HTML, CSS, POSITIONING, EXTERNAL FILES, ARRAYS, AND INTRODUCTION TO FUNCTIONS

WEEK 2

Last week we introduced **arrays** via **data types**.

Of the **data types** we covered last week, an **array** is an **object**.

The purpose of an array?

We store collections of data in an array, which is great for enumerating or recording data.

Each item in an array is called an **element**, and each element has an **index**.

The **index** always starts at 0

```
var names = ['Jerry', 'George', 'Elaine', 'Kramer'];
```

We can target each **element** in the array via its **index**:

```
var first = names[0];
names;
```

What does **names[2]** return?

We can assign **values** to an array via the **index**

```
var names = ['Jerry', 'George', 'Elaine', 'Kramer'];
names[0] = 'Justin';
```

What will **names[0]** return? And what does the array contain now?

```
names;
=> ['Justin', 'George', 'Elaine', 'Kramer']
```

We can also find the number of **elements** in an **array** using the **length** property

```
var names = ['Jerry', 'George', 'Elaine', 'Kramer'];
names.length
=> 4
```

The **length** property will always give us a value one digit greater than the last **index**

To say this explicitly, **length** returns the number of items in the **array**, not the **index**

So the **index** of the last element is **length-1**

```
var names = ['Jerry', 'George', 'Elaine', 'Kramer'];
var lastIndexedItem = names.length-1;
lastIndexedItem;
=> 3
```

There are also **types** within **arrays**.

They can contain any type of **element** or **data** in JavaScript, and they can shrink or grow.

```
var sillyArray = [ 'Hello World!', true, undefined,
null, 42, ['Look!', 'a',
'nested Array!'], false ];
```

Side note, this code is *very* poor; you're just making your life difficult.

Keep different **data types** in separate **arrays**

Strings are similar to **arrays** in that we can find the **length** of them the same way we operate on **arrays**

```
var string = "Hello World!";
string.length;
=> 12

string[0];
=> H
```

We can also create arrays

```
var a = new Array();
a[0] = "dog";

a;
=> ["dog"]

var pets = new Array("dog", "cat", "unicorn");
pets;
=> ["dog", "cat", "unicorn"]
```

There are also a bunch of helper **methods**

The toString() method returns a string with each element separated by a comma:

```
array.toString();
```

The join() method returns a string with each element separated by a **parameter**:

```
array.join( param );
```

The pop() method returns the last item from the **array**:

```
array.pop();
```

The push() method adds one or more items to the end and returns the new **length**:

```
array.push( item1, item2, ..., itemN );
```

We can reverse the **array**:

```
array.reverse();
```

We can remove and return the first item:

```
array.shift();
```

We can add one or more **elements** to the front and return the new length:

```
array.unshift( item1, item2, ..., itemN );
```

An example, create an **array** and add **elements** to it using the **push method** in repl.it

```
var message = [];
message.push(1);

message.push( 'e', 'g', 'a', 's', 's' );
=> 6

message.push( 'e', 'm', 'T', 'E', 'R', 'C', 'E', 'S', 'X' );
=> 15
```

pop(), shift(), unshift()

```
message.pop();
=> 'X'

message.shift();
=> 1

message.unshift( 'duh' );
=> 14
```

Array reversal using reverse()

```
message.reverse();
[ 'S', 'E', 'C', 'R', 'E', 'T', 'm', 'e', 's', 's', 'a', 'g',
```

Turn that **array** into a string

```
message.join(' ');
'S E C R E T m e s s a g e duh'
```

Alright, let's talk about **iterating** and **loops**, then after we'll do a **for loop**

In a basic sense, loops execute blocks of code a set number of times.

An **infinite loop** is when we don't give the code a stopping point.

That will break your code, and I'm sure you'll all accidentally do it soon enough.

But, to reiterate (pun?), the **loop's** power is in the ability to run the same code over and over and over again.

```
var departments = ['Fine Art', 'Illustration', 'Cartooning'];

for ( var i = 0; i < departments.length; i++ ) {
        var department = departments[i];
  console.log( department );
}
```

JavaScript **arrays** have several **iterator methods**. Many of the methods require a **function** to be passed in as an **argument**

Each **element** in the **array** has the **statement** in the **function body** applied to it individually.

For example, the forEach() method is a cleaner approach to the previous code:

```
var departments = ['Fine Art', 'Illustration', 'Cartooning'];

departments.forEach( function( department ) {
        console.log( department );
});
```

In the previous example, **'department'** was just an **element**; it was arbitrary

And the **function** is called a callback

In brief, a **callback** is a **function** to execute for each **element**

The **callback** also takes three **arguments**, the element value, the element index, the array being traversed

So, this:

```
departments.forEach( function(department) {
  console.log(department);
});
```

And this:

```
function useThisLater(element, index, array) {
  console.log("element: " + element);
  console.log("index: " + index);
  console.log(" ");
}

departments.forEach( useThisLater );
```

Function the same way.

So, we just covered a lot of ground and remembering all of the particular **syntax** and names of these **methods** is exceedingly difficult to memorize--which is totally fine and normal.

Because of this, we constantly need to reference **documentation**.

If you recall, many websites for documenation are on the syllabus.

In any case, let's take roughly 10 or so minutes to skim over some documention on the Mozilla developer site.

Go here: https://developer.mozilla.org/en-US/docs/Web/JavaScript and track down the documention for:

- .every()
- .some()
- .filter()
- .map()

After you've looked over the **documentation**, open repl.it and create these **arrays**:

```
var evens = [];
evens.push( 2, 4, 6, 8, 10 );

var odds = [];
odds.push( 1, 3, 5, 7, 9 );
```

The **every() method** tests whether **ALL** elements in an array pass the test implemented by the provided **function**

```
var evenResult = evens.every(function(num){
        // console.log(num);
        return num % 2 === 0;
});
console.log("evenResult", evenResult);

var allDivisibleByFour = evens.every(function(num){
        return num % 4 === 0;
});
console.log("allDivisibleByFour", allDivisibleByFour);
```

The **some() method** tests whether **AN** element in the array passes the test implemented by the provided **function**

```
var someDivisibleByFour = evens.some(function(num){
        // console.log(num);
        return num % 4 === 0;
});
console.log("someDivisibleByFour", someDivisibleByFour);
```

The **filter() method** creates **a new array** with all elements that pass the test implented by the provided **function**

Note, this **method** does not mutate the original **array**

```
var bigNums = evens.filter(function(num){
        return num > 5;
});
console.log("bigNums", bigNums);

var smallNums = odds.filter(function(num){
        return num < 5;
});
console.log("smallNums", smallNums);
```

The **map() method** creates **a new array** with the results of calling a provided **function** on every element in the original **array**

```
var timesFive = evens.map(function(num){
        return num * 5;
});
console.log("timesFive", timesFive);

var timesTen = odds.map(function(num){
        return num * 5;
});
console.log("timesTen", timesTen);
```

Now, create a .js file and name it
LastName_FirstName_WEEK2.js

Using the same methods we looked at documentation for, do the following:

Evens operations:

- Use the **every() method** to check if **ALL elements** in the evens array are divisble by 2
- Use the **some() method** to check if **AN element** in the evens array are divisible by 4
- Use the **filter() method** to create a new **array** with all numbers in the evens array that are greater than 5

Odds operations:

- Use the **filter() method** to create a new **array** with all numbers in the odds array that are less than or equal to 5
- Use the **map() method** to create a new **array** by multiplying all of the **elements** in the odds array by 10

When you're finished, turn that file in to Canvas.