Loops

Learning Objectives

- Be able to use loops to control behaviour
- Be able to use a for loop

What Are Loops

Loops in Python are used to execute a block of code a specified number of times. One of the principles of writing clean code is Don't Repeat Yourself, or DRY. Writing loops is one way to help us write more DRY code. Instead of repeating a block of code, we can tell JavaScript to run it multiple times.

Although there are many ways to do loops (or iterate, as its also called), we will focus on one for the course for now: The `for` loop.

Let's create a new file called `loops.js`

`for` Loop

So we've already looked at lists and collections of things, and seen how we can access individual items in that collection and do stuff with the items we access. But what if we want to do that stuff on all the items in the collection.

Let's work with a recipe example. We could write an entire paragraph about chopping each individual vegetable for our dish - or we could just say in the instructions:

Chop all veggies

This is essentially what we are going to do using JavaScript.

We will need an array for a `for` loop, since the idea is that we execute a block of code *for* each element in a collection.

Let's start with a simple example, looping through an array of numbers:

```
const numbers = [1, 2, 3, 4, 5]

for (number of numbers) {
    console.log(number)
}
```

So what is going on here?

When we are using the `for...of` syntax, we essentially break down items in an array, storing them in a sort of temporary variable called "number".

> Note: arrays should always be placed in a variable which is plural, and when we loop through them, we should name the temporary variable the singular term of the array's plural. It is not mandatory, but it's one of the most helpful naming conventions. NEVER use single letter variables, not even for for loop temp variables!

Essentially, if we have 5 items in our array, the first time the loop runs the value of `number` will be the first item in the array:

```
//first run
const numbers = [1, 2, 3, 4, 5]

   // number now has a value of 1
for (number of numbers) {
    console.log(number)
}
```

Once we reach the closing brace, the loop has no more work to do, checks if there are still items in the array it hasn't iterated through, and *loops* back to the beginning:

```
//second run
const numbers = [1, 2, 3, 4, 5]

   // number now has a value of 2
for (number of numbers) {
    console.log(number)
}
```

And so on, until the loop reaches the last values in the array!

Between the braces, we have the option to do whatever we want. The sky is the limit!

Let's combine this with a conditional statement. We will look for only the even numbers:

```
const numbers = [1, 2, 3, 4, 5]

for (number of numbers) {
    if(number % 2 === 0){
        console.log(number)
    }
}
```

We can do operations on these even numbers if we want to:

```
const numbers = [1, 2, 3, 4, 5]

for (number of numbers) {
    if(number % 2 === 0){
        console.log(number * 2)
    }
}
```

The complexity of the operations in the code block is up to us - and if you consider the possibilities using more data, or more complex values, this makes it an exciting prospect!

Let's think about common use cases for looping. Often we want to filter values based on a certain conditions - for example, filter out your friends on instagram who posted recently, find all posts that you've liked recently, find only youtube videos from creators you've subscribed to.

Since the result of this filtering is another collection, we should use another array to store values. Let's say we want to find all numbers that are even:

```
const evenNumbers = []
const numbers = [1, 2, 3, 4, 5]

for (number of numbers) {
    if(number % 2 === 0){
        evenNumbers.push(number)
    }
}

console.log(evenNumbers)
```

We need to only log the values out once we finish the for loop. After the closing brace we are finished with out looping.

Looping through objects

Copy the following into your file:

```
const employees = [
    {name: "John Doe", salary: 60000, department: "marketing"},
    {name: "Alice Cooper", salary: 75000, department: "engineering"},
    {name: "Seamus Finnigan", salary: 85000, department: "logistics"}
]
```

Remember when we said that objects are usually the way to store complex data? Now we can see how we can utilise their power the best!

Let's loop through all objects, and just log them out:

```
for(employee of employees){
    console.log(employee)
}
```

Now if you imagine printing out the employee's name on the screen, we would not use the whole object - the object only makes sense in a programming context. It's better to just get their names for example:

```
for(employee of employees){
    console.log(employee.name)
}
```

Let's go one step further! Let's consider the following 2 questions.

- What is the average salary at this company?

Calculating averages is fairly simple: We need the total of the salaries, and then divide by the number of employees.

We need the total to be outside the loop, because every time we would run a loop otherwise, we just reassign the total instead of keeping track of it.

```
let total = 0 // we start at 0 to have a number that we can use to add up value, a
nd we also make it a let to ensure we can reassign it, since this value will chang
e over time.
for(employee of employees){
    total = total + employee.salary; // we reassign the value, for each iteration,
 this number should increase by the current salary, and added to the previous tota
l.
}
console.log(total / empployees.length) // divide by the number of employees
```

Fantastic!

If you'd like to play with the values, duplicate the details of an employee, modify the values, run the code again, and see the changes in the result.

Imagine that we have the data of thousands of employees - now it makes more sense to use a loop, and to store values in objects.