~~word~~ var keyword.

In python:

```
x = 44
```

We don't want to do that in javascript because that would actually create a global variable, if you were to put a semicolon at the end.

```
var x = 44;
```

that creates a local variable called x in JavaScript, and stores the value 44 in it.

Conditionals

- All of the old favorites from C are still available for you to use;

We can use that in JS too. ↓

- if
- else if
- else
- switch
- ?: (Boolean)

} it's in C also.

Loops

- All of the old favorites from C are still available for you to use.

- While
- do-while
- for

} it's in C too!

Functions

- All functions are introduced with the `function` keyword.
- JS functions, particularly those bound specifically to HTML elements, can be anonymous — you don't have to give them a name!!!

→ ok... but how can I call a function if it doesn't have a name?

→ BINDING THINGS TO HTML ELEMENTS

→ document object model —

Arrays

- JS, like C and like Python, are analogous to Python's list.

Declaring an array is pretty ← STRAIGHT/FORWARD ☺ simples/direto

```
var nums = [1, 2, 3, 4, 5];
```

↳ creates a local array

I can also mix types...

```
var mixed = [1,
             true,
             3.333,
             'fix'];
```

it's different than C.

Objects

- JS has the ability to behave in a few different ways, but in particular it can behave as an object-oriented programming language; *
- An object is sort of analogous to a C structure;

~ An object is really similar in spirit to a structure that we are familiar with, hopefully, with ©.

- In C, structures contain a number of fields or members, which we might also call properties.
- But the properties themselves cannot live stand on their own.

properties OR MEMBERS or FIELDS

If we define the structure for a car like this

```
struct car
{
    int year;
    char model[10];
};
```

} 2 of's fields or members

```
struct car herbie;
herbie.year = 1963;
herbie.model = "Beetle";
```

- We always have to associate these things with a particular struct car.
- Objects, meanwhile, have properties, but also methods, or functions that are inherent to the object, and mean nothing outside of it.

But we can't define a function inside of the curly braces of a struct! =in C...

parameter

```
function(object);
```

But isn't a object oriented!

- means the object is at the CORE of everything and instead, we're going to call this function that is a part of the object.

objeet.function()

- There are functions that are affiliated or associated with objects, and we call those functions as follows by specifying the object and saying we want some function to execute on that object.

- The fields and methods of an object are similar in spirit to the idea of a dictionary, with which we're familiar from Python.

Fields, methods ↔ dictionary ↔ Python

var herbie = {year: 1963, model: 'Beetle'};

in pairs.

Loops (redux)

- How do we iterate across all of the key-value pairs of an object (or indeed, all of the elements of an array)?

In Python, we saw something like this:

```
for key in list:
    // use key in here as a stand-in for list[i]
```

You can't do that in JS, but we can do something pretty similar for var key in object.

```
for (var key in object)
{
    // use object[key] in here
}
```

- We also have a slightly different variation for our key of object, which, instead of having to use object square bracket key, we can now refer to iterating across all of the values as opposed to this, which iterates across all the keys, this would iterate across the values:

```
for (var key of object)
{
    // use key in here
}
```

- We can cover both sides of a key value pair using FOR var key in OR for var key of ;

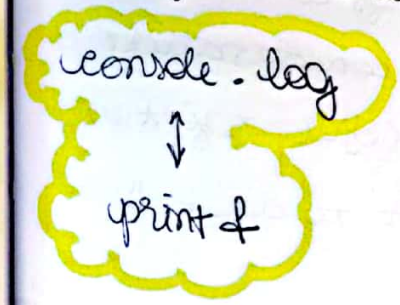
≠

- Example of an array where we're going to use these two different types of loop

```
var wkArray = [ 'Monday',      0
                'Tuesday',     1
                'Wednesday',   2
                'Thursday',    3
                'Friday',      4
                'Saturday',    5
                'Sunday' ];    6
```

6 elements in my array

And I want to use a FOR or day in week array. So I want to use a FOR in loop and I want to console dot log.




```
for (var day in wkArray)
{
    console.log(day);
}
```

- If I wanted to print out the days instead I would use a FOR of loop, and I would get the following printed out to the console - Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday.

- Printing and console interpolation

```
console.log(wkArray[day] + 'is day
    number' + (day+1) +
    'of the week!');
```

We do that using plus ⊕

↳ I want to concatenate strings together, but maybe here, I want to add

So if I want to console.log a bunch of different strings concatenated together, I just use ⊕ to do it. ☺

```
console.log(wkArray[day] + 'is day number'
    + (day + 1) + 'of the week!');
```

something that we don't expect to see

```
console.log(wkArray[day] + 'is day number'
    + (parseInt(day) + 1) + 'of the week!');
```

we need to use that.

Functions (redux)

- Arrays are a special case of an object (in fact, everything in JS is a special case of an object), and has numerous methods that can applied to them:

array.size() array.push(x)
array.pop() array.shift()

Quick intuition to use an anonymous function

- There is also a method for arrays called map(), which can be used to apply a function to all elements of an array. *

* map() → apply a function to all elements of an array!

```
var nums = [1, 2, 3, 4, 5];

nums = nums.map(function(num) {
    return num * 2;
});
```

→ x2

apply to all of the elements of the array.

(But why do on I not giving that function a name?)

I'm just doing this mapping here = I'm never going to need to double these numbers again.

And then, if I execute this line of code, what's going to happen to nums?

It's going to just DOUBLE every element of nums!

Events

- An event in HTML and JS is a response to user interaction with the web page.
- A user clicks a button, a page has finished loading, a user has hovered over a portion of the page, the user typed in an input field.
- JS has support for event handlers, which are callback functions that respond to HTML events.
- Many HTML elements have support for events as an attribute.

Example:

```
<html>
    <head>
        <title> Event Handler </title>
    </head>
    <body>                                  → event handled
        <button onclick="">Button 1</button>
        <button onclick="">Button 2</button>
    </body>
</html>
```

<button onclick = "alertName(event)">Button1</button>

The event is automatically generated by the page, and it basically contains (ML) of the information about what just happened.

And from that information, we can extract new information

Example:

```
function alertName(event)
{
    var trigger = event.srcElement;
    alert('You clicked on ' + trigger.innerHTML);
}
```

I can access that trigger's inner HTML to figure out which button was clicked. the inner HTML is what is between the button tags.