**Loops**

**Intro to Loops**

There are often times that we want to execute a block of code, not once, but maybe a hundred times, or more. Writing codes in these situations can take millions of lines with the same instructions.

To solve it, we can use loops to repeatedly execute a block of code.

**While Loops**

Loops will let us ***iterate*** over values and repeatedly run a block of code.

var a = 0;

while (a < 50) {

a = a + 1

console.log(a)

}

**Parts of a While Loop**

There are many different kinds of loops, but they all essentially do the same thing:

* When to start: defining the starting value of a variable for instance.
* When to stop: the logical condition to test whether the loop should continue.
* How to get to the next item: the incrementing or decrementing step - for example x = x \* 3 or x = x - 1

var start = 0; // when to start

while (start < 10) { // when to stop

console.log(start);

start = start + 2; // how to get to the next item

}

**For Loops**

A lot of programmers prefer not using a while loop, once there is a loop that gives more control over the looping process: **for-loop**.

**Parts of a For Loops**

For-loops are the most common type of loop in JavaScript.

The difference between a for and a while loop is that a for-loop forces us to specify the starting and stopping point of our loop.

for ( start; stop; step ) {

// do this thing

}

for (var i = 0; i < 6; i = i + 1) {

console.log("Printing out i = " + i);

}

We need to use the semicolons to separate each statement of a for-loop.

**Nested Loops**

We can nest loops inside each other (just like conditional statements):

for (var x = 0; x < 5; x = x + 1) {

for (var y = 0; y < 3; y = y + 1) {

console.log(x + "," + y);

}

}

***Prints****:*

*0, 0*

*0, 1*

*0, 2*

*1, 0*

*1, 1*

*1, 2*

*2, 0*

*2, 1*

*2, 2*

*3, 0*

*3, 1*

*3, 2*

*4, 0*

*4, 1*

*4, 2*

For each value of x in the outer loop, the inner for loop executes completely. The outer loop starts with x = 0, and then the inner loop completes its cycle with all values of y:

*x = 0 and y = 0, 1, 2 // corresponds to (0, 0), (0, 1), and (0, 2)*

Once the inner loop is done iterating over y, then the outer loop continues to the next value, x = 1, and the whole process begins again.

x = 0 and y = 0, 1, 2 *// (0, 0) (0, 1) and (0, 2)*  
x = 1 and y = 0, 1, 2 *// (1, 0) (1, 1) and (1, 2)*  
x = 2 and y = 0, 1, 2 *// (2, 0) (2, 1) and (2, 2)*  
etc.

**Incremental and Decrement**

With loops we often need to increase or decrease the value of a variable, in order to step through the loop.

With JavaScript, and many other programming languages, we can use an increment operator, which works as a shortcut:

x++ or ++x // same as x = x + 1

x-- or --x // same as x = x - 1

x += 3 // same as x = x + 3

x -= 6 // same as x = x - 6

x \*= 2 // same as x = x \* 2

x /= 5 // same as x = x / 5

**Scope and Ways of Declaring Variables**

There are different ways to declare a variable, by using let, const or var.

To understand the difference between the three ways of declaring a variable, we need to know what a scope is.

**What is a Scope?**

The scope is defined as a specific portion of the code. There are three types of scopes in JavaScript.

1. Global Scope: when a particular variable is visible anywhere in the code - global variable.
2. Function Scope: when a particular variable is visible within a specific function only - local variable.
3. Block Scope: when a particular variable is visible within a block only ({. . .})

/\*

\* Global scope.

\* This variable declared outside of any function is called Global variable.

\* Hence, you can use this anywhere in the code

\*/

var opinion = "This nanodegree is amazing";

// Function scope

function showMessage() {

// Local variable, visible within the function `showMessage`

var message = "I am an Udacian!";

// Block scope

{

let greet = "How are you doing?";

/\*

\* We have used the keyword `let` to declare a variable `greet` because variables declared with the `var` keyword can not have Block Scope.

\*/

} // block scope ends

console.log( message ); // OK

console.log( greet ); // ERROR.

// Variable greet can NOT be used outside the block

console.log( opinion ); // OK to use the global variable anywhere in the code

} // function scope ends

showMessage()

**Variable Declaration**

There are three ways to declare a variable:

1. let: It’s a new way to declare a variable in any scope - **Global, Local, or Block**. The value of this variable can be changed or reassigned anytime within its scope.
2. const: It is also a way to declare constants in any scope - **Global, Local, or Block**. Once you are assigned a value to a const variable, the value of this variable CANNOT be changed or reassigned throughout the code.
3. var: This is the old way of declaring variables in only two scope - **Global, or Local**. Variables declared with the var keyword can not have Block Scope. The value of this variable can be changed or reassigned anytime within its scope.