Revision: June

Programming Basics

Content

Loops
Advanced Array Methods
Callback Functions
Switch
Classes
Bind, Apply, Call

For Loop

```
1. let sum = 0;
                //initializer, test condition, updater
2. for (let i = 1; i <= 50; i++) {
3.   sum = sum + i;
4. }
5. console.log("Sum = " + sum);
```

While Loop

```
1. let sum = 0;
2. let number = 1;
3. while (number <= 50) {   // -- condition
4.   sum += number;          // -- body
5.   number++;               // -- updater
6. }
7. console.log("Sum = " + sum);   // => Sum = 1275
```

For Vs. While Loop

So when do we use for and when while? If the number of iterations is known, use the for-loop. If you want to loop until a certain condition is met, use the while-loop.

Do-While Loop

```
1. let sum = 0;
2. let number = 1;
3. do {
4.   sum += number;            // -- body
5.   number++;                 // -- updater
6. } while (number <= 50);     // -- condition
7. console.log("Sum = " + sum);     // => Sum = 1275
```

This is known as a *post-test loop* as the condition is evaluated after the block has executed.

While Vs. Do-While

The do-while loop is executed at least once whereas the while loop may not execute at all.

The do-while is typically used in a situation where the body of a loop contains a statement that generates a value that you want to use in your conditional expression, like this:

```
1. do {
2.    // read a character from keyboard in the body
3. } while (if ch === '0');      // => terminate loop if '0' is entered
```

forEach()

The forEach() method executes a function once for each element in an **array**.

```
1. let array1 = ['a', 'b', 'c'];

2. array1.forEach(function(element) {
3.   console.log(element);
4. });

5. // expected output: "a"
6. // expected output: "b"
7. // expected output: "c"
```

for ... in

The for ... in statement loops over properties of an **object**.

```
let string1 = "";
let object1 = {a: 1, b: 2, c: 3};

for (let property1 in object1) {
  string1 += object1[property1];
}

console.log(string1);
// expected output: "123"
```

Nested Loops

```
for (let i = 1; i <= 5; i++) {
    for (let j = 1; j <= 5; j++) {
        let result = i * j;
        console.log(`${i} X ${j} = ${result}`);
    }
}
```

Map

MDN Definition: The map() method calls a function on each element of an array and creates a new array with the results.

```
let arr = [1, 2, 3, 4];
const mappedArr = arr.map(x => x ** 2);
console.log(mappedArr);
// output: [ 1, 4, 9, 16 ]
```

Filter

MDN Definition: The **filter()** method creates a new array wihen all the elements meet the condition of the function.

```
const numbers = [1, 2, 3, 4, 324, 432, 32, 90, 80];
const result = numbers.filter(number => number > 10);
console.log(result);
// output: [ 324, 432, 32, 90, 80 ]
```

Reduce

MDN Definition: The **reduce()** method executes a function called on each element of the array, resulting in a single output value.

```
const numbers = [1, 2, 3, 4, 324, 432, 32, 90, 80];
const result = numbers.reduce((acc, cur) => acc + cur);
console.log(result);
// output: 968
```

Reduce

Accumulator value: It is the accumulated value previously returned in the last invocation of the function.

Current value: The current element being processed in the array.

Sort() Compare Function

compareFunction(a,b) returns 1
Then a comes before b (a is greater than b)
compareFunction(a,b) returns -1
Then b comes before a (a is less than b)
compareFunction(a,b) returns 0
Then the order of a and b remains unchanged (a and b are equal)

Callback Functions

```
function add(a, b , fn){
    return `The sum of ${a} and ${b} is ${a+b}. ${fn()}`;
    }
    // callFn function is called just
    function callFn(){
    return 'This is the callback function working.';
    }
    // calling add() function
    console.log(add(5,6,callFn));
// The sum of 5 and 6 is 11. This is the callback function working.
```

Switch

The **switch statement** evaluates an expression, and matches the expression value to the switch case. It the expression is a match, it executes statements associated with that case, as well as statements that follow the matching case.

Classes

Classes act as blueprints! They are ways of storing data related to a single thing.

Let’s say that “Human” is a class. And we need to store data relating to humans. All of that will be stored as properties.

A *method* is a series of instructions that we give a class to make it create, change or remove properties; or, to provide us with some data based on its properties.

Classes: Super & Extend

An advantageous feature of classes is that they can be **extended** into new object blueprints based off of the parent.

This prevents repetition of code for objects that are similar but need some additional or more specific features.

The *extends* keyword is used in *class declarations* to create a class as a **child** of another class.

bind(), call(), apply()

The bind() method creates a new function where “this” refers to the parameter in the parenthesis. This way the bind() method enables calling a function with a specified “this” value.

call() and apply() can run a function on an object.