Loops

What is a loop?

A loop is a piece of code that repeats. It is used when you have a situation which potentially requires the same segment of code to be run a number of times, until a certain condition is met.

Let's look at an example to get started. Our slightly silly program will ask the user for a

```
// The user is going be asked how many apples he wants, between 0 and 3.
let numApples = Number(window.prompt("How many apples do you want?"))
// (We use `Number` above to make it a number, not a string)
// We are going to add "apple" to this array for however many apples the
user asked for, so we will set up an array to put the 'apples' into.
let userAppleArray = []

// Here we add the logic to add the apples to the array.
// We need need to check three times whether we have hit our target.
if (userAppleArray.length < numApples) { // first apple
    userAppleArray.push("apple")
}

if (userAppleArray.length < numApples) { // second apple
    userAppleArray.push("apple")
}

if (userAppleArray.length < numApples) { // third apple
    userAppleArray.push("apple")
}
```

Running through this code we can see that we get a number from the user, and then we respond to that number. Because we can't predict in advance what that will be, we are at the mercy of the user, and so we need to have these repeated `if` statements.

Let's say the user enters `1`. We need to add one 'apple' to the array.

- We get to the first if statement, and `userAppleArray.length` is `0`.
- `0` is less than `1`, the user's number assigned to `numApples`..
- ..So we enter the brackets and run the `if` code, pushing an `"apple"` string into our array, which now looks like this:

```
["apple"]
```

We then hit the second `if` statement, and test the comparison. The length, `1`, is **not** *less than* `1`, and so the code isn't run. And we know that the same will happen for the third `if` statement.

We might be seeing the problem with this type of code. We need to write lines and lines of code to account for all the possibilities, and often they won't be required (although sometimes they will). What if the question involved `1000` possible apples? Or one million? Trouble..

So instead, let's try a loop:

```
// The user is going be asked how many apples he wants, between 0 and 3.
let numApples = Number(window.prompt("How many apples do you want?"))

let userAppleArray = []

// here we will replace the repeated `if` statements with a loop.
while (userAppleArray.length < numApples) {
    userAppleArray.push("apple")
}
```

We have the same result, but we are letting the `while` loop take care of repeating things for us. We can see that the condition is exactly the same. what a `while` loop does is repeat the code *while* the condition is true. This may be never, or in some instances, indefinitely.

Note that in most instances we will want the loop to break eventually, and this means that we will need some way to escape the loop.

We can make a different loop here, and have a look at some other posibilities:

```
let n = 0;

while (n < 5) {
    console.log("In the loop!")
    n = n + 1   // we could write n++ here too
}

console.log(n)
```

Run this code, and have a fiddle around with the values here to get a good sense of what is going on. Also play around with the comparison operator.

`break` - escaping a loop

```
let i = 0;

while (i < 6) {
  if (i === 3) {
    break
  }
  i = i + 1;
}
```

We can also escape from the loop with a `break` statement. There are some good examples of why you would use this, but they are probably best saved until you have loops under your belt a little more. For now, concentrate on setting the condition to be the limiting point, and we can discuss examples of `break` when they come up.

Logical Operators

In some cases you might need to ask a more complicated question in your condition, for example, whether a value is greater than a certain number, *and* less than another.

`true` and `false`

Before we kick off here, we need to discuss the idea of 'true' and 'false' in coding (represented by the booleans `true` and `false`). These are the designated values of the booleans in JS, and they are *not* the strings, `"true"` and `"false"`. These are very different things. The booleans represent the concepts of truth and falsity, and are their own special category. The main thing is to not confuse them with the strings (that are just strings).

AND (&&)

For example:

```
while(x > 12 && < 20) {
    console.log(`number ${x} is a 'teen'.`)
}
```

Here there are two conditions that must be satisfied for the loop to run - that is: that the number is greater than 12 *and* less than 20. We use the `&&` (AND) boolean operator to ask that *both* conditions be satified to make the whole condition true.

OR (||)

We could also have a situation where one of two conditions is enough to run the code. For example:

```
if (hour < 10 || hour > 18) {
  console.log('The office is closed.')
}
```

Here we are saying that *either* of the conditions can be met for the code to run. We are using the `||` (OR) operator (those are 'pipes', on the far right of the keyboard).

NOT (!)

The NOT (!) operator changes a `true` to `false`, and a `false` to `true`.

These are the most common boolean operators, and the ones that you should concentrate on for now. Keep in mind also that you don't want to make the logic in your `if` statement too convoluted. It's an area of code that you need to make as clear as possible.

Resource: https://javascript.info/logical-operators

Other loops

There are also `for` loops, and `do.. while` and a few more, but I don't tend to use them much, and I think concentrate on the concept of the loop with `while`, and if you want to branch out then you can have a look at this resource:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

Part of the way JS operates in the browser means that while loops are important, they also bring certain issues into play, and we will see this as we progress. The concept of loops is vital, however.

Summary

- Loops are a way to repeat a section of code.
- A while loop can be thought of as a repeating `if` statement.
- It comes with a condition, which is checked each time the loop is to run. If the condition resolves to `true`, then the loop runs the code between the containing `{ }`.
- It takes the form:

```
while(<condition>) {
    // do this code..
}
```

- We need to be careful that we have a way to break the loop.
- We saw a couple different ways of doing this:
    - With a changing counter that eventually makes the condition `false`.
    - With a `break` statement.
- Logical operators
    - AND (&&)
    - OR (||)
    - NOT (!)