Intro To JavaScript – Day 3

Arrays

- Initialize an empty array: `let myArray = [];`
- Initialize an array with values: `let colors = ['red', 'green', 'blue'];`
- Access items of an array:

```
console.log(colors[0]);
console.log(colors[1]);
console.log(colors[2]);
```

- Size of an array: `console.log(colors.length);`
- Fastest way to empty/clear an array: `myArray.length = 0;`
- Add an item to the end of an array...
  - Option 1: `colors.push('purple');`
  - Option 2: `colors[colors.length] = 'yellow';`
- Arrays of arrays:

```
let points = [
    [0, 1],
    [1, 2],
    [2, 3]
];
console.log(points[0][0]);
console.log(points[0][1]);
```

- Arrays of objects:

```
let points = [
    {x: 0, y: 0},
    {x: 1, y: 1},
    {x: 2, y: 2}
];
console.log('x: ' + points[0].x + ', y: ' + points[0].y);
```

- Arrays of arrays AND objects:

```
let points = [
    [0, 1],
    {x: 1, y: 2}
];
console.log(points[0][0] + ', ' + points[0][1]);
console.log('x: ' + points[1].x + ', y: ' + points[1].y);
```

- Iterating through an array (do not use a for-in loop to iterate through an array! for-in is slower than counted iteration, and we usually care about performance.):

```
let colors = [ 'red', 'green', 'blue', 'purple', 'yellow' ];
for (let color = 0; color < colors.length; color++) {
    console.log(colors[color]);
}
```

- Removing an item from an array using `delete`:

```
let colors = [ 'red', 'green', 'blue', 'purple', 'yellow' ];
delete colors[3];
console.log(colors);
```

  Notice that it shows position 3 as 'undefined'. The value was deleted, but not the array element.

- Removing an item from an array using `slice` and `concat`:

```
let colors = [ 'red', 'green', 'blue', 'purple', 'yellow' ];
let newColors = colors.slice(0,3).concat(colors.slice(4));
console.log(newColors);
```

  Holy cow, this is a lot of work just to delete an item, there has to be a better way. There is.

  Also, notice that `slice` returns a new array, it doesn't modify the original array.

- Removing an item from an array using `splice`:

```
let colors = [ 'red', 'green', 'blue', 'purple', 'yellow' ];
colors.splice(3,1);
console.log(colors);
```

- How about one more little trick, applying a function to each element of an array:

  --- Using `.forEach` ---

```
let values = [1, 2, 3, 4, 5, 6];
values.forEach(function(value, index, array) {
    array[index] += 1;
});
console.log(values);
```

  --- Using `.map` ---

```
let values = [1, 2, 3, 4, 5, 6];
let newValues = values.map(function(value) {
    return value + 1;
});
console.log(newValues);
```

There are more array methods of interest, recommend looking into them:

- filter
- every
- some
- reduce
- indexOf
- lastIndexOf

Conditional Statements

```
let x = 1;
let y = 1;
if (x == y) {
    console.log('x is equal to y');
}
else {
    console.log('x is not equal to y');
}
```

Now, change `y = 1` to: `y = '1'`; Explain that `==` will use coercion, what you really want to do is use `===`

```
let x = 1;
let y = '1';
if (x === y) {
    console.log('x is equal to y');
}
else {
    console.log('x is not equal to y');
}
```

JavaScript has the standard relational and logical operators that you'd expect:

- < <=
- > >=
- != !==
- && || !

Other operators of interest

- `typeof` – Returns a string that represents the type of the object

```
let myDate = new Date();
console.log(typeof myDate);
```

- `instanceof` – Returns true/false if the object is of the type specified.

```
let myDate = new Date();
console.log(myDate instanceof Date);
console.log(myDate instanceof Object);
console.log(myDate instanceof Number);
```