LOOPS

We use loops to repeat some code either until a condition is met or for a certain amount of repetitions.

Previously we used the .map() method.
This uses a loop to access every element of an array.

We will look at two types of loops.
The ‘for’ loop and the ‘while’ loop.

for

Generally we use a for loop if we know how many repetitions we want to perform.

Syntax

The for loop has the following syntax

```
for ( i = 0 ; i < 10 ; i++ ){
     //code block
     }
```

Let’s break this down:
First we have the ‘for’ keyword.

```
for
```

Then we create 3 expressions.

```
(
i = 0;          First expression runs once before the loop.

i < 10;         Second expression is a condition.
                When this is false the loop will
                stop running.

i++;            Third expression will run as the final
                expression of every repetition.
)
```

After this we create the code block for the loop.

```
for ( i = 0 ; i < 10 ; i++)
{
// code block goes here
}
```

Everything inside the code block will be executed with every repetition.

Let’s look at an example:

```
for ( i = 0 ; i < 10 ; i++){
     console.log(i);
}
```

This will result in:

```
0
1
2
3
4
5
6
7
8
9
```

With arrays

It is common to use a for loop to go through an array. We can access the elements by their index.

```
const myArray = [ 'One' , 'two' , 'three' ]

for ( i = 0; i < myArray.length; i++ ){

        console.log( myArray[i] );

};
        // 'one'
        // 'two'
        // 'three'
```

This will log all the elements of the array.
Think about how the value of i increases from 0 - 2.

The code block of the loops can be thought of like so:

```
console.log ( myArray[0] );      //i = 0
console.log ( myArray[1] );      //i = 1
console.log ( myArray[2] );      //i = 2
```

To define the condition of the loop we used:
i < myArray.length

Remember .length will return the number of elements in the array.

```
console.log( myArray.length );   // 3
```

As such our for loop can be seen as:

```
for (i = 0; i < 3; i++){
//console.log
};
```

This should look familiar and is a loop that will run for 3 repetitions.
In our case it will run for however long our array is ensuring that we access all elements.

while

As opposed to the for loop we generally use a while loop if we don’t know how many repetitions we need.

Syntax

The while loop has the following syntax

```
while ( //condition ){

//code block

};
```

The while loop only takes 1 expression, unlike the for loop which takes 3.

This expression is the condition for the loop and it works the same as the 2nd expression of the for loop.

```
            this one
for ( i = 0; i < 10; i++ ){
//code block
};
```

At the beginning of every repetition the condition is checked.
If true, the code block will run and the loop will repeat.
if false, the code block will not run, the loop will end and we will continue with the rest of the code.

Let’s imagine we have a number guessing game. Users put their guess in an input box, we’ll assume it’s a website for ease.

Let’s look at an example.

```
const numberToGuess = 5;

const numberGuessed = prompt("Please pick a number");

if(numberToGuess == numberGuessed){

    alert("congratulations");
} else {
    alert("try again");
}
```

The above code is valid, but if users guess the wrong number they have to refresh the page or program to try again.

We want to keep asking until the users’ find the right number.
We don’t know how many tries it will take for the users to guess right so we can’t use a for loop.

This is where the while loop comes in handy. Let’s rewrite our program:

```
const numberToGuess = 5;

let numberGuessed = prompt("Please pick a number");

while(numberToGuess != numberGuessed){

    numberGuessed = prompt('Wrong number, try again');

};

alert('Congratulations');
```

Now our number game works as intended.

Note: You can try the number game by running the numberGame.html file in a browser.