Java Script

Learn Basics of Scripting Language and Use in Programming Easily

James Jackson

JavaScript

Learn Basics of Scripting Language and Use in Programming Easily

James Jackson

Introduction

JavaScript is one of the oldest programming languages known that are still in use by programmers today. The most beautiful part of this language is that it is relatively simple compared to other programming languages. JavaScriptis a very fundamental language to learn during your early years as a programmer. It has the flexibility needed to create very complex projects. The easy manipulation and richness of this language make it one of the simplest yet versatile programming languages on the current market.

Like other programming languages, the first things you should focus on are the basic elements of this language. Mastery of the building blocks of JavaScript will help you in the long run and will aid you in writing complex code.

JavaScript is often confused with Java, another programming language, but the two are entirely different object oriented programming languages. JavaScript was written in 10 days by Brendan Eich, a programmer working at Netscape. The prototype was written to complement Java which was being included in Netscape's web browser, Navigator. The language was originally named Mocha, but was changed in September, 1995, to LiveScript. In December, under an agreement with Sun Microsystems, who owned the Java language, LiveScript was renamed JavaScript.

There have been hundreds of thousands of JavaScript programmers up until now, and there are some mistakes that are made by every single JavaScript programmer during their early years. It is essential that you learn how to avoid these mistakes and learn to code in the proper way. You can look at JavaScript as being the first step which towards learning other different programming languages. You will be pleasantly surprised by how much easier it will be for you to master different languages once you have obtained a clear grasp on JavaScript.

So, let’s begin now shall we?

Chapter One

Syntax

The first thing you should know in JavaScript is Syntax. Syntax is a set of rules that governs the writing and interpreting of any program written in JavaScript.

The Basics of Syntax

var
try
new
finally
continue
for
instanceof
else
with
throw
in
catch
do
while
this
if
case
delete
debugger
switch
typeof
function
default
enum
void
static
break
return
super
extends
class
const
yield
public
protected
private
package
let
interface
implements
import
export

If you are absolutely new to the world of Programming, then you should know that every programming language is comprised of sets of specific rules. These rules are followed in order to create a structured blueprint which the computer recognizes and then executes. When we talk about a computer program, in the simplest terms, we are referring a list of instructions which tells the computer what to do and what to execute.

These instructions are given through the various rules, which are called "**Syntax**". In short, "**Syntax**" is the set of rules that defines Fixed Values and Variable Values.

Imagine that you are in a new class in college and the instructor comes in and writes the following message on the board: "21 read 14 6 to problems to 35 do pages and." You're going to be very confused. Even though you recognize the words, you don't know what to do with them because they aren't in an order or format that you recognize. That's because this hypothetical professor isn't following the Syntax rules of the English language. If the professor writes, "Read pages 21 to 35 and do problems 6 to 14," you know exactly what to do.

The same is true of your computer. If you don't use the right order and correct format, your computer won't know how to proceed with your program and will stop, creating errors as it tells you that it doesn’t understand what you want it to do.

The most basic elements include but are not limited to **Statements**, **Variable Naming** and **White Space Syntaxes.**

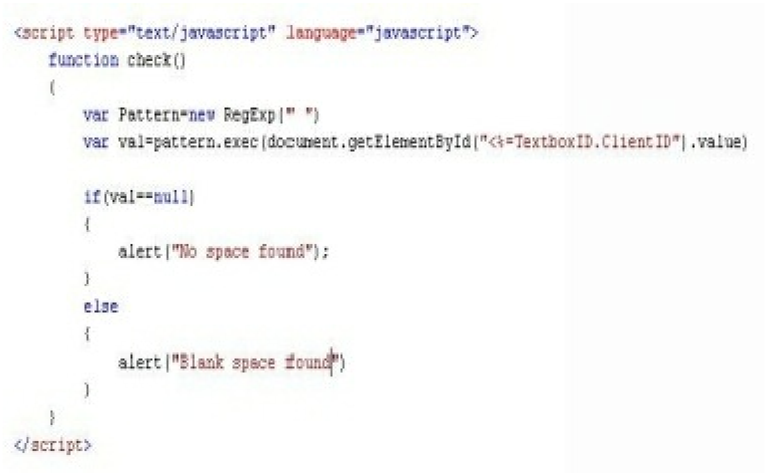
When working with a programming language, all the instructions are called “**Statements**”

Before moving on with the statements, there are some important things you need to know.

JavaScript is Case Sensitive

JavaScript is, in fact, a case sensitive language. This might create some confusion for those who have already learned multiple programming languages. We know there are many languages out there that are not case sensitive. When writing code in JavaScript, you will always have to start the name of constructor with a capital letter which is followed by the name of a function in lower-case letters.

Whitespace



```
<script type="text/javascript" language="javascript">
    function check()
    {
        var Pattern=new RegExp(" ")
        var val=pattern.exec(document.getElementById("<%=TextboxID.ClientID").value)

        if(val==null)
        {
            alert("No space found");
        }
        else
        {
            alert("Blank space found")
        }
    }
</script>
```

If you have browsed through some of our sample codes, you have probably noticed the presence of a lot of blank areas outside the string constants. These spaces, tabs and newlines are referred to as “Whitespace”. One of the core differences between other languages and JavaScript is that is that in other languages it does not really matter how much blank space you leave between lines of your code, but in JavaScript, the whitespace directly impacts the semantics of the program itself.

The Significance of Semicolon

The statements written in JavaScript are mostly separated by **semicolons**. One key factor at play when writing code in JavaScript is a function called

Automatic Semicolon. This feature is present in the language to make it a little easier for users. The way this works is that whenever a statement is written before a new line has been parsed, the first statement will automatically be taken as a complete and correct statement, acting as if a semicolon was in place. The option is always available for you to adjust the setting so you can prevent the automatic semicolon insertion mechanism. If you prefer to manually inserting semicolons in your code; this will eliminate undesired insertion of semicolons.

A Deeper Look at the Statements

The statements in JavaScript are made of various components. They create the language commands and instructions of the language itself. Above is a chart that shows the most common kinds of statements used in JavaScript.

- **Values**
- **Operators**
- **Expression**
- **Keywords**

- **Comments**

The above components are some of the most basic elements of JavaScript. We shall now go through each of them in brief details.

Values

When programming in JavaScript Syntax, you will come face to face with two types of values. Namely

- **Fixed Values** which are also called literals.
- **Variable Values** which are variables with interchangeable values.

Fixed Values Rules

When working with fixed values, there are two rules you should keep in mind:

Numbers can be written either with a decimal point or without a decimal point.

Ex: 10

10.10

Strings (Text) can be written with either double quotes or single quotes.

Ex: “I am Jim”

‘You are Henry’

Variables Rule

A slightly different rule is followed when writing JavaScript **variables**(Variables act as containers to store data values).The **Var** keyword is used to <u>declare</u> variables**.** Also, an <u>equal sign (=)</u> is used to <u>assign values</u> to the declared variable.

For Example:

var a;

a = 9;

(a is defined as a variable, and a is assigned the value 9)

Similarly, you can put a string in a variable as well.

var food = “I Love Food”;

As we have mentioned earlier, the white spaces does play an important role in

JavaScript programming, in this scenario, the white spaces inside the quotations marks will act as blank spaces.

```
var food = “ I Love Food” ;
```

When naming your variables, use words that are meaningful instead of fragments or shortcuts. Consider if you want to create a variable that will handle the names of colleges you want to attend. You could define your variable as such:

```
var = schlnam;
```

and that is a valid naming scheme. However, when it comes time to debug your code to make it work better, you might not remember what that variable means. Instead, defining it as:

```
var = schoolNames;
```

makes it clear what the variable does and what it contains.

Keep in mind that variable names are generally not capitalized when creating them in JavaScript. That’s because JavaScript is a case sensitive language and capitalization is reserved for object types, such as Strings. That isn't to say that capitals aren't used in variable names. Use capitals when making variables with multiple words in their names to make them easier to read. Consider the following:

var = schoolnames

or

var = schoolNames

By capitalizing the second word, it makes the variable name easier to recognize.

JS Operators

There are seven different Arithmetic Operators that are used to calculate values in JS. These operators are addition (+), subtraction (-), multiplication (*), division (/), modulus (remainder) (%), increment (++), and decrement (--).

You're probably familiar with the normal math operators of addition, subtraction, multiplication, and division, so we're going to just cover the three you probably don’t know.

Modulus is used to return the remainder after one number is divided by another. Increment is used to add 1 to a number and decrement is used to subtract 1 from a number.

Consider the following where y = 5.

For x = y % 2, x will have a value of 1. That’s because 5/2 = 2 with a remainder of 1. For x = y % 3, x will have a value of 2.

With increment and decrement, where you place the ++ or – matters in determining the final value.

The two calculations of x = ++y, and x = y++ will have different results for x, because JavaScript will do the operations in order.

In the case of x = ++y, JavaScript says, okay, first I have to increment y, then I have to assign that value to x. So the end values will be x = 6, BUT ALSO, y = 6.

In the case of x = y++, JavaScript will first assign the value of y to x, THEN increment y. That makes the return values x = 5 and y = 6.

This is similar for the decrement operator --. If you say, x = --y, then y = 4 and x = 4. If you say x = y--, then y = 4, but x = 5.

When doing simple calculations using the built-in mathematical operators, parentheses play an important role as they determine the operating order of the calculation.

Ex-

- 3 * 2 + 6

  This calculation will print a value of 12 as multiplication gets the greater priority.

- 3 * (5 + 3)

  This calculation will give you a value of 24, as the presence of the parentheses will force the program to calculate the values inside the parentheses, and then multiply it with the value outside.

When writing codes of greater lengths, you will sometimes want to make sure that your codes are a little bit more accessible, which is done using **Tabs.**

When you using a tab while coding, it will not hamper or alter how the code is being read or executed, but it will simply make it easier for you and the other members of your team to read it more easily.

Ex:

```
var food = “ I Love Food()

          {

                    console.log( ‘hello’);

          };
```

A Brief Intro to JavaScript Expressions

When talking about expressions in JavaScript, we are not talking about how the program will react if it is sad or happy! In JavaScript, by an expression we generally refer to a combination of operators, variables and values all of which the computer reads when processing the program. The calculating process is referred as evaluation.

For Example:

2 * 10 evaluates to 20

JavaScript is very flexible. You are not bound to only use fixed numbers when doing a calculation. You can even use variables to compute something.

For Example:

X * 24

Another Example:

var food = 14;

2 * food

This will return a value of 28 to you, as during computation, the value of “food” is read as 14, because its value was already declared on the previous line.

The values which can be used in an expression are not limited to numbers, you can use strings as well.

“Alpha” “+” “Delta”

This will evaluates to “Alpha Delta”

What are JavaScript Keywords?

Keywords are command identifiers which are pre-installed in the JavaScript database. Keywords tells the browser to perform specific actions. For example, the **var** command we have used to create new variables is actually a keyword. It tells the browser that a new variable needs to be created.

For Example

```
var a = 8 + 9
```

A new variable is created. It has the computed value of 8 +9 which is 17

```
var b = a * 10
```

Here, the variable b gives the value of a*10. Since we have already declared the value of a in the previous line, which is 17, the final result for the value of b will be 17*10, which evaluates to 170.

Comments

When coding, there will be times when you want to add notes to your code as documentation or prevent test lines of code from being executed. How can you do that?

In JavaScript, if you put a double Slash // before acode, or write a code in between /* and /*, then the code will be treated as a comment,which means it will be ignored during execution.

Ex-

var a = 8; //

This will be executed.

On the other hand,

// var a = 9;

This command will **Not** be executed.

A glimpse at identifiers

Identifiersare used to name the variables, keywords, and functions. These identifiers are often used to make it easier for programmers to recall the purpose a variable or function they set.

Identifier has to be a letter, an underscore (_), or a dollar sign ($). The remaining characters used in the naming convention can be anything from letters, digits, underscores, and dollar signs. Identifiers named using this convention are called **Legal Names.**

Ex-

varIcedtea

var $IamA_Robot

Some Common Mistakes to Avoid

Don’t forget to apply the curly brackets while writing your code.

Don’t forget to put semicolons after each statement.

Always remember that you have to use **var** to declare a variable before using it.

Practice

Give the basic definition of the following with examples:-

a) Syntax

b) JavaScript Statement

c) JavaScript Expression

d) JavaScript Keywords

e) Identifiers

Write a simple program which creates two variables and evaluate the sum of those two values.

Write a simple program which combines a string variable and a number variable and evaluate to the result.

Chapter Two

Introduction To The basic Operators

The basic operators in JavaScript are created with signs and symbols. We have already familiarized ourselves with a few operators, such as the equal sign operator which is used to assign value to a variable.

Ex-

```
var a = 8;

var b = 9;
```

The meaning of the above two lines is that the variables a and b are being declared and a value is assigned to them. In this case, the value of a is 8 and the value of b is 9.

Similarly, JavaScript has a number of other basic **mathematical operators** built inside its core script. Let’s look at the following four operators:

- The addition operator (+)
- The multiplication operator (*)
- The division operator (/)
- The Subtraction Operator (-)

Each of these operators works the same as how they would work in a calculator.

Ex-

```
var a = 10;
var b = 5 ;
a/b
```

This is a simple division, which divides 10 by 5 giving 2.

```
a + b
```

This operation performs an addition, 10 + 5 gives 15.

```
a – b
```

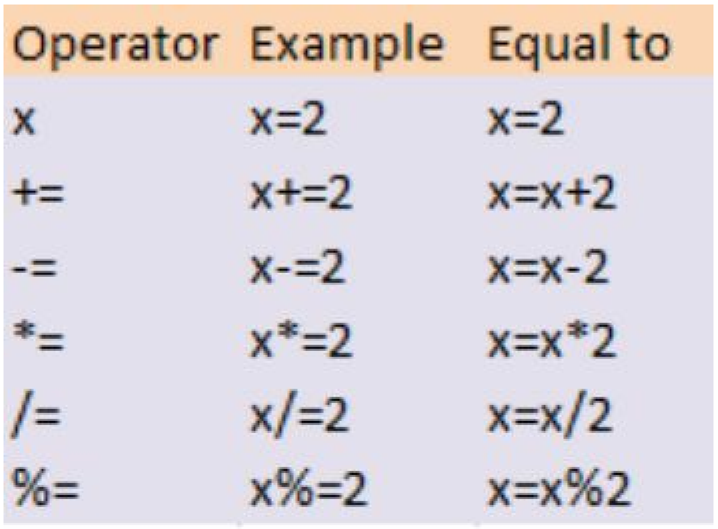
This operation does a subtraction, 10-5 gives 5.

a *b

This is a basic multiplication, 10 * 5 = 50.

Aside from the basic operators, here are some more:

Different types of assignment operators (Usage: Assigning values to variables)



Operator	Example	Equal to
x	x=2	x=2
+=	x+=2	x=x+2
-=	x-=2	x=x-2
*=	x*=2	x=x*2
/=	x/=2	x=x/2
%=	x%=2	x=x%2

The examples in the chart are all numbers, but operators are not only for numbers.

Concatenationallows you to combine two individual strings together and turn them into one string using + operator or += assignment operator.

Ex-

var Spray = “Hello” ;

var Body = “World” ;

console.log(Spray + “ “ + Body);

This will give an output of “Hello World”

Ex2-

var a=”I am ”

var b +=”Legend”

The result will be: I am Legend

Also, you can add a string and a number together:

X=”3”+1 (Result will be 31, not 4)

X=”Jessica”+1 (Result will be Jessica1)

The result will be a **String**.

Unlike other programming languages, the behavior of the operators that deal with Strings and those that deal with numbers vary to a great extent. Having a deep understanding of the core differences between the two is important in order to make sure that both of them are used properly.

For example, one of the prime differences you will notice while using these two types of operators can be found during the usage of the Addition Operator.

See the following example:

Consider a scenario where we are trying to add a number and a string.

Example 1.

```
var Bar = 5;

var Legal = “6”

console.log (Bar + Legal)
```

The output here will be 56.

If you want to force the strings to act as simple numbers, you can add a Number keyword in front of the string variable.

Example 2.

```
X=1+Number(“6”)
```

The result will be 7 in this case.

Example 3.

```
var Bar = 5;
```

var Legal = “6”;

console.log (Bar + Number(Legal))

The output of this statement is 11.

These are examples of how JavaScript allows you to do type conversion manually from numbers to Strings, Strings to numbers, or Boolean values to numbers or Strings.

We’ve already shown you how to convert Strings to numbers using the global Number() method. A similar method exists to convert numbers to Strings. This uses the global String() method. See the following Examples.

Example 4.

String (3.14) will convert the number 3.14 to a String.

String(x + y) converts the expression of x + y into a String.

You may wonder why you would ever want to convert a number to a String. There are times when you will want to preserve the accuracy of a number out to a certain number of decimals. Because of how computers deal with numbers by converting them to binary and back, you can lose accuracy when working with just the number type. In this case, preserving the value as a string ensures that you won’t lose any accuracy in your values.

You can also convert Boolean values into numbers or Strings.

Number(TRUE) will return a value of 1

Number(FALSE) will return a value of 0

String(TRUE) returns a String, “true”

String(FALSE) returns a String, “false”

The Logical Operators

When you face a scenario where you need to code something that will only work based on the results from two or more conditions, you will need to utilize the logical operators that are built in the JavaScript framework. The logical operators are **AND(&&)**,**OR(||)** operators, and **NOT(!)** Operators. These work by comparing the Boolean value from each comparison. There are only two Boolean values, TRUE and FALSE.

For Example:

X && Y

The result of the AND operator is true if both X and Y are true.

The result of the OR operator is true if either (X or Y) or both are true.

Example for Not Operator:

!X

The result of the NOT operator is true if X is not true.

To be more detailed, here are some more examples of the AND, OR, NOT operators.

Ex-

x = 5

Y = 0

(x>3 && y<3) returns true because 5 is greater than 3 AND 0 is less than 3.

X=3

Y=2

(X==3 || y==3) returns true because 3 = 3. It doesn't matter that the y portion of the comparison is false because OR only cares if one OR the other is true.

```
X=1
Y=5
```

!(x==y) returns true because the condition inside the parentheses is false. The opposite of false is true.

The function of the AND operator is to allow a code to run only if both of the given conditions is found to be valid.

On the other hand, An OR operator only allows a code to run if either one of the given conditions are found to be valid.

At last, the NOT operator allows a code to run if the given condition is found to be not true.

From the previous example, you saw the “==” symbol. This is one of the comparison operators.

Comparison operators are used to compare operands. A logical number will be returned if the comparison found to be true. Or, an action will be taken if the result is true.

For example-

if (gender ==“Female”) Text = “Sorry, our content is only for men”;

In the examples we used for logical operators, we used equal operator (==). It means returns true if both operands are equal.

In the summary of this chapter, we include a list of the different types of comparison operators.

The Conditional Statement

If you want to assign values to variables under a certain condition, then you can use the “if” and “else” blocks to easily manage certain pieces of your code. This will decide which will run and which will not, depending on your adjusted conditions.

Another way of doing this is by using the conditional, or ternary, operator. The syntax for a conditional statement is:

variablename = (condition) ? value1:value2

What this operator does is look at the condition and if the condition is true, it assigns value1 to the variable. If the condition is false, it assigns value2 to the variable.

Example

```
var role = (gender ==male) ? “King”;”Queen”;
```

If the gender is male, the value of the variable will be king. Otherwise, the value will be “Queen”.

Another example of how to do the same thing using an if / then statment:

```
var x= male;

var y = female;

if (y)

        {

                Console.log (‘Queen’);

        }

if(y)

        {
```

```
			// this code will not run because the previous if statement has
already evaluated y to be true.
	}
```

Or

```
	{
			if (x)
	{
			Console.log ('King');
	}
```

You might be wondering now what the point of using so many curly braces is, right? Well, always keep in mind that the curly braces are actually implemented into the programming language to help you create more readable and accessible codes. Also, these curly braces help you determine the scope of your variables as well. For single sentences (even if the statements don't require you to use curly braces), it is still a good habit to use them because they help you avoid errors and confusion. This will help during your future endeavors in writing longer code with various if/else blocks.

Some Common Mistakes to Avoid

Keep in mind not to accidentally use the “=” assignment operator for undesired results.

Make sure you are absolutely clear about the AND (&&) and OR (||) Operators. At the beginning, it might get a little bit confusing, but these are very essential for large scale programming and they will get easier as you practice.

Practice:

Write a simple program that will have three individual values in variables, and find out their mean.

Write a simple program which utilizes the || operator in order to execute a simple statement.

Summary

Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical "and" evaluates to true when both operands are true	3>2 && 5>3	False
	||	Logical "or" evaluates to true when either operand is true	3>1 || 2>5	True
	!	Logical "not" evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	"A"+"BC"	ABC

Chapter Three

An Introduction to Loops

Loops are for when you have to do a similar task multiple times. Say for example, you have to write the names of your guests who are coming to your event. How are you going to translate that into JavaScript? Sure, you can go ahead and try to set each name into a specific variable, but this will be very time consuming. It might work for you when there're maybe ten or twenty guests, but how about when there are a hundred? You don't want your arm to fall off right? Therefore, it is essential to learn the basics of "Loops".

Loops in JavaScript simply enable a programmer to do a similar task repeatedly with very little effort.

For example, instead of writing:

```
Text += houses[0] + "<br>"

Text += houses[1] + "<br>"
Text += houses[2] + “<br>”

Text += houses[3] + “<br>”
```

You can just write:

```
for (i = 0; i<houses.length; i++) {
    text += houses[i] + "<br>";
}
```

In the example below, I have thought of a scenario where I want to walk 8 steps to the east. This is what the code will look like –

```
for ( step = 0 ; step < 8 ; step++){‘walk east one step’}
```

The above code is a simple code that starts running from step 0, and after taking each step, it adds a value to the step variable increasing its value by +1. The cycle keeps on running until the value of the step variable reaches 7, which is when the program stops.

Using loops is fun, and you should be happy to know that this is not the only kind of loop that’s available at your disposal. In fact, there are a wide variety of different loop statements that you will be able to use depending on the task.

All loops fall into one of two categories: pre-condition and post-condition. In a pre-condition loop, the check on the condition is made before the code inside the loop is processed. This means that the loop may not run at all if the condition is not satisfied. On a post-condition loop, the condition is checked after the loop has run. This means that the loop will always run at least once. This distinction is important to keep in mind when you think about what kind of loop you want to use.

Let’s now have a look at the different types of loops in JS.

1. **The For Loop**

   TheFor loop is a primary example of a pre-condition loop. The loop is pretty simple and will keep on repeating itself until the specified condition has been fulfilled, or in other words, the specified condition has been deemed false.

   The skeleton of the For loop is –

   for([initialExpression] ; [condition] ; [incrementExpression])

```
{

Statement

}
```

For example, say we want to print the String, Hello World! five times. We can either do this by repeating the command console.log(“Hello World!”); five times or we can create a for loop as follows:

```
for (vari = 0; i< 5; i++) {

            console.log(“Hello World!”);

}
```

Whenever the conditions of a For loop is satisfied, the following processes are happening inside of it-

- If the conditions are met, then the initializing expression (if any) will be executed. In the above example, the initial expression sets the variable i (which serves as a counter) to 0.
- After which, the given condition will be evaluated. If the found

condition is true, then the loop will keep on executing itself, until it has reached a state where the given condition appears to be false, in which case the loop will stop. If however, the condition expression has been completely removed, then the loop will keep on executing itself infinitely.The condition is that i should be less than 5.

- Once the loop has executed itself, it will look inside for a statement which it needs to be run. If you want to have multiple statements inside your loop, then you can use a block statement ({...}) to separate those statements in particular groups.
- Once the statement running is complete, the loop will look at the incrementExpression, where it will update the Initial Expression and keep on repeating the cycle over and over again. On the first loop, i = 0. On the second loop, i = 1. Thus, you can see that it will take five loops for the condition to return a Boolean value of false where i< 5 (0, 1, 2, 3, 4). On the sixth loop, i will equal 5, making the condition false. It is important to note that you can

also use any comparison operator for the condition. In this way, for(vari = 0; i< 5; i++) and for (vari = 0; i<=4; i++) will both run the same number of loops.

The proper way to use the For Loop is:

```
  for (statement a; statement b; statement c) {
code block to be executed
}
```

Statement a is executed before the code block starts. Statement b is the condition to run the loop, and statement c is executed each time the code block has been executed. (JavaScript For Loop, from W3school.com)

One thing to remember is that you don’t have to use the increment operator to change your condition variable. You can also use the decrement operator, which will allow you to perform a countdown. Consider the following code:

```
for(vari = 30; i>= 0; i--)
        {
                await sleep(1000);

                console.log(i);

        }
```

This code will count down from 30 to 0, printing a new number every second. You can also use this code in a program without the console.log command to set an internal timer, making it a handy way to delay another function.

You can also use a for loop to iterate through an Array to find a certain value. Let’s say that you have an Array of phone numbers and you’re looking to see if a certain one is stored. You create an interface that accepts a phone number as an input variable, number and then searches the phoneNumber array to see if it exists. The for loop will iterate through the array and will return a true value if the phone number is found. This for loop would look like this:

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

        if (phoneNumber[i] == number) {

                console.log(“true, phone number found in entry# ” + (i+1));

        }
}
```

The for loop looks through the phoneNumber array at each cell and compares the value of that cell with the number variable. If they are the same, it prints where the phone number was found. You may notice that the cell number has 1 added to the value. That's because the first cell in any Array is initialized at zero. So phoneNumber[1] is actually the second entry in the array.

2. **The Do/While Loop**

This loop is a post-condition loop. That is because the loop is run first, then the condition in the while statement is checked. The loop will keep running itself until it has found a specified condition to be false.

```
 do {code block

}

While (condition)
```

Ex-

```
do {
```

```
i+ = 1;

console.log(i);

}

while (i< 5);
```

The above example will keep on repeating itself until the value of i has reached 4 which is less than 5, as specified by the condition.

3. **The while statement**

Unlike the “Do while” loop, the while loop will only execute the statement if the condition is met. This makes it a pre-condition loop.

The while loop has a similar structure as Do/While Loop:

```
While (condition){

code block

}
```

Ex-

```
        n = 0;

        x = 0;

        while (n < 3)

        {

    n++;

    x+ = n;

}
```

What's interesting is if you omit statement a, and statement c of the For Loop, it's the same as the **while** loop.

4. The Label Statement

The label statement can be described as a sign board for a specific loop. Once you have named a loop using the label statement, you will be able to refer to the loop throughout your program using the specified name, after that you can use the **break** or **continue** statement to command the program to either run or disrupt the execution of the loop. The naming can be done using the traditional methods of choosing JavaScript variables.

The structure is:

```
   Label :
    Statement
```

Or

```
markLoop:


while (theMark == true)
       {
                doSomething();
       }
```

Now that you have labeled this loop as markLoop, you can simply call it anytime. When you call it, the loop will run as if you had typed it out. That means you can refer to it in your code at any given time without having to rewrite the loop in its entirety. This becomes extremely useful when you have a loop that has a lot of sub loops nested inside.

5. **Break Statement**

The break statement is used to terminate the execution of a loop. It can be either used alone on its own, or in conjunction with the label loop. When it is used in collaboration with the label statement, it will terminate the actions of the specified label. On the other hand, when it's used on its own, it will terminate the nearest enclosing while, do/while, for or switch statement immediately.

The structure is:

```
break;

break label;
```

The first one is when you use break as a singular command, while the second one is when you use it in conjunction with a label.

Ex-

```
for( i = 0; i<a.length ; i++ )

{
```

```
if(a[i] == theValue)

{

Break;

}

}
```

6. Continue Statement

This statement can be describedas the opposite sibling of the break statement. It has all the features of the Break statement. And this statement can also be used in conjunction with a label statement or on its own. The only difference is: instead of terminating the execution, this statement will actually continue the process of execution. When using with label, it will continue the labeled statement. When used on its own, it will continue the nearest enclosing statement before going to the execution of the immediate next statement.

Here is the syntax:

```
continue;
```

```
continue label;
```

or

```
i = 0;

n =0;

while (i< 5)

{

i++;

if ( i == 3)

{

continue;

}

n +=I;

}
```

The above example is for a code that has a while loop with a continue statement. The code here only executes when the value of i is 3.

7. **For….in statement**

This statement iterates a specific variable over all the enumerable properties of an object. This type of loop allows you to access the keys of the object, but it doesn't provide a reference to them. This distinction is important, because it means that you can do an action when a value is present, but it does not allow you to manipulate that value.

The syntax is :

```
for ( variable in object)

{

Statement

}
```

8. **For…of statement**

This loop can be used if you want to browse through the indexes of other objects such as Array, Map, Set, Arguments amongst others. This statement will allow you to invoke a custom iteration hook with statements that will be individual for each of the distinctive properties of the objects. For...of was created because when for...in was used with Arrays, you could get additional properties returned that you weren't expecting. This made for...in unsafe to use with Arrays because when you iterated through them, you could get values returned that would throw errors.

The basic syntax is:

```
for ( variable of object)

{

statement

}
```

Some Common Mistakes to Avoid

While setting up the conditions for a loop, keep in mind the proper use of the operators (<) (>) and (==). A simple misuse of any of these operators can ruin the whole program.

When using the ‘label’ statement, make sure to use names that are easy to remember so that you will be able to access them comfortably in the future.

Have a good look at the situations where you are using the break and continue statements. Be aware that these two statements will only affect the code inside that particular parenthesis.

Practice

Write a simple program that uses the For Loop.

Describe the working regime of an For Loop

Write a simple program which uses the break and continue statements.

Write a simple program using the label statement and call the labeled statement at a later part in your program.

Chapter Four

The Basics of JavaScript Function and Scope

Let's see now, normally when we are talking about a 'function' what's the first thing that comes to your mind? Obviously an activity right?

The functions in JavaScript also work in a similar manner. Functions are blocks of codes which are required to be executed over and over again by the program. A Function is used to perform a particular task. A Function can contain any number of arguments and statements; they can even have none. Depending on how the structure is coded, it may or may not return any value to the user.

A Function is declared in the following manner:

```
Function name(              ) { /*code blocks to be executed*/}
```

To start the Function, we start with the function keyword, and then we add the **name,** and parentheses**.** To finish the function, we place the code blocks to be executed in a curvy brackets.

You can also put the function in a named variable-

```
var jam = function() {/* code blocksto be executed*/}
```

Below are few examples of how you can execute a function-

i) This is the example of the most basic function

```
varsayHello = function(person, greeting) {
var text = greeting + ' , '+ person;
console.log(text);
};
sayHello (' Jessica' , ' Hello');
```

ii) This is the example of a function that returns a value

```
var greet = function (person, greeting)
{
var writing = greeting + ' , ' + person ;
return function () {console.log(text); };
}
console.log(greet('Richard' ,' Hello"));
```

iii) Sometimes, you might want to use a nested function. (Note: A nested function is a function within another function.)

```
varsayHi = function( person, greeting) {
var text = greeting + ',' + person;
return function() {console.log (text);}
};
var greeting = greet( ' Richard' , 'Hello');
greeting();
```

Self-Executing Anonymous Function

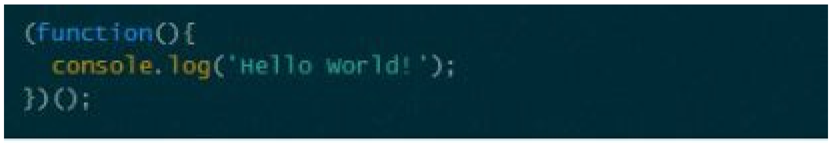
Programmers have always been looking for the most advanced and clever methods available to improve their programming experience and ways to make it more accessible and easier for them. This resulted in the creation of the**Self-**

Executing Anonymous function.

The core purpose of the self-executing anonymous function is to create a JavaScript function and then immediately execute it upon its conception.

This makes it much easier for a programmer who isworking on large scale programs. Likewise, it will help you to code without creating a messy global namespace.

A basic Self-Executing Anonymous Function Would be-



```
(function(){
  console.log('Hello world!');
})();
```

It is very important that you understand how to use this type of function if you plan on programming complex code in JavaScript. Using the Self-Executing Anonymous Function can produce some great code that is easy to use. As an example, the JQuery library is designed to make using JavaScript on websites much easier. It does this by wrapping the entire library in one large self-executing function.

Typeof Operator

You might run into a situation where you need to determine the type of variable that you are working in JavaScript. You won't have call Sherlock Holmes to find that! Instead, you can simply use the 'typeof' operator to determine the type of any specific value. In other words, the typeof operator is used to evaluate the type of the operand.

EX-

Varmyvar=0

alert(typeofmyvar) //alerts "number"

Number isn't the only type of operand that can be detected. It can also be: string, Boolean, object, null, and not defined.

Scope

Scope is the accessibility of a variable.

A good understanding of Scope is necessary when it comes to debugging because it allow you to know whatvariable from which code block is causing the problem.

The simple rule here is that whenever you declare a variable inside a scope, it will only be recognized by the statements that are inside that scope; the statements that are outside the scope will not acknowledge its existence and so that variable will not work.

Another way to look at this is to imagine your entire code as a hotel with specific functions and sections of code as hotel rooms. The hotel rooms represent the private scopes of the code, while the common areas represent the global scope. A person in one hotel room cannot see or use what is in another hotel room. Staff who work in the hotel (and the global scope) also don't have access to private hotel rooms unless they have specific permission. Meanwhile, guests can go through the common areas and make use of any object there.

Thus, you can see how scope can affect how your code runs. If you need two different functions to access the same variable, you need to ensure that they both have access to it. One key way to ensure that the necessary variables have access to is to make your variables globally accessible.

There are two possible alternatives if you want your variables to become globally accessible.

The first thing you can do is to declare the variable outside the scope of your given piece of code, this will allow any functions in your program to be able to call it and recognize it.

The other thing you can do is to declare the variables inside your scope without using the word var. If the same variable was not defined at the beginning of the code outside the scope of the piece of code in question, then the variable will act similarly to a global one.

Ex-

```
var foo = ‘hello’;

var talkHello = function() {

console.log(foo);

};

talkHello(); // logs ‘hello’
```

```
console.log(foo); // also logs ‘hello’
```

As you can see, the variable foo is declared outside of the function talkHello. That means that foo is a global variable and any function will be able to access it, so when the function talkHello calls it, it is accessible.

The following example is contradictory to the first example. This shows that a code block that was written outside the scope is not being able to recognize the variable.

```
var talkHello = function() {
var doo = ‘hello’;
console.log(doo);
};
talkHello(); // logs ‘hello’
console.log(doo); // gives an empty log.
```

In this example, the variable doo is called inside the function talkHello. This creates a private variable that is only accessible to the function, hence when the command console.log(doo) attempts to access it, it returns a null value.

Some Common Mistakes to Avoid

Always remember that a variable is only accessible by the functions within a specific scope, outside of that scope the variable is invalid.

Practice

Explain what a scope is and how the variables are affected by it.

Write a simple program to illustrate the functionalities of a:-

a) Simple Function

b) A Function with returnable value

c) A function passed as a parameter (Argument)

Explain the concept of a Self-Executing Anonymous Function and give an example.

Chapter Five

Some prominent Features of JavaScript

Many new programmers often get a little bit disappointed when they hear people call JavaScript a language for kids. Well, let me clear this up for you. First, JavaScript is hands down, one of the strongest programming languages out there. It's a popular skill required for web development.

The beauty of JavaScript lies in its simple yet complex presentation. It has been used for many years now by numerous developers. It can add a fresh spin on their web pages, ranging from very subtle but important features such as automatic image changers, to more intricate designs that enhance their beauty.

The following are just a few of the features that help JavaScript to stand out on its own in the crowd.

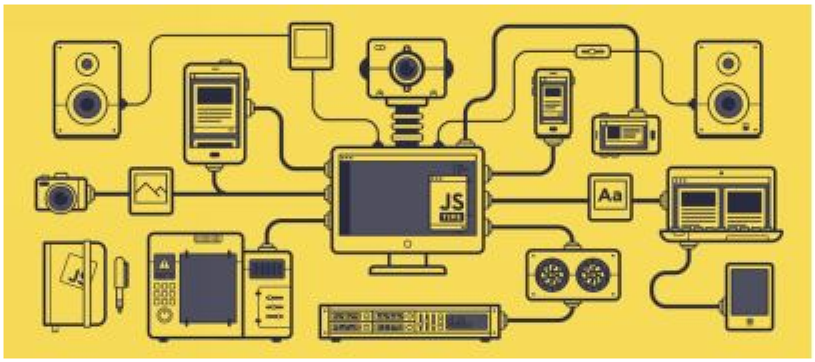
Integrated Browser Support

Right off the bat, the user friendly nature of JavaScript is very intriguing and it attracts many new programmers and experts. Many web development languages out there require installing a supplementary flash plugin in order to be allowed to access flash content and work on the browser. The users of JavaScript are mostly spared from this hassle as almost all the browsers have invariably accepted JavaScript as their core scripting language. This allows them to deliver integrated support for the language. In the most critical cases, the users will only need to handle some particular tasks that rely on DOM with care, but other than that, JavaScript is fully accessible and free to be used.

A Functional and Flexible Programming Language

For programmers who prefer a lot of functions in their programs, JavaScript isa joy. This is because of the versatile manipulation techniques incorporated within the language that allows the programmers to determine the execution of a function. Two of the more prominent features of Java that put it at the frontend offunctional programming is that it givesprogrammers the freedom to assign a function to any variable, and make a function accept another function using the arguments parameters. Such in-depth ability for customization makes it so much more fun for programmers who like to have more freedom in programming.

The ability to use JavaScript on both Client and Server Side



Since JavaScript has been officially recognized by the web-browsers as being the prime development language for them, JavaScript has been given the administrative privileges to have access to the document model objects of any browser. What this does is that it gives the programmer another level of freedom. It allows programmers to change the structure of the web pages on the fly. Not only that, the language can also be used to add various interesting effects to the web pages. And if you are working with Alfresco, then you are in luck! Because JavaScript also allows web scripts to be created in Alfresco. This allows programmers to add custom tasks to any Alfresco server very easily.

When you combine all of these together, JavaScript is truly a magnificent programming languagethat is powerful yet easy to use.

Self-Conscious ability to detect browser and OS

Sometimes when you are programming, you might face some issues and need to write code that is dependent on the operating system and web browser that you are using.

JavaScript has been intelligently designed to allow itself to be self-aware of the browser and OS which you are using. This allows you to easily adjust the language in accordance with your requirements to perform all the OS dependent actions with ease.

The Diverse Support for Object Oriented Programming

Object oriented programming is perhaps the keystone of programming languages these days. Programmers often like to choose their language depending on the accessibility to the handling mechanism of these objects. When brought in to comparison, JavaScript has a very different and unique structural system that determines how to handle the objects. It offers a huge amount of support, while at the same time it remains relatively easy to learn and use. This encourages the users to write more complex object oriented code smoothly and efficiently with very little complication.

This is one of the many reasons why JavaScript always remains on the top of the list for programmers and also in the industry.

If you are interested in object oriented programming, then you can't go wrong with JavaScript.

Conclusion

A lot of people believe that the only way to learn proper programming is by going back to school or other education centers. But l don't believe it's true.Thanks to the tremendous advancements in online communication, you can easily learn and master any new skill through internet tutorials and videos. The information available out there for you to grasp is endless.

JavaScript is not a program that is only for geniuses and geeks. This is a very standardized language and it can easily be picked up by anyone. Just like languages in real life, this language will also help you to open unlimited possibilities and opportunities.

Thank you for purchasing this book and giving me your precious time. I hope that this book has helped you, and I wish that going through this book, you have obtained a basic knowledge of all the elementary prospects of JavaScript programming which will help you tokick start your career as a programmer.

I encourage you to practice JS programming as much as you can. Soon, you will become the programmer you want to be as long as you dedicate yourself. Let the flame of ambition burn inside you. Remember, if you really want to do this, don’t jump over to another programming language before you master this one.

This is just the beginning of your journey. And I hope you enjoy the process before you arrive at your destination.