Section 4 - Arrays and Loops

Learning Outcomes

Arrays and loops are an essential constructs in the majority of programing languages. The ability to define a list of values and iterate over the list performing operations is a critical skill to master.

- **Arrays** - introduction
  - syntax - 3 ways to declare arrays
  - attributes - `lenght` and uses
  - functions - `sort()` , `pop()` , and `push()`
- **Loops** - `for` and `while` loops

Resources

- Arrays - https://www.w3schools.com/js/js_arrays.asp
- Array Functions - https://www.w3schools.com/js/js_array_methods.asp
- Array Sorting - https://www.w3schools.com/js/js_array_sort.asp
- Convert String to Array - https://www.w3schools.com/js/js_string_methods.asp
- For Loop - https://www.w3schools.com/js/js_loop_for.asp
- While Loop - https://www.w3schools.com/js/js_loop_while.asp

Arrays

An array is a special variable, which can hold more than one value:

```
const cars = ["Saab","Volvo","BMW"];
const daysOfTheWeek = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];
const studentNames = ["Nash","Ridley","Ezra","Cora","Elsa","Mila"];
```

Why Use an Array?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

Arrays help maintain large sets of data under a single variable name to avoid confusion that can occur when using several variables. A value in the array can be accessed via it's index number.

Creating an Array

These are the three most common ways to create arrays for use in JavaScript programs.

1) Array Literal

Using an array literal is the easiest way to create a JavaScript Array. It is a common practice to declare arrays using square brackets `[` and `]`.

```
Syntax:
const array_name = [item1, item2, ...];
```

```
Example:
const daysOfTheWeek = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"];
```

> Note the use of the `const` keyword which is recommended when declaring arrays.

2) Using the JavaScript Keyword `new`

The following example also creates an Array, and assigns values to it:

```
const cars = new Array("Saab", "Volvo", "BMW");
```

3) Using `split()` with a String of values

A string can be converted to an array with the `split()` method:

```
Syntax:
string.split(separator, limit)
```

Parameter	Description
separator	*Optional*. The character (or regular expression) to use for splitting.
limit	*Optional*. An integer that limits the number of splits.

```
let input = "Saab,Volvo,BMW";
const cars = input.split(",");
```

`split()` can be applied to string literals

```
const cars = "Saab Volvo BMW".split(" ");
```

`split()` can also be applied to the result of other functions that return strings

```
const cars = prompt("Enter cars:", "Saab|Volvo|BMW").split("|");
```

Array Functions

`length`

Arrays have an attribute called length which is the number of elements in the array.

```
const cars = ["Saab", "Volvo", "BMW"];
let carCount = cars.length;    // carCount is 3
```

`sort()`

The `sort()` method sorts an array alphabetically:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
```

Array Indexes

Values in an array can be accessed by their index value. Indexes start with zero `0`.

Banana	Orange	Apple	Mango
0	1	2	3

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
```

In this example, the value of `fruits[0]` is `Banana` and `fruits[3]` is `Mango`.

> **Note**: To access the last element in an array by index use `length-1`.
>
> In the example above `fruits.length` is `4` and the index of the last element is `3`, so `fruits.length-1` can be used to access the last element by index.

`push()` and `pop()` Elements

When you work with arrays, it is easy to add and remove elements.

This is what popping and pushing is:

- `Pop()` - remove items from an array
- `Push()` - add items to an array

The `pop()` method removes the last element from an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

The `pop()` method returns the value that was "popped out":

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();  // The value of fruit is 'Mango'
```

The `push()` method adds a new element to an array (at the end):

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

The `push()` method returns the new array length:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");  // The value of length is 5
```

Loops

Loops can execute a block of code a number of times. Loops are handy, if you want to run the same code over and over again, each time with a different value.

`for` loop

The `for` loop has the following syntax:

```
Syntax:
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

- **"Initialization"** *(Statement 1)*: executed ***one time*** before the execution of the code block.
- **"Conditional check"** *(Statement 2)*: executed ***zero to many times*** defines the condition for executing the code block.
- **"Increment"** *(Statement 3)*: is executed ***every time*** after the code block has been executed.

```
Example:
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

- **Statement 1** sets a variable before the loop starts (`let i = 0`).
- **Statement 2** defines the condition for the loop to run (`i` must be less than 5).
- **Statement 3** increases a value (`i++`) each time the code block in the loop has been executed.

`while` loop

The `while` loop loops through a block of code as long as a specified condition is `true` .

```
Syntax:
while (condition) {
  // code block to be executed
}
```

In the following example, the code in the loop will run, over and over again, as long as a variable (`i`) is less than 10:

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

> **Note**: a `for` loop can be used when the number of elements are known and `while` loops is used when the number of elements or iterations is not known. In many cases `for` loops and `while` loops can be used interchangeably.