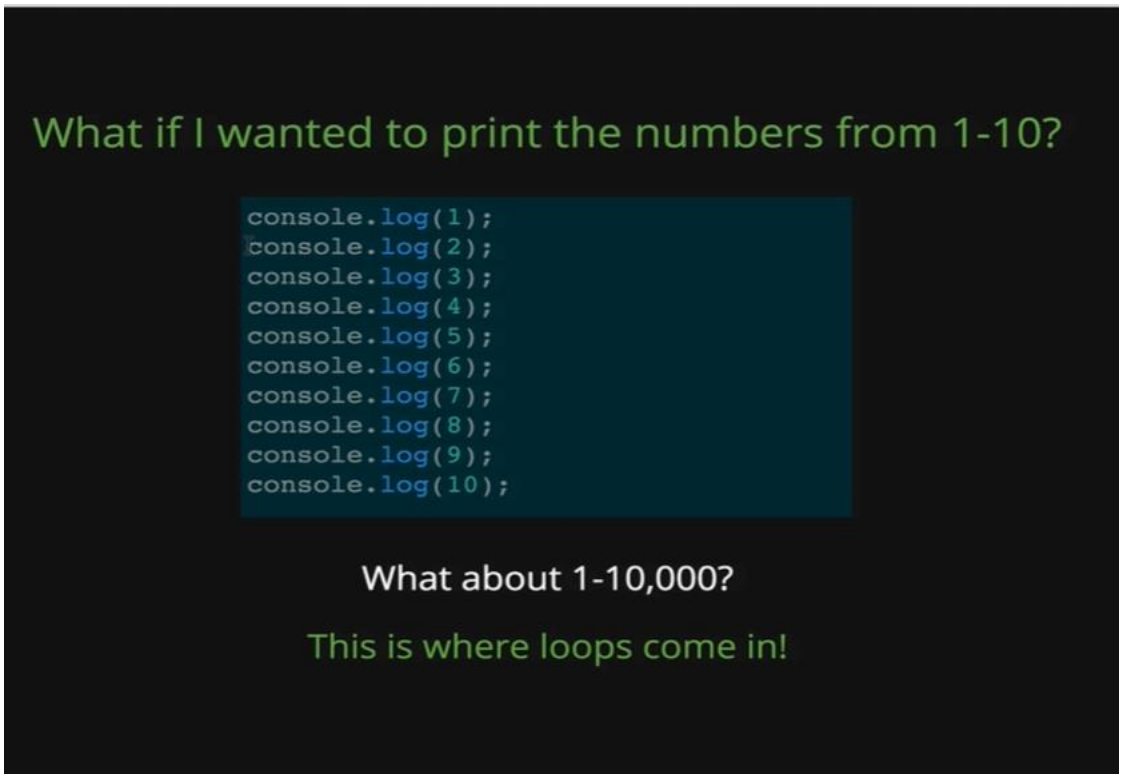
Objectives:

- *Understanding the purpose of loops*
- *Define "DRY" code*
- *Write simple while loops*

If I wanted to print the numbers from 1 to 10, each one on a different line with what we know so far we would need to have 10 different console.logs.



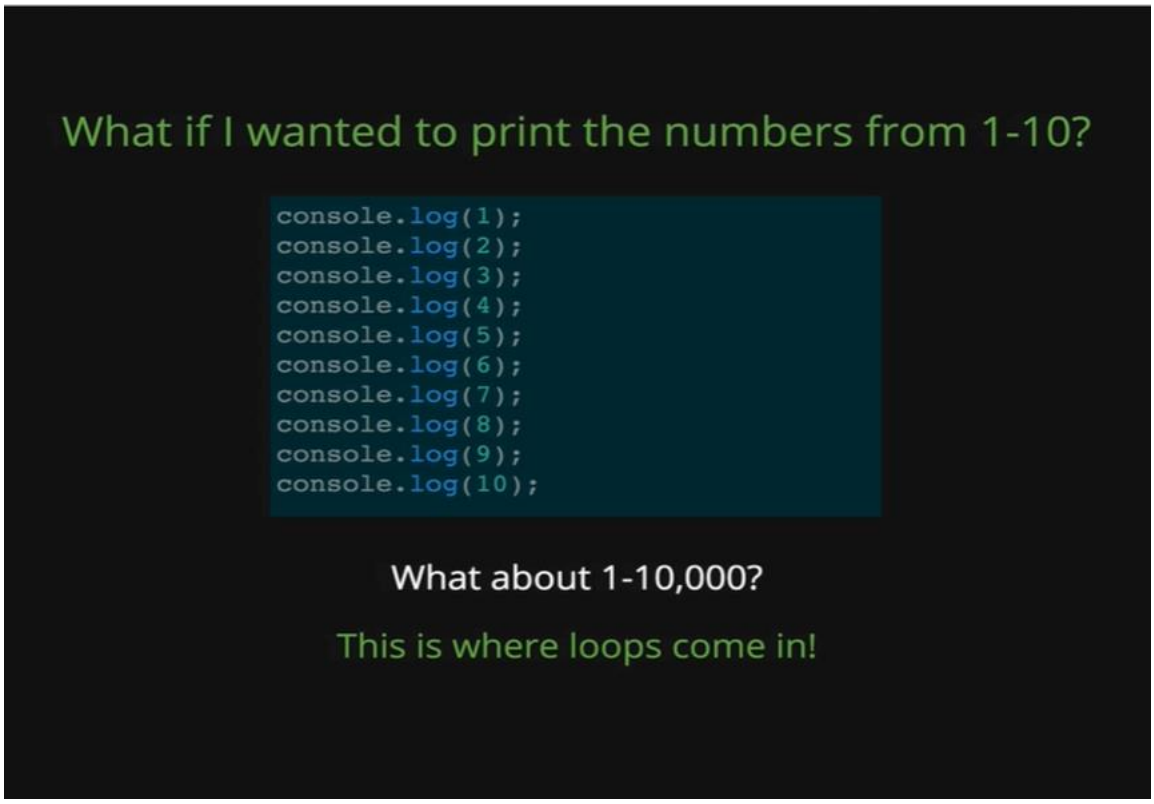
And that is not an ideal thing and what if we want to do all the numbers between 1 and 10000? Or the first million numbers? Suddenly, I am having to write a lot of codes myself. So, this is where loops come in, even though we would not be producing numbers 1 to 10000 in a real production application.

Lets take an example like Facebook, where a single photo or post can have 10000 different comments on it, like those photos that go viral, that gets thousands and thousands of comments, behind the seen there is some sort of loop that is being used to print out all the comments on the page rather than for every comment there needing to be a separate line of code.

So, before we see the syntax of "while" loops in JavaScript, we will know a new concept called DRY code. So, DRY stands for Don't Repeat Yourself, it's a concept that turned around a lot in

all sorts of programming languages but what it comes down to is that we do not want to repeat our code.

So, lets look at an example now,

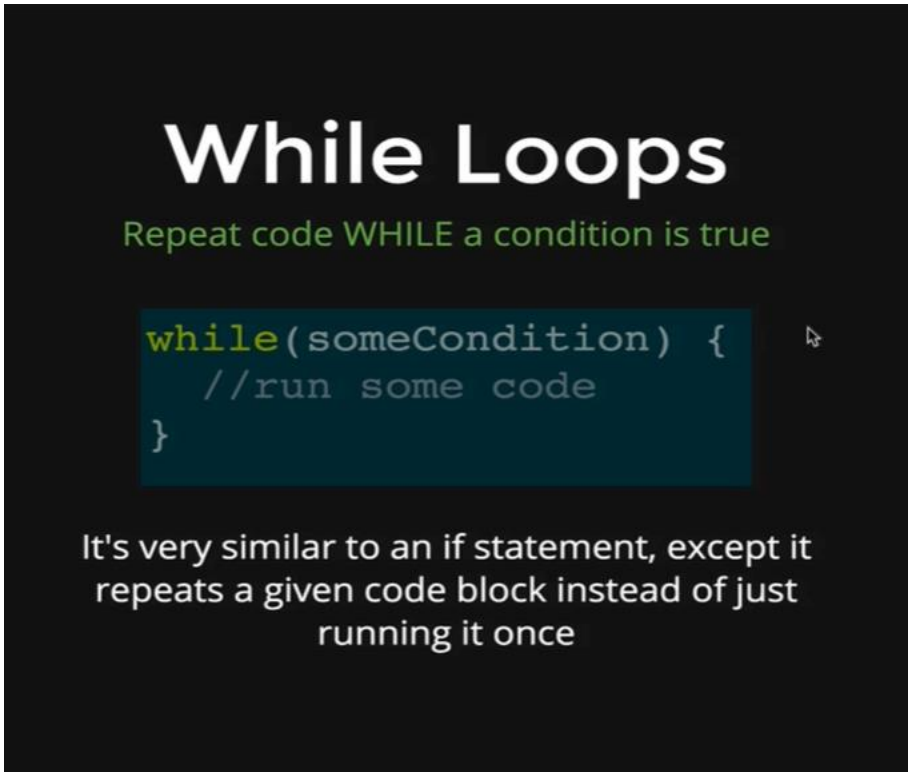


This code is very repetitive, the only difference is the number we are printing out but otherwise every line is the same. So, this code is not something that we will consider DRY, some people will call it WET, which I have heard WET stands for Write Everything Twice, that's not as common to hear though as DRY.

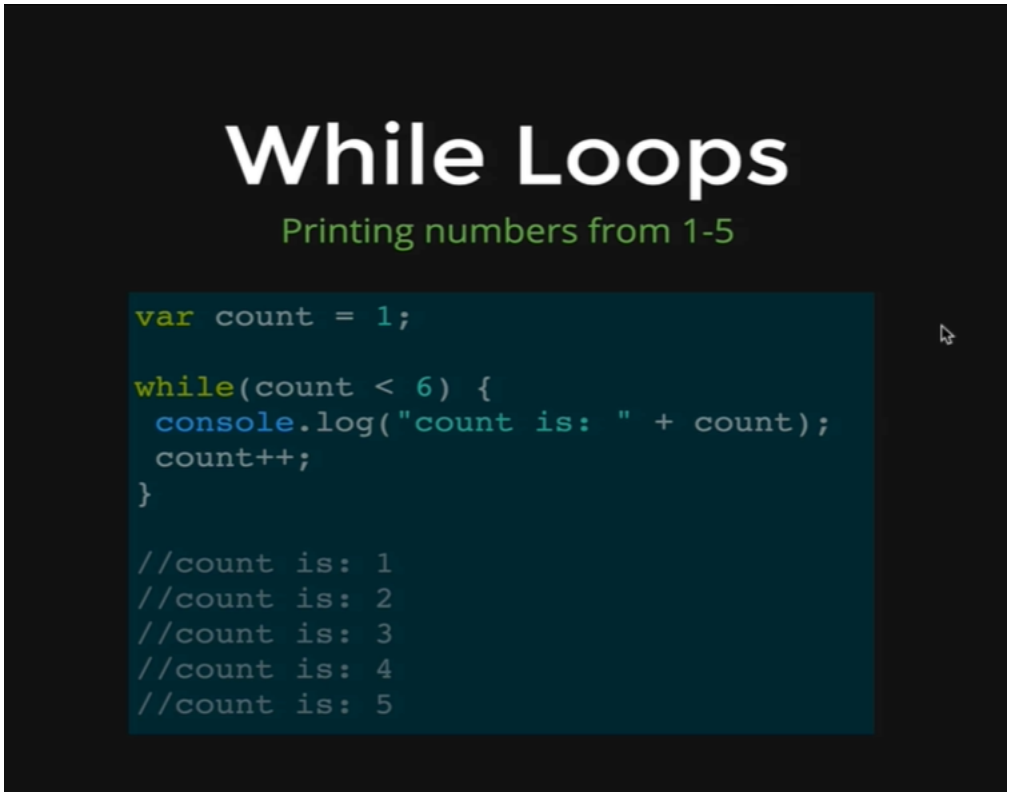
So, what loops do is they allow us to DRY up our code. They are one of the tools that is disposal, some of the other we will learn about are arrays, functions and objects, but loops are most fundamental ones.

So, we are going to start by talking about While loops. There are multiple types of loops that we are going to see the first one is the while loop.

The while loop is very similar to an *if* statement.

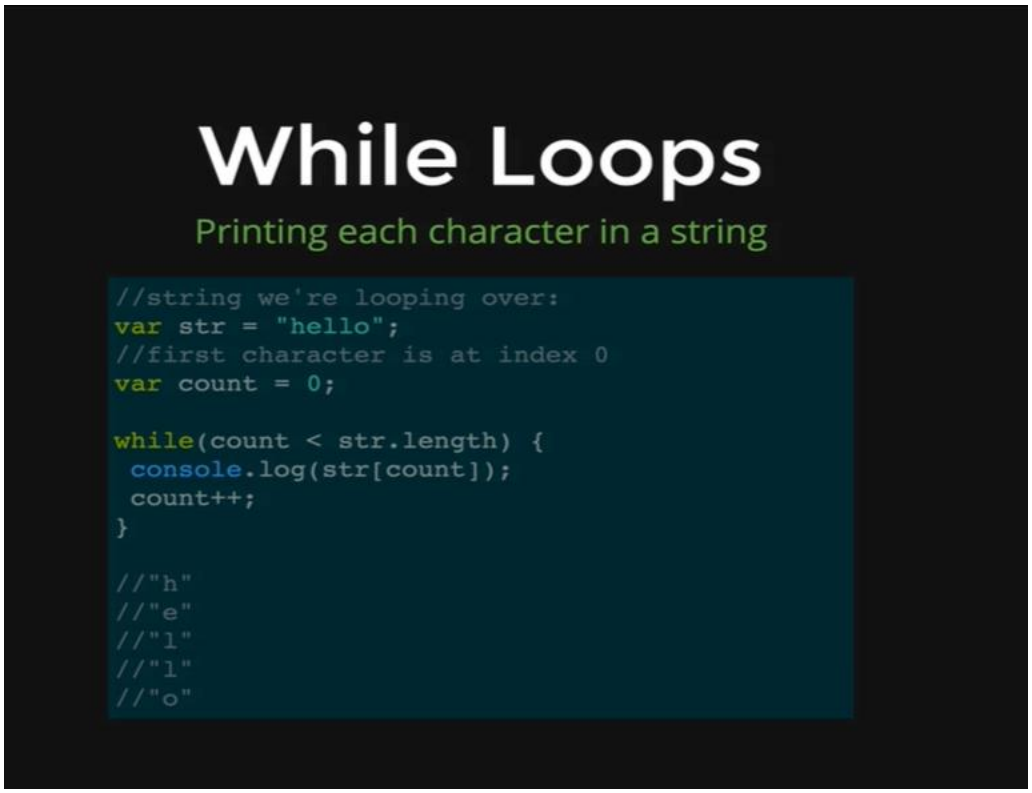


So, it takes a condition like x < 5 or answer != end and while that condition is true it will repeat the code that we put inside the curly braces. So, an *if* statement is very similar instead it does not repeat the code it just runs it one time. A while loop will continue to run the code as long as the condition is true.



Here is an example, this is how we can print the numbers from 1 to 5 using a while loop. We start with the variable called “count”, it could be named anything but count which is assigned to 1, meaning it will start its count from 1. Then we have our while loop that says count < 6, so the very first time this code runs count = 1, so 1 is less than 6 which means the condition inside the while loop is true and that’s why the code inside the curly braces is run, so that is going to print out *count is: 1* and then it will add 1 to count which increases the value of count to 2 and it goes back to the condition of the while loop and it checks, is count which is 2, is 2 less than 6, that condition is true so it prints again and then it adds 1 to count again and now count is 3, the while condition is true again and then it prints *count is: 3* and then adds 1 to count again increasing its value to 4 and so on, until the final time when count is 5, 5 is less than 6 we print out count is 5, we add 1 to count which is now 6 and then it tries to run again and then it realizes 6 is not less than 6 so then its done and that’s it.

So, as you can see a loop can save lot of our times by repeating codes itself and printing results instead of us writing all those results ourselves.



We have another example here where instead of just printing numbers we are using a while loop to loop through a string and print out every character separately. The output looks like h e l l o in separate lines, so that's five console.log statements and the way that we achieve that, we start with our string equal to hello then it will count which will be the number that we use to access a character in the string at the index and remember the first character is always at index 0. So, then what we are going to do is say while count is less than the string, the length is 5 (hello), so while the count is less than 5, we are going to print out the string with a character at index of count.

What that means is,

var str = "hello";

var count = 0;

str.length

➔ *5*

Count is 0, which is less than 5, thus the while loop condition is true and that's why we will console.log the count.

console.log(str[count]);

➔ *h*

So that is string of index 0, which gives us ‘h’ and then we add 1 to count and now count is from 0 to 1.

count

➔ *1*

So, we repeat the code in while loop again, is count < str.length?, is 1 < 5? The answer is Yes, so then we repeat the line again.

console.log(str[count]);

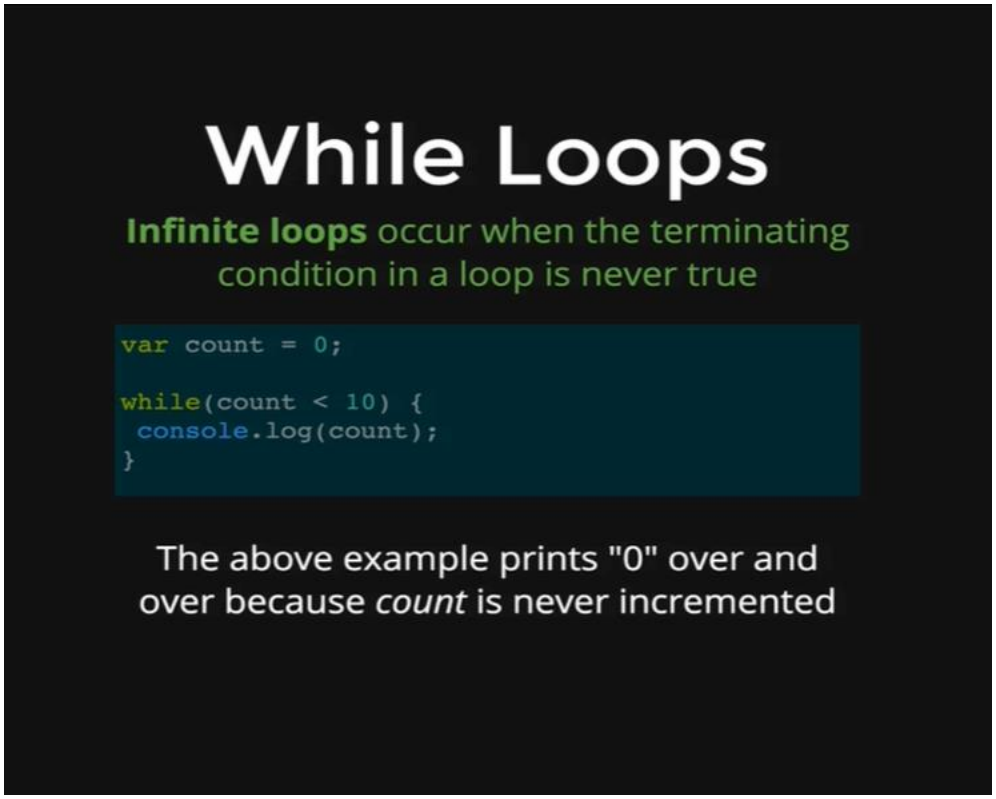
So now the count is 1 and we get printed ‘e’

console.log(str[count]);

➔ *e*

This keeps going on until the very last time and we print out ‘o’ and we add 1 to count which is now equal to 5 and 5 is not less than 5 so the loop is over. Remember that the length is always one greater than the highest index of a string, so the length is 5 characters, but the maximum index is 4 because we start at 0 and o is at index 4. So, that’s how we can use a loop to print out every character in a string.

One last note about while loops is that we can create something called an infinite loop if we are not careful.



An infinite loop occurs when the condition we provide is never false, so it just keeps going and going and going forever and these are obviously problematic, they can crash your browser, they can take up all your memory in JavaScript, its not something you would ever want to do. Here's an example of how one would happen, we have count equal to 0, and then we are saying while count is less than 10 console.log (count). Well, count is always less than 10 because it is 0 and we are never changing count so it is never incremented, it is never going to be over 10, and this will just print 0 forever.