# JAVASCRIPT

STEP BY STEP
BEGINNER'S
GUIDE


## PROGRAMMING BASICS FOR ABSOLUTE BEGINNERS

NATHAN CLARK

# JavaScript

*Programming Basics for Absolute Beginners*

*Nathan Clark*

# Table of Contents

This document is presented with the desire to provide reliable, quality information about the topic in question and the facts discussed within. This eBook is sold under the assumption that neither the author nor the publisher should be asked to provide the services discussed within. If any discussion, professional or legal, is otherwise required a proper professional should be consulted.

This Declaration was held acceptable and equally approved by the Committee of Publishers and Associations as well as the American Bar Association.



Any information contained in the following pages is considered accurate and truthful and that any liability through inattention or by any use or misuse of the topics discussed within falls solely on the reader. There are no cases in which the Publisher of this work can be held responsible or be asked to provide reparations for any loss of monetary gain or other damages which may be caused by following the presented information in any way shape or form.

The following information is presented purely for informative purposes and is therefore considered universal. The information presented within is done so without a contract or any other type of assurance as to its quality or validity.

Any trademarks which are used are done so without consent and any use of the same does not imply consent or permission was gained from the owner. Any trademarks or brands found within are purely used for clarification purposes and no owners are in anyway affiliated with this work.

# Introduction

Congratulations on purchasing *JavaScript: Programming Basics for Absolute Beginners* and thank you for doing so.

The following chapters will discuss how you are going be able to use JavaScript to your advantage. By the end of this book, you are going to know how to use JavaScript despite the fact that you have never used it before. You will not be able to do everything but you are going to know the basics.

There are plenty of books on this subject on the market, thanks again for choosing this one! Every effort was made to ensure it is full of as much useful information as possible, please enjoy!

# What is JavaScript?

One of the languages that is used in programming is JavaScript. Many times it is used to design webpages so they have the ability to be interactive with users. The user's computer is going to run JavaScript without any content needing to be downloaded. A lot of the time, JavaScript can be used in creating quizzes and polls.

As a high-level program, JavaScript is dynamic, untyped, and interpreted. Out of all the content that you typically see online, it is created with JavaScript or two other technologies. Many major web browsers, email servers, and website run on JavaScript code.

JavaScript contains first class functions that are a prototype which makes it a multi-paradigm language with the ability to support programming styles that are object oriented. The API is used for when you are working with text, regular expressions, and dates. But, this is not going to include any input or output when it comes to storage or networking.

JavaScript is not Java! There are going to be similarities between the two programs, but they are two very different programs that are going to influence how a program works in different ways.

JavaScript is not limited to creating things that go online. You are also going to have the ability create PDF files, browsers that are site specific, and widgets that can be used on your desktop. Thanks to the new JavaScript platforms, the new virtual machines that you see are faster. Because of the new virtual machines, JavaScript has increased in popularity on the server site.

On the client side of things, JavaScript is going to be an interpreted language similar to Python. The browsers that have come out recently are going to perform on JavaScript as a just in time compilation.

Taking JavaScript a step further, it can be used to make mobile applications, desktops, and games. You are also going to be able to create a network that works side by side with the server to use runtime environments.

# Turning JavaScript On and Off

JavaScript is supported by almost every browser. There is no way to not have JavaScript installed on your browser, but you can turn it off if you are worried about people using it to abuse its privileges on your computer. JavaScript will be based on the web pages that you are using. It is also determining if what you input into the website is important or not.

## *Internet Explorer 7 & 8*

- Go to tools
- Select internet options
- Click on security
- Click on the internet symbol
- Click on custom level
- Under settings go to scripting
- Select the radio button next to JavaScript to determine if you want it on or off
- Close everything out and your settings will be saved

## *Internet Explorer 6*

- Open tools
- Go to internet options
- Select security
- Click on the globe
- Go to custom
- Move to scripting
- Select if you want JavaScript enabled or not
- Close out all the dialog boxes

## *Firefox*

- Open tools
- Select preferences
- Select content
- Enable or disable JavaScript
- Close the boxes

## *Chrome*

- Locate the spanner icon on the top of the menu
- Go to options
- In options go to under the bonnet
- Click on content settings
- Open privacy settings
- Enable or disable JavaScript
- You will have the option of allowing all sites to run JavaScript or no sites
- Close content settings
- Close options

## *Opera*

- Open tools
- Hover over quick preferences
- In the submenu, click enable or disable

## *Safari*

- Open the safari menu
- Click on preferences
- Open the security icon
- Go to web content so you can enable or disable JavaScript

## *Camino*

- Open Camino menu
- Click on preferences
- Select web features
- In control content, you will be able to enable or disable JavaScript

## *Sea Monkey*

- Open the edit menu
- Select preferences
- Go to the plus side under advanced
- In scripts and plug-ins, you will be able to enable or disable JavaScript

# JavaScript Basics

JavaScript is not actually HTML code. As you are creating JavaScript code, you will be required to allow your browser to understand that you are inputting JavaScript to the HTML. JavaScript function is <script>. The function <script> type = “text/javascript”> <script> is going to allow the page to know exactly where the JavaScript coding is going to start and where ends.

**Example**

```
<html>
<head>
<title> My JavaScript Page </title>
</head>
<body>
<script type = “text/javascript”>
Alert (“Welcome to my world!!!”);
</script>
</body>
</html>
```

The word alert is going to cause an alert box to pop up on your screen. To make it go away so that you can continue with what you are doing, you are going to have to click “ok”. The alert command will be entered when the browser needs to recognize the code as JavaScript. Whenever you do not put <script> in the coding, the browser will think that what you have entered is regular text. You can put JavaScript in both the head and the body of a document that is written in HTML. You should try to keep as much of your code in the head section.

## *First Script*

Before you are able to make your first program using JavaScript, you have to know more things than where you need to put your code. JavaScript lines have to end with semicolons. All code can be placed in one line without the worry of destroying how it performs. Even though it is not going to affect how the code performs, it is going to destroy how your script looks. Your text should be inside of a set of quotations. When you forget to put text in quotations, it is going to be considered variables. Capitals are going to mean something different than lowercase letters. Capitals should only be placed in their correct places so that your script is not messed up.

**Example**

```
<html>
<head>
<title> My JavaScript Page </title>
</head>
<body>
<script type = “text/javascript”>
</script>
Document.write (“Welcome to my world!!!”);
</script>
</body>
</html>
```

Document.write will tell the users to place the contents in the parentheses into the document. You will have an output similar to the Python test file of Hello World!

**Example**

```
<html>
<head>
```

<title> My JavaScript Page </title>

</head>

<body>

Hello!!! <br>

<script>

Document.write (“Welcome to my world!!!”);

</script>

Enjoy your stay…<br>

</body>

</html>

Output:

Hello!!!

Welcome to my world!!!

Enjoy your stay…

Text will be written where it has been placed in the HTML code. Many HTML tags are written with document.write.

## *Capitals*

In JavaScript, yourname  and YOURNAME are two different variables.

**Example**

```
<html>
<head>
<title> My Page </title>
</head>
<body>
<script>
myvalue =2;
 myvalue =5;
result = myvalue + myvalue;
document.write(result) ;
</script>
</body>
</html>
```

**Example**

```
<html>
<head>
<title> My Page </title>
</head>
<body>
<script>
myvalue =2;
 MyValue =5;
```

```
result = myvalue + MyValue;
document.write(result) ;
</script>
</body>
</html>
```

Both have different variables, therefore they will have different outputs. Do not mix and match syntax. Which syntax you use is not as important as you sticking with it all through your code.

## *Pop-ups*

You will get three pop-up boxes when using JavaScript. The alert box, confirm box, and the prompt box.

### Alert

The syntax is going to alert to something that is in the text that you entered. You need to click “ok” to continue. You are going to be notified of information you have to know.

### Confirm

This box is to confirm your text. You will need to click “ok” or “cancel”. You are going to be forced to accept something. When “ok” is clicked, the value is true. When “cancel” is clicked, the value will be found as false.

**Example**

If (confirm(“Do you agree”)) {alert)”You agree”)}

Else(alert (“You do not agree”)) ;

### Prompt

This box is going to prompt your text vs the default value. You will have to click “ok” or “cancel”. Before you can continue on the page, you will have to input something. “okay” is going to return the entry. “cancel” is going to say that the return was empty.

**Example**

Username = prompt (“Please enter your name” , “Enter your name here”) ;

# Syntax of JavaScript

Syntax is a basic set of rules on how JavaScript code is constructed.

## *Programs*

The program is going to have a set list of instructions as to what needs to be executed by the computer program so that the program or website works correctly. These instructions are considered to be statements. These statements are separated by semicolons.

**Example**

```
Var x = 5;

Var y = 7;

Var z = x + y;
```

JavaScript programs are going to be executed inside of a web browser.

## *Statements*

There are five components of a statement in JavaScript: Keywords, Values, Operators, Comments and expressions.

## *Values*

There are two different value types. Fixed and variable. Fixed values are also known as literals. Variable values are nothing more than regular variables.

## *Literals*

There are rules when writing fixed variables. You will need to write numbers with the appropriate decimal points.

**Example**

10.50

1001

Strings are going to be inside of quotes no matter if it is double quotes or single quotes.

**Example**

“Jane Doe”

‘Jane Doe’

## *Variables*

Stored data from values is the definition of a variable. Var is going to tell JavaScript that it is a variable. The equals sign is going to tag a value to the variable.

**Example**

```
Var x;

X = 6;
```

## *Operators*

Assignment operators are used to assign values to variables. The assignment operator is going to be the equals sign.

**Example**

Var x = 8

Var y = 9

You can also use arithmetic operators like addition, subtraction, multiplication, and division.

## *Expressions*

The combination of operators, variables, and values will make up an expression. Operators, values, and variables will compute to equal some sort of value. The process of computing the value is called an evaluation.

**Example**

10 * 20

The output will be 200. The expression can also have a variable value.

**Example**

x * 5

Values will be different types of data like numbers or strings.

**Example**

“Jane” + “ “ + “Doe”

The output is Jane Doe

## *Keywords*

Keywords identify any action that needs to be performed. Var is going to tell the browser to make a new variable.

**Example**

Var x = 6 + 8;

Var y = x * 15;

## *Comments*

All comments will not be executed. Double dashes after a code or between is going to be a comment. Comments are ignored because they are there to help programmers understand the code that has been written.

**Examples**

Var x = 9; //I am going to be executed in order to give you an output.

// var x = 7; //this is not going to be executed because it has a double dash before the code, therefore making it to where it is just a comment.

## *Identifier*

Identifiers are names for different things that you are going to find in JavaScript. Identifiers name the various variables, functions, labels and keywords. Legal names have the same rules as any other programming language. First characters will have to be dollar signs, underscore, or letter. Characters that come after can be any sign, number, or letter. Numbers are not allowed to be the first letter in an identifier because JavaScript will think it is a number, not a name.

## *Case Sensitive*

Identifiers are very case sensitive. Variables like middleName and middlename are going to be different variables.

**Example**

lastName = “Mann”;

lastname = “Johnson”;

However, var and VAR are the same.

## *Joining Variables*

There are 3 ways that words are usually joined to make a single variable:

**Hyphen**

Hyphens, such as My-name-is, are not allowed in JavaScript. They are reserved for subtraction only.

**Underscore**

My_name_is

**Camel case**

There will be two or more capital letters, such as myNameIs. Think of the humps on a camel. Most programmers leave the first letter lowercased.

## *Character Set*

JavaScript contains a Unicode set of characters. There are going to be a majority of characters, punctuation, and symbols that will be covered with Unicode.

# The Variables of JavaScript

Variables are boxes with a name tied to it. Variables store values. Your box's name is going to be the name of the variable. The contents of the box will pertain to the variable. It is much like the memory of a computer where information is put on the memory like the contents are put on the variables. Variables will refer to the name you give it.

**Example**

```
<html>
<head>
<title> My Javascript Page </title>
</head>

<body>
<script>
Myname = "Susann" ;
Document.write(myname) ;
</script>
</body>
</html>
```

The text that you want to be stored on a variable will need to be in a set of quotes. This is going to allow the program to know the difference between a variable and normal text.

**Example**

```
<html>
<head>
<title> My Javascript Page </title>
```

```
</head>
<body>
<script>
Susann = “my first name” ;
Myname = Susann ;
Document.write (myname) ;
</script>
</body>
</html>
```

The output is:

- Susan is the variable
- It is located in the myname variable
- The myname is going to be placed in the document
- The end result is my first name

## *Values Assigned to Variables*

The most common way for a value to be assigned to a variable is going to be an equals sign.

## *Operators*

Operators like a++ and a - - will be ones that you are most likely not going to want to use. You can use other operators to do the exact same job as these operators. But, they are going to make typing out your code faster, so many programmers use them so that their coding goes quicker. The operator ++ will increase by an increment of one. - - will decrease by an increment of one. The percentage sign is going to return the modulus of the remainder of two numbers that have been divided.

## *Variables Compared*

There are multiple ways you can compare variables. The simplest way is double equals sign. When you forget do not use double equals, you are not going to be comparing variables. The variable that is listed on the left is going to be assigned to the value that is listed on the right.

**Example**

If (firstname = “Susann”) {alert(“Nice name!!!”)}

This is common, but it will cause your script to run differently. You can use operators that you learned when you were in elementary school in JavaScript:

- == equaled to
- != not equaled to
- < less than
- > greater than
- <= equal to or less than
- >= equal to or greater than

# Functions in JavaScript

Functions in JavaScript will be blocked in a code that is designed to operate a specific task. Functions will be used to execute whenever it has been invoked or called upon.

**Example**

```
FunctionmyFunction(p1, p2) {

Return p1 * p2;

// you function is going to return the product of p1 and p2

}
```

## *Function Syntax*

The function will be defined by a keyword function. The name is followed by a set of parentheses. Names will have digits, underscores, letters, and dollar signs. They follow the same rules variables use. The parentheses will have parameter names that are separated by the use of a comma. All the code that is to be performed will be in curly brackets.

**Example**

```
Functionname(parameter1, parameter2, parameter3) {

Code to be executed

}
```

Parameters will be names already in the definition function. Arguments are going to contain real values that the function has accepted when it has been invoked. The arguments will behave like local variables.

## *Invocation*

When a code is put into a function, it will be used after it has been invoked. The code is invoked when:

- It is invoked from the script
- An event occurs like when a button has been clicked.
- It is self-invoked

## *Returns*

The code will come to a statement that has been returned which will mean that it is going to stop functioning. The code is going to be invoked when the statement is returned to an executed code that comes after a statement has been invoked. The function will evaluate a returned value. The returned value is going to be returned to the original caller.

**Example**

Var x = myFunction (4, 3); //this function has been called upon and is going to be returned with the value being x.

Function myFunction(a, b) {

Return a * b; //function returns the product of a times be

}

X is going to be 12 which is going to be your output.

## *Why Functions?*

Functions are going to enable codes to be reused. Codes will be defined once and then it can be used multiple times over. The same code will be used with different arguments to get different results.

**Example**

```
Function toCelsius (Fahrenheit) {

Return ( 5/9) * (Fahrenheit -32);

}

Document.getElementById(“demo”).innerHTML = toCelsius(77);
```

## *Parentheses Operator*

To access a function without using parentheses, you are going to get a definition returned.

**Example**

```
Function toCelsius( Fahrenheit) {
Return ( 5/9) * (Fahrenheit – 32);
}
Document.getElementById(“demo”). innerHTML = toCelsius;
```

## *Variable Values in Functions*

Functions can be used like a variable is used. They are going to be used in formulas, assignments, and calculations. You do not have to use variables when you can put the value that is returned in a function.

**Example**

Var x = toCelsius(77);

Var text = “The temperature is “ + x + “ Celsius”;

Functions can be used direction as the variable’s value

**Example**

Var text = “The temperature is “ + toCelsius(77) + “ Celsius”;

# JavaScript Decision Making

There are different statements that you use for decision-making.

## *If Statement*

The statement will check a condition to see if it is true or false. The condition will be an expression with a true or false return. The statement that satisfies the expression is going to be executed.

**Syntax**

If(condition)

{

Statement 3

Statement 4

…

}

If one statement has needs be carried out after the condition, you will not use the curly brackets. If there are multiple statements, the brackets will need to be used.

**Syntax**

If(condition)

Statement

It is recommended that curly brackets be used so that your code is easier to manage and clear so that it can be understood.

**Example**

<script>

If (6 > 2)

{

Document.write( “yes 6 is greater than 2”);

Document.write “<br />” + “JavaScript is easy”);

Document.write( “<br />”);

}

Output:

Yes 6 is greater than 2.

JavaScript is easy.

**Example**

If(true)

Document.write (“Ah! A Boolean inside condition. Also, I am not using curly brackets”);

// it does work without having to use curly brackets

Output:

Ah! A Boolean inside condition. Also, I am not using curly brackets.

**Example**

If ( 2 == 4)

{

Document.write (“This is not going to be printed”);

}

//since the condition has turned out to be false, the statement is not going to be executed.

</script>

## *Else Statement*

This statement will be used with the if statement. Whenever the condition is listed as false, the else statement is going to be carried out.

**Syntax**

```
If(condition)
{
Statements
}
Else
{
Statements
}
```

Curly brackets are dropped when an else statement is used. This is only going to happen when a single statement is needing to be executed.

**Example**

```
<script>
If 5 > 7
{
Document.write(“True”);
}
Else
{
Document.write(“False”);
}
```

The output is going to be false.

## *Else If Statement*

There are a variety of conditions that have to be checked, then there are going to be a multitude of if statements that have to be used. The if conditions will be checked at the same time. If one condition fails, then the check will be stopped. This is when you will use an else if statement.

**Syntax**

```
If(condition)
{
Statements
}
Else if (condition)
{
Statements
}
Else
{
Statements
```

When all the conditions fail, the ending else statement will be carried out.

**Example**

```
<script>
Var a = 3;
Var b = 4;
If (a > b)
{
Document.write(“a is greater than b”);
}
```

```
Else if (a < b)
{
Document.write(“ a is smaller than b”);
}
Else
{
Document.write(“Nothing worked”);
}
</script>
```

## *Switches*

Switch statements will do the same thing as the else if statement. The switch statement will be used when there are multiple conditions. Switch statements are used to perform better than an else if statement.

**Syntax**

```
Switch(expression)
{
Case value -1: statements; break;
Case value -2: statements; break;
Case value 3: statements; break;
….
Case value-n: statements; break;
Default: statements; break;
}
```

A switch statement will assess the statement to check and see if it matches any cases. If it matches, then the statement that is inside the case will be executed before being followed by a break statement. Break statements make it to where no other case statements are executed.

If an expression cannot be matched to a case value, then the default case is going to be the one executed. It does not matter what kind of statement is in the case. Curly brackets do not have to be used. Break statements will be omitted if the default construction is the last statement that is found in the switch.

**Example**

```
<script>
Var a = 8
Switch(a)
{
```

```
Case 1: document.write(“one”); break;
Case 2: document.write( “two”); break;
Case 3: document.write( “three”); break;
Case 4: document.write( “four”); break;
Case 5: document.write( “five”); break;
Default: document.write(“number not found”);
}
// outputs five
</script>
```

Case values are able to be strings, characters, and numbers.

**Example**

```
<script>
Var I = ‘j’;
Switch(i)
{
Case ‘a’: document.write(“a found”); break;
Case ‘b’: document.write(‘b found”); break;
Case ‘c’: document.write(“c found”); break;
Case ‘j’: document.write(“j found”); break;
Case ‘string’: document.write(“string found”); break;
Default: document.write(“nothing found”);
}
//outputs j found
</script>
```

## *Ternary*

Ternary operators are the operators that many programmers use. There are three operands for a ternary operator.

**Syntax**

Condition ? if-true-execute-this : if-false-execute-this;

If the condition is right, then the statement will be before a colon in order to be executed. If the statement is returned false, the statement after the colon will be executed. A variety of statements cannot be used because JavaScript will not allow it. Multiples statements can be executed by using functions and naming that function should the condition be found true or false.

**Example**

<script>

True ? document.write ("True value found") : document.write("False value found");

Document.write("<br />");

// outputs True value found.

( 6 > 5 && 3 == 2) ? document.write("True") : document.write("False");

Document.write(" <br />");

// outputs False

Var a = (true) ? 1 : 2;

Document.write(a);

//outputs 1

</script>

## *If Else and Switch Statements That Are Nested*

You can put an if, or, else or switch statement in another condition. You are going to come to know this as nesting. Else, if, or switch statements can be nested to any level. But the code is going to be more confusing than it was before.

**Syntax**

```
If(condition)
{
If (other condition)
{
Statements;
}
}
Else
{
If (other condition)
{
Statements;
}
Else
{
Switch(expression)
{
Case value-1: statements; break;
…
}
}
```

}

# Control Flow Statements in JavaScript

When you find a source file, the narratives in it are going to execute in the sequence that they are listed much like how you read a book. Flow statements are going to deal with how the operators flow through branching, looping, decision-making, and enabling the program to operate correctly because of specific chunks of code that are based on certain conditions.

## *If-Then*

The if then statement will be the simplest statement in a control statement. If then statements are going to perform a code segment only if it evaluates as true.

**Example**

Driver's Ed class will only enable the brakes to decrease the car's speed only if the car is already in motion.

```
Void applyBrakes() {
// the "if" clause: car must be moving
If (isMoving) {
// the "then" clause: decrease current speed
currentSpeed -- ;
}
}
```

If the condition is false, the control will move to the following if then statement. Open and closed curly brackets are optional if the clause found in them turns out to be the only statement.

**Example**

```
Void applyBreaks() {
// same as above example, only the curly brackets are not going to be here.
If (isMoving)
CurrentSpeed -- ;
}
```

If you decide to omit the curly brackets, this is going to be your choice. But, if you omit them, you are making your code fragile. When a second statement is later, adding a then clause will cause you to possibly forget the curly brackets. The compiler will not catch the mistake and you are going to get improper

results.

## *If-Then-Else*

The if-then-else statement will add a secondary path for the code to use if the if clause fails.

**Example**

```
Void applyBreakes() {
If (isMoving) {
CurrentSpeed -- ;
} else {
System.err.printin (“The car is not moving!”)
}
}
```

## *Switches*

Switch statements have a variety of paths of execution. Switch statements work with byte, char, short, and int data types as well as enumerated types.

**Example**

```
Public class SwitchDemo {

Publicstatic void main (String[] args) {

Int month = 8 ;

String monthString ;

Switch (month) {

Case 1: monthString = “January” ;

Break ;

Case 2: monthString = “February” ;

Break ;

Case 3: monthString = “March” ;

Break ;

Case 4: monthString = “April” ;

Break ;

Case 5: monthString = “May” ;

Break ;

Case 6: monthString = “June” ;

Break ;

Case 7: monthString = “July” ;

Break ;

Case 8: monthString = “August” ;

Break ;

Case 9: monthString = “September” ;
```

```
Break ;
Case 10: monthString = “October” ;
Break ;
Case 11: monthString = “November” ;
Break ;
Case 12: monthString = “December” ;
Break ;
Default: monthString = “Invalid month” ;
Break ;
}
System.out.println(monthString) ;
}
}
```

The main switch statement part is a switch blog. The comments that fall in the block switch will point out multiple cases or default labels. Switch statements evaluate expressions before they are executed. All the statements that succeed are going to be labeled with identical tags. Month’s names are going to be used with an if-then-else statement.

**Example**

```
Int month = 8
If (month == 1) {
System.out.println (“January”) ;
}else if (month ==2) {
System.out.println (“February”) ;
}
```

… so on and so forth.

An if-then-else statement will be used if the clarity of the expression needs to be examined. The if-then-else statement will test an expression if the conditions and values are needing to be tested based on integers, enumerated values, and string values. Break statements will end the switch statements.

Control flow will continue with the comment that follows the switch block. Break statements are used if the switch block falls through, therefore all statements are going have identical labels to be executed in the order that they appear no matter what case labels there are or until the statement that is listed as a break has been reached.

**Example**

```
Public class SwitchDemoFallThrough {

Public static void main (String[] args) {

Java.util.ArrayList<String> futureMonths =

New java.util.ArrayList<String> () ;

Int month = 8 ;

Switch (month) {

Case 1: futureMonths.add (“January”) ;

Case 2: futureMonths.add (“February”) ;

Case 3: futureMonths.add (“March”) ;

Case 4: futureMonths.add (“April) ;

Case 5: futureMonths.add (“May”) ;

Case 6: futureMonths.add (“June”) ;

Case 7: futureMonths.add (“July”) ;

Case 8: futureMonths.add (“August”) ;

Case 9: futureMonths.add (“September”) ;

Case 10: futureMonths.add (“October”) ;

Case 11: futureMonths.add (“November”) ;

Case 12: futureMonths.add (“December”) ;
```

```
Break ;
Break ;
Default: break ;
}
If (futureMonths.isEmpty()) {
System.out.println (“Invalid month number”) ;
} else {
For (String monthName : futureMonths) {
System.out.println (monthName) ;
}
}
}
}
```

The output is: August – December. The ending break did not need to be placed in the code being that the flower vanished from the switch statement. A break is going to be recommended because it will be simpler to modify the code and you are reducing your chances of an error code. The default section of code will deal with values that are not in the case section. Code has the ability of having multiple cases.

**Example**

```
Class SwitchDemo2 {
Public static void main (String[] args) {
Int month = 2;
Int year = 2016;
Int numDays = 0
Switch (month) {
Case 1: case 3: case 5:
```

```
Case 7: case 8: case 10:
Case 12
numDays = 28
break ;
case 4: case 6:
case 9: case 11:
numDays = 31;
break;
case 2:
if ((year % 4 == 0) &&
!(year % 100 == 0))
|| (year % 400 == 0))
NumDays = 31;
Else
numDays= 30;
break;
default:
system.out.println (“Invalid month.”) ;
break ;
}
System.out.println (“Number of Days = “ + numDays) ;
}
}
```

The output is going to be 29 since there are 29 days in February.

## *Switch Statement Strings*

String objects have places in switch statements. At the point in time a string is placed in a switch, the expression will be compared to another expression with associating case labels. Strings can be accepted in a switch if the case is lowercase and the case labels are lowercase.

**Example**

```
Public class StringSwitchDemo {

Public static int getMonthNumber(String month) {

Int monthNumber = 0

If (month == null) {

Return monthNumber ;

}

Switch (month.toLowerCase()) {

Case “January”:

monthNumber = 1

break ;
```

This will be continued until all 12 months have been written out in lower cased with the number that is associated with it.

## *While and Do While Statements*

While statements work on blocks of statements continually until the condition has been met as true.

**Syntax**

```
While (expression) {
Statement(s)
}
```

While statements will evaluate expressions, and return a Boolean statement. When the condition is found to be true, then the statement in the while block is going to be executed. The while statement will test the statement and execute it until it is proven false.

**Example**

```
Class WhileDemo {
Public static void main(String[] args) {
Int count = 1 ;
While (count < 11) {
System.out.println(“ Count is: “ + count) ;
Count++;
}
}
}
```

Infinite loops are implemented in the while statement with this syntax.

**Example**

```
While (true){
// your code goes here
```

}

Do while statements are part of JavaScript as well.

**Example**

```
Do {
Statement(s)
} while (expression) ;
```

The difference is to evaluate the equation at the base of the loop rather than the one located at the top. Every piece in the loop will be evaluated at least once.

**Example**

```
Class DoWhileDemo {
Public static void main(String[] args) {
Int count = 1
Do {
System.out.println ("Count is: " + count) ;
Count++;
} while (count < 11) ;
}
}
```

## *For Statement*

The statement will provide a compact way to iterate the values inside of a range. This is known as a for loop since it loops until the condition that is set in place is found to be satisfied.

**Syntax**

```
For (initialization; termination;

Increment) {

Statement(s)

}
```

You should remember that increment expressions are invoked after iteration of the loop. It is okay for this expression to increase or decrease for a value. The initialization expression is going to initialize the loop and execute it once the loop starts. Termination will evaluate as false when the loop stops.

**Example**

```
Class ForDemo {

Public static void main(String[] args) {

For (int i=l ; i<11; i++){

System.out.println (“Count is: “ + i);

}

}

}
```

The output is 1-10.

The code will state the variable in the initialization expression. The variable scope will extend from the declaration to the end of the block. The for statement will be used for the termination of an expression too. If the variable

is a for statement, it is not going to be used outside of the loop. It will declare the variable in the initialization process. The names I, k, and j are used for loop controls. After being declared in the initialization expression, the lifespan will be cut short and the number of errors reduced. There are three expressions that are used in a “for” loop.

**Example**

```
// infinite loop
For ( ; ;) {
// your code goes here
}
```

A for statement will contain another variation that is going to be used with iteration. The variation is known as an enhancement for the statement and will be used to make a loop smaller thus making it easier to read.

**Example**

```
Int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
Class EnhancedForDemo {
Public static void main (String[] args){
Int[] numbers =
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
For (int item : numbers) {
System.out.print;n (“ Count is : “ + item) ;
}
}
}
```

Variables are going to hold a value in the example above so that your output is 1-10 once again. This variation will be used when writing out for statements for the general way that a for statement is written.

## *Break Statement*

There are two variations of break statements. The unlabeled and the labeled. The unlabeled is used with switch statements. Unlabeled breaks are going to be able to terminate for, while, and do-while loops.

**Example**

```
Class BreakDemo {
Public static void main(String[] args) {
Int[] arrayofInts =
{32, 87, 3, 589,
12, 1076, 2000,
8, 622, 127};
Int searchfor = 12;
Int I;
Boolean foundIt = false;
For (I = 0; I < arrayOfInts . length; i++) {
If (arrayOfInts [i] == searchfor) {
foundIt = true;
break;
}
}
If (foundIt) {
System.out.println("found " + searchfor + " at index " + i);
} else {
System.out.println (search for + " not in the array");
}
}
}
```

Break statements will have bold print where the loop is terminated after the value that needs to be found is reached. The flow from the control is going to be transferred to the loop for loops. The output is going to be 12 found at index 4. Breaks that are unlabeled are going to terminate the inner parts of a switch, do-while, for, and while statement.

Labeled breaks are going to stop the statements on the outside. The example will use a nested loop to locate values on the 2-d array. Once the value has been located, the labeled break is going to terminate the for loop.

**Example**

```
Class BreakWithLabelDemo {

Public static void main(String[] args) {

Int[] []arrayOfInts = {

{ 32, 87, 3, 589 },

{ 12, 1076, 2000, 8 },

{ 622, 127, 77, 955}

};

Int searchfor = 12

Int I;

Int j = 0

Boolean foundIt = false;

Search:

For (I =0; I < arrayOfInts.length; i++) {

For (j = 0; j < arrayOfInts[i].length;

J++) {

If (arrayOfInts[i][j] == searchfor) {

foundIt = true;

break search;
```

```
}
}
}
If (foundIt) {
System.out.println(“Found “ + searchfor + “ at “ + I + “, “ +j);
} else {
System.outprintln(searchfor + “ not in the array”);
}
}
}
```

The output is 12 found at 1.

The labeled statement is stopped by the break statements, and the control flow will not be transferred to the labeled variable. The flow will be transferred to the statement after the label stops the statement.

## *Continue Statement*

Continued statements skip iteration that is in placed in for do-while, while, and for loops. Forms that are unlabeled are skipped to the parts that in the inner parts of the body and then evaluate a Boolean equation that dominates the loop. The example will show the directions through the string all the while counting how many times the letter p occurs. If there is no p, then the statement will skip the loop and move on to the next letter. If the letter is p then it will increase the count.

**Example**

```
Class ContinueDemo {
Public static void main(String[] args) {
String searchMe = “Peter Piper picked a “ + “peck of pickled peppers”;
Int max = searchMe.length();
Int numPs=0;
For (int I = 0; I < max; I++) {
// interested only in p’s
If (searchMe.charAt(i) ! = ‘p’)
Continue;
// process p’s
numPs++;
}
System.out.println(found” + numPs + “ p’s in the string. “ );
}
}
```

The output is 9 p’s.

If you remove the continue statement, there will be 35 p’s. A continue statement

that has a label is skipped for the current iteration on the outer loop with a label. In the following example there are two loops that have been nested to search for a substring in the string. Two nested loops will be required in order for one to iterate over the substring and any other string that may be searched.

**Example**

```
Class ContinueWithLabelDemo {

Public static void main(String[] args) {

String searchMe = “Look for a substring in me” ;

String substring = “sub”;

Boolean foundIt = false;

Int max = searchMe.length() –

Substring.length();

Test:

For (int I = 0; I <= max; i++) {

Int n = substring.length();

Int j = I;

Int k =0;

While (n -- != 0) {

If (searchMe.charAt(j++) != substring.charAt((k++)) {

Continue test;

}

}

foundIt = true;

break test;

}

System.out.println(foundIt ? “Found it” : “Didn’t find it”);

}
```

}

## *Returned Statements*

Returned statements are found at the end of branching statements. Returned statements are the exit method found in the most current control flow that will be returned to when the method was first invoked. There are two return statement forms. The value in one will be returned while the value in the other is not going to be returned. To return a value, you will need to have a value after your keyword that was returned.

The syntax is return ++ count;

After a data type has been returned it is going it will be required to match the type that the method declared. When a method declares the value as bad, the returned form is not going to give a value as a return.

The syntax for a value that is not returned as a value is return;

# Placement in JavaScript

JavaScript is flexible as to where it is actually placed when it comes to being put into an HTML document. There is a preferred way of putting it in a document though, just so that the coding knows what it needs to execute and when.

- Script in <head> will be </head>
- Script in the <body> will be </body>
- Script in the <body> </body> and <head> </head> section
- Section in an external file in the <head> section is </head>

## *Script in the Head Sections*

Whenever you need the script to run because of an event, like the user clicking their mouse, then you are going to have to place the script into the head section so that the event works out properly.

**Example**

```
<html>
<head>
<script type = “text/javascript”>
<! –
Function sayHello() {
Alert(“Hello Earth”)
}
// - - >
</script>
</head>
<body>
<input type = “button” onclick =” sayHello()” value = “Say Hello” />
</body>
</html>
```

The output for this is going to be Hello Earth.

## *Script in the Body Sections*

When you need the script to run while the page is loading so that it can generate the content that is located on the page, your script will need to go in the body part of your document to run correctly.

**Example**

```
<html>
<head>
</head>
<body>
<script type = “text/javascript”>
<! - -
Document.write (“Hello Universe”)
// - - >
</script>
<p> This is web page body </p>
</body>
</html>
```

The result for this code is going to be:

Hello Universe.

This is web page body

## *Code in the Body and Head Sections*

Code can be placed in both sections at the same time to create a different result than if it had just been placed in one section or the other.

**Example**

```
<html>
<head>
<script type = “text/javascript”>
<! - -
Function sayHello() {
Alert (“Hello people of Earth”)
}
// - - >
</script>
</head>
<body>
<script type = “text/javascript”>
<! - -
Document.write( “Hello people of Earth”)
// - - >
</script>
<input type = “button” onclick = “sayHello()” value = “Say Hello” />
</body>
</html>
```

Your result is going to be Hello people of Earth.

## *External Files*

Whenever you are working more in depth with JavaScript code, you will find that there are some cases where the same code is going to be placed across several pages on a single site. You are not going to have to use the same code with multiple HTML files. The script tags are going to give you the place that you are going to need to store the code in an external file so that you can proceed correctly with the HTML file.

**Example**

<html>

<head>

<script type = “text/javascript” src = “filename.js” ></script>

</head>

<body>

…..

</body>

</html>

When JavaScript is being used with an external file, you will have to type out all of the source code with the extension of .js.

**Example**

Function sayHello() {

Alert (“Hello son”)

}

# Cookies

Cookies are the data that you use to store in text files on a computer. The web server is going to send the browser a connection and once it has, the connection will be closed and the server is going to erase anything that it knew about the user. Cookies were created so that the problem of “how should information be remembered by a web browser?” could be fixed.

Websites remember information like the name of the users so that when the user returns to the website, they are greeted by name. The name value pair is going to be stored by cookies for use at a later date.

Browsers get requests from their servers on what needs to be stored. Any cookies that belong to the page are going to be added with the request for when the server is connected to the browser so that the server gets the correct information to remember data about the users.

## *Creating Cookies*

With JavaScript, you are going to be able to create, delete, and read cookies with the document.cookies function. Creating a cooking with JavaScript is going to look like this.

**Example**

Document.cookie = “username = Mary Jane” ;

You can also create expiration dates for the cookie so that whenever the browser is closed by the user, the information is erased.

**Example**

Document.cookie = “username = Mary Jane; expires Wed 9, July 2016 12:01:00 UTC”;

Given the correct parameters, the browser can know exactly which path the cookie is going to belong to. However, whenever a cookie is created, it is going to automatically belong to the page that is using it.

**Example**

Document.cookie = “username = Mary Jane; expires Wed 9, July 2016 12:01:00 UTC; path =/” ;

## *Reading Cookies*

JavaScript uses var x = document.cookie so that it can read any cookies. The document.cookie will return any cookies it finds in a single string.

## *Changing Cookies*

Cookies can be changed just like they are created. The same syntax is going to be used. Old cookies are going to be overwritten rather than deleted.

## *Deleting Cookies*

When you want to delete a cookie, you will need to set up a parameter so that the date has already passed and the cookie can expire.

**Example**

Document.cookie = “username = ; expires Sat 01 Jan 1980 00:00:00 UTC”;

You are not going to need to choose a specific cookie to delete.

## *Strings*

Document.cookie is going to appear as a normal text string, even though it is not. Whenever writing out the whole string with this function, you will read it with only the name value pairs.

New cookies are going to be set but the old cookies will not be overwritten. But, the new cookie is going to be added to the function document.cookie so that it can be read by that function again. If you want a value for a specific cookie, you will need to enter the whole function into JavaScript before searching for the cookie's value instead of the whole string's value.

**Example of a function set to a cookie**

First the function has to be created so that the name of the user can be stored inside of a cookie.

```
Function setCookie(cname, cvalue, exdays) {

Var d = new Date();

d. setTime(d.getTime() + (exdays * 24 *60 *60 *1000));

var expires = "expires=" + d.toUTCString();

document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
```

The parameters will name the cookie (cname) and then the value is going to be (cvalue) while the expiration date is going to be (exdays). The cookie is going to be set by stringing the name, value, and expiration date together in the string.

## *Functions for Getting Cookies*

Whenever you create a function, the return is going to be for one specific cookie's value.

**Example**

```
Function getCookie(cname) {
Var name = cname +"="
Var ca = document.cookie.split(';');
For (var I = 0; I <ca.length; i++) {
Var c = ca[i];
While (c.charAt(o) == ' ') {
C = c.substring(1);
}
If (c.indexOf(name) == 0 {
Return c.substring(name.length, c.length);
}
}
Return "";
}
```

The name of the cookie will be considered the parameter. To create a variable, you will need the name and the text so that you can search properly using (cname + "=".

The document.cookie will be split with semicolons so that there is an array that is called ca. Loops through the ca array are going to read every value that has been provided. Once the cookie is found, the value is going to be returned. If it is not found, then your return is going to be return"".

## *Checking Cookies*

After the cookie has been set, the greeting it was set to is going to be displayed. If it has not been set, then no greeting will be displayed but a prompt box will appear and ask for the user's name so that it can store the cookie for up to a year with the setCookie function.

**Example**

```
Function checkCookie() {
Var username = getCookie ("username");
If (username! = "") {
Alert( "Welcome again " + username);
} else {
Username = prompt ("Please enter your name:" , "");
If (username != "" && username != null) {
setCookie ("username", username, 365);
}
}
}
```

## *All Together Now*

The following example is going to use checkCookie() when the page has loaded.

**Example**

```
Function setCookie( cname, cvalue, exdays) {
Var d = new Date();
d.setTime(d.getTime() + (exdays* 24 *60 *60 *1000));
var expires = “expires=” +d.toUTCString();
document.cookie = cname + “=” + cvalue + “;” + expires + “;path=/”;
}
Function getCookie(cname) {
Var name = cname + “=”;
Var ca = document.cookie.split(‘;’);
For( var I = 0; I < ca.length; I++) {
Var c = ca[i];
While (c.charAt(0) == ‘ ‘) {
C = c.substring(1);
}
If (c.indexOf(name) == 0) {
Return c.substring(name.length, c.length);
}
}
Return “”;
}
Function checkCookie() {
Var user = getCookie(“username”);
```

```
If (user != “”) {
Alert (“Welcome again + user);
} else {
User = prompt (“Please enter your name:” , “”);
If (user != “” && user != null) {
setCookie( “username”, user, 365);
}
}
}
```

# What You Can and Cannot Do with JavaScript

There are some things that you are going to be able to do with JavaScript. Just like any other program, there are going to be limitations as to what JavaScript is going to allow you to do.

## *What Can I Do With JavaScript?*

- Interact with forms
- Give feedback
- Detect what a user is doing
- Notify users of things
- Ask if something is correct or not
- Get the user to input data
- Validate a stopped script
- Conversions
- Calculations
- Add content to documents
- Work with pop-up windows
- Dom scripting
- Sign up forms
- Search boxes
- Constantly changing information
- Information for forms
- Layout issues
- Enhance HML interfaces
- Animate elements

## *Things You Cannot Do With JavaScript*

- Write files on a server without help
- Access databases
- Read or write on files from clients
- Close a window it did not open
- Access web pages in another domain
- Protect your pages source or images

# Common Pitfalls

- Confusing the equals operator ( = =) and the does not equal operator ( !=).
- Confusing the strict equals operator ( = = =) and the strict does not equals operator ( ! = =).
- There are two types of operators because of JavaScript's premise of true and false statements.
- ==/ != works when you expect the operands to be the same type, if they aren't these operators will try to force them to be.
- Truthy and falsy values  are for non-Boolean expressions  by pretending they are Boolean.
- These values will be evaluated as false: NaN, False, Null, 0, Empty string ("").
- === and !== are better to use than == and !=.
- NaN is not a number that is only helpful in edge cases.
- If you have to use NaN use isNan().
- Eval() is a function that tries to do everything.
- Script is going to need to be allowed to run dynamically.
- Eval() can be used for JSON.
- However, try and stay away from eval().
- JavaScript may be dynamic, but it has to be organized for maintenance experience.

# Tips and Tricks

**Don't forget the var keyword when assigning values**

Assignments can be undeclared variables that get results thanks to a global variable that is created to avoid global variables.

**Use = = = not = =**

== and != is going to perform an automatic conversion if it is needed. === and !== cannot perform conversions. It will however compare types and values faster than ==.

**Null, 0, false, undefined, NaN and empty string ("") are all falsy**

**Use a semicolon for the termination of a line**

Semicolons are a good way to practice line terminations. In many problems, it is inserted by JavaScript parser.

**Create object constructors**

```
Function Person(firstName, lastName) {
This. firstName = firstName;
This. lastName = lastName;
}
Var Nathan = new Person("Nathan", "Clark") ;
```

**Be careful as you use instance of, constructor, and typeof**

Typeof is a unary operator that returns strings to represent primitive variable types. Typeof null is going to return "object" for many object types.

Constructor is the property of internal prototype property. It can be written over by code.

Instanceof will check the chain from the constructor to see if it is going to be returned true or false.

**Create functions that are self-calling**

Also known as a self-invoked anonymous function or immediately invoked function expression, the function will automatically execute when it is created.

**Get random items from arrays**

```
Var items = [12, 548, ‘a’, 2, 5478, ‘foo’8852, ‘Doe’, 2145, 119];

Var randomItem = items [Math.floor (Math . random() * items. Length)];
```

**Get random numbers in a particular range**

This code is useful when you are trying to create false data for testing reasons like a salary that is between two sets of numbers.

```
Var x = Math.floor (Math.random() * (max – min + 1)) + min;
```

**Generate an array of numbers with numbers from 0 to max**

```
Var numbersArray = [] * (max – min +1)) + min;

For( var I = 1; numbersArray.push (i++) < max;) ; // numbers = [1, 2, 3 …100]
```

**Generate random sets of alphanumeric characters**

```
Function generateRandomAlphaNum (len) {

Var rdmString = “”;

For( ; rdmString.length < len; rdmString += Math.random() .toString(36). Substr (2)) ;

Return rdmString.substr (0, len);

}
```

**Shuffle arrays of numbers**

The better option is to implement random sort orders by their code by using native JavaScript sort functions.

**String trim functions**

A trim function that is classic from JavaScript and other programming languages that remove any whitespace from the string that is not going to happen in JavaScript so it can be tacked to a string object. Native implementations of trim() function is used in more current JavaScript engines.

**Append to arrays from different arrays**

```
Var array1 = [12, “foo”, {name “Joe”} , -2458];

Var array2 = {“Doe” , 555, 100};

Array.prototype.push.apply (array1, array2) ;

/*array1 will be equal to [12, “foo”, {name “Joe”}, -2458, “Doe”, 555, 100]
```

**Turn argument objects into arrays**

```
Var argArray = Array.prototype.slice.call(arguments) ;
```

**Verify given arguments with a number**

```
Function isNumber(n) {

Return !isNaN(parseFloat (n)) && isFinite(n);

}
```

**Verify that a given argument is an array**

Note: if the toString() method is written over. You are not going to receive the results you expect by using this trick. You are also able to execute instanceof if you do not work with several frames. But, if you have various contexts, you are going to get the wrong result.

**Get the max and min in an array of numbers**

Var numbers = [5, 458, 120, -215, 228, 400, 122205, -85411];

varmaxInNumbers = Math.max.apply (Math, numbers) ;

var minInNumbers = Math.min.apply (Math, numbers) ;

**Empty an array**

Var myArray = [12, 222, 1000 ];

myArray.length = 0; //myArray will be equal to[]

**Do not delete to remove items from arrays**

Use the splice function instead of the delete function if you want to remove items from an array. When you use the delete function, it is going to replace the item with an undefined variable instead of deleting it from the array.

**Truncate arrays by using length**

Var myArray = [12, 222, 1000, 124, 98, 10];

myArray.length = 4; //myArray will be equal to [12, 222, 1000, 124]

**Use logic of AND/ OR for conditions**

Logical OR is also going to be used in setting a default value for function in an argument.

**Use map() functions to loop through an array's item.**

Var squares = [1,2,3]. Map(function (val) {

Return val* val

)) ;

//squares will be equal to [1, 4, 9 16]

**Rounding a number to the N decimal place**

Var num = 2.443242342;

Num = num.toFixed(4); //num will be equal to 2.4432

The toFixed() function is going to return a string instead of a number.

**Floating point problems**

0.1+ 0.2 is going to equal 0.30000000000000004. The knowledge that you need to know is that all of the numbers in JavaScript are floating points that are represented by a 64 bit binary internally.

**Check the properties of an object when it is used in a for-in loop**

The code snippet is going to help avoid any iterating through the properties of the objects prototype.

```
For (var name in object) {
If (obeject.hasOwnProperty(name)) {
// do something with name
}
}
```

**Comma operator**

```
Var a = 0;
Var b = (a++, 99) ;
Console.log(a); // a is going to be equal to 1
Console.log(b); // be is going to be equal to 99
```

**Cache variables that have to be calculated or queried**

The jQuery selector is going to cache the DOM element

Var navright – document.querySelector (‘#right’);

Var navleft = document.querySelector (‘#left’) ;

Var navup = document.querySelector (‘#up’) ;

Var navdown = document.querySelector (‘#down’);

**Verify arguments before passing it to isFinite()**

isFinite (0/0) ; // false

isFinite (“10”) ; //true

**Stay away from negative indexes in arrays**

Var numbersArray = [1, 2, 3, 4, 5] ;

Var from = numbersArray. indexOf (“foo”) ; //from is equal to -1

numbersArray.splice(from,2) ; //will return [5]

Any arguments that pass through splice should not be negative.

# Loops

When an operation is repeated, it is considered to be a loop. Loops have several instructions that considered in continuing with the same code block until the condition has been returned as true or false depending on which one is needed. To govern loops, you have to have a variable that is conflicting that is going to increase or decrease each time that a loop is repeated.

There are two loop types used in JavaScript. For loops and while loops. Statements will be executed at their best when they have been used in a loop. If you use a break, you can continue the statement inside of the loop.

## *For Loop*

For loops are going to be executed until the condition returns as false. The syntax is the same as every other programming language. There are three arguments with a for loop.

**Example**

```
For (initialization; condition; increment)
{
// statements
}
```

Loop statements can only be executed if:

- The statement and the control go to step two.
- The equation is performed and the condition is true. Then the statement will be carried out if the value is false and the loop will then be terminated.
- The expression at the beginning is executed. This expression will initialize one or several different loops as they move through the syntax to allow expressions from various complexities and degrees.
- The increment definition is removed, updated, or executed.

**Example**

```
<script type = “text/ javascript”>
Document.write(“<hl>Multiplication table</hl>”);
Document.write(“table border = 2 width = 50%”);
For (var I = 1; I <=9;i++) { //this is the outer loop
Document.write(“<tr>”);
Document.write(“<td>” +I + “</td>”);
For (var j = 2; j <= 9; j++) { // inner loop
```

```
Document.write (“<td>” +I *j + “</td>”;
}
Document.write(“</tr>”);
}
Document.write(“</table>”);
</script>
```

The example below will have a for statement that is going to count the number of options that have been selected in the list. The for statement is going to declare any variables of I and initialize them to zero. The I variable is going to be checked less than how many choices have been selected. If it is successful, then the comment and the variable I are going to increase by one each time it passes through the loop.

**Example**

```
<script type = “text/javascript”>
Function howMany (selectItem) {
Var numberSelected = 0
For (var I = 0; I < selectItem.options.length; i++) {
If (selectedItem.options[i] . selected == true)
numberSelected++;
}
Return numberSelected
}
</script>
<form name = “selectForm”>
<p>Choose some book types, then click the button below: </p>
<select multiple name = “bookTypes” size = “8”>
<option selected> Classic </option>
```

<option> Information Books </option>

<option> Fantasy </option>

<option> Mystery </option>

<option> Poetry </option>

<option> Humor </option>

<option> Biography </option>

<option> Fiction </option>

</select>

<input type – “button” value = “How many are selected?”

Onclick = “alert (‘Name of options selected: ‘ + howMany(document.selectForm.bookTypes)) “>

</form>

However many books the user selected will be the output. For each user, the output is going to be different. But, the output will always be somewhere between zero and eight for this example.

## *While Loops*

While loops is a loop that is commonly found in JavaScript, but it is not the most common. The while statement will repeat its loop until the condition is found to be true. If the condition is false, then the statement will stop and the control is going to be passed to the statement that follows the loop.

**Syntax**

```
While (condition)
{
// statements
}
```

The following example is going to show how to define a loop that begins with a variable of I = 0 so that it will continue to run until the I variable is more than 10. Every statement that passes through the loop, I is going to increase by one.

**Example**

```
<script type = “text/javascript”>
Var I = o;
While (i<=10) // output from the value from 0 to 10
{
Document.write(I + “<br>”)
I++;
}
</script>
```

No matter what loop you are currently writing, you need to ensure that the conditions are going to eventually turn out to be false. If this does not happen, then the loop will never terminate and it will go on forever.

## *Break and Continue*

Sometimes loops may start without any constraints by a true/false condition. But, the statements that are located inside of the brackets are going to decide when it is time to exit the loop. Two statements can be found inside of a loop, the break statement and the continue statement.

Continue statement is going to stop the statement block for any for or while loops before continuing the loop at the next starting point. Break statements are going to stop the while or for loop before continuing with the script that trails the loop should there be any.

**Example**

```
<script type – “text/javascript”>

Document.write(“<p><b>Example of using the break statement: </b></p>”);

Var I = 0;

For (i=0; i<=10; i++) {

If (i==3) {break}

Document.write (“The number is “ +i);

Document.write(“<br />”);

}

Document.write(“<p><b>Example of using the continue statement: </b><p>”);

Var I = 0

For (I = 0; I <= 10; i++) {

If ( i==3) (continue)

Document.write(“The number is “ + i);

Document.write(“<br />”)

}

</script>
```

# Operators in JavaScript

Expressions are going to have operators and operands. Operands are the numbers that are used in the expression and the operator is the arithmetic sign. The types of operators supported by JavaScript are:

- Arithmetic
- Comparison
- Logical
- Bitwise
- Assignment
- Conditional
- Typeof

## *Arithmetic*

- Addition where the operands are added together.
- Subtraction where the second operand is going to be subtracted from the first.
- Multiplication where both operands are multiplied together.
- Division where the numerator is divided by the denominator.
- Modulus where the remainder is going to be an integer that is left over from division.
- Increment where the integer increases by one.
- Decrement where the integer decreases by one.

**Example**

```
<html>
<body>
<script type = “text/javascript”>
<! - -
Var a = 33;
Var b = 10;
Var c = “test”;
Var linebreak = “<br />”;
Document.write(“a + b =”);
Result = a + b;
Document.write (result) ;
Document.write(linebreak);
Document.write (“ a – b = “);
Result = a – b;
Document.write(result);
```

```
Document.write(linebreak);
Document.write(“a / b = “);
Result = a / b;
Document.write(result);
Document.write(linebreak);
Document.write(“ a % b = “);
Result = a % b;
Document.write(result);
Document.write(linebreak);
Document.write( “a + b + c = “);
Result = a + b + c;
Document.write(result);
Document.write(linebreak);
A = ++a;
Document.write (“++a = “);
Result = ++a;
Document.write(result);
Document.write(linebreak);
B = --b
Document.write(“ - - b = “);
Result = - - b;
Document.write(result);
Document.write(linebreak);
// - - >
< / script>
```

Set the variables to a different value and try it yourself.

</body>
</html>

Your output is going to be:

A + b = 43

A – b = 23

A / b = 3.3

A % b = 3

A + b + c = 43Test

++a = 35

-- b = 8

## *Comparison*

- Equal is going to be the value from two operands that will check to see if they are equal or not. If they are equal, then the condition is true.
- Not equal will check to see if the operands are equal, if they are not, then the condition is true.
- Greater than will say that the left value is going to be greater than the one found on the right and if it is, then the condition for it is true.
- Less than says that the left number should be less than that on the right and if it is, then it is true.
- Equal to or greater than will check to see if the left value is larger than or equal to the value found on the right. If it is, then it is true.
- Less than or equal to is going to state that the value on the left should be of lesser value or of equal value on the right and if it is found to be, then the condition is true.

**Example**

```
<html>
<body>
<script type = “text/javascript”>
<! - -
Var a = 10;
Var b = 20;
Var linebreak = “<br />”;
Document.write( “(a == b) => “);
Result = ( a == b);
Document.write(result);
Document.write(linebreak);
Document.write( “ (a < b) => “);
```

```
Result = (a < b);
Document.write(result);
Document.write(linebreak);
Document.write( " (a > b) => " );
Result = (a > b);
Document.write(result);
Document.write(linebreak);
Document.write ( " (a != b) => ");
Result = a != b);
Document.write(result);
Document.write(linebreak);
Document.write( " (a >= b) => ");
Result = ( a >= b);
Document.write(result);
Document.write(linebreak);
Document.write( " (a <= b) => ");
Result = (a <= b);
Document.write(result);
Document.write(linebreak);
// - - >
</script>
</body>
</html>
```

The output for this example is going to be:

(a == b) = false

(a < b) = true

(a > b) = false

(a != b) = true

(a >= b) = false

(a <= b) = true

## *Logical*

- Logical AND states that both operands have to be non-zero. If they are then the condition is found to be true.
- Logical OR says that at least one of the operands has to be non-zero. If one is found to be non-zero, then it is a true statement.
- Logical NOT says that the logical state is going to be reversed based on the operands state. If the condition is true, then it will be false because of the logical NOT.

**Example**

```
<html>
<body>
<script type = “text/javascript”>
<! - -
Var a = true;
Var b = false;
Var linebreak = “<br />”;
Document.write (“ (a && b) => “);
Result = (a && b);
Document.write(result);
Document.write(linebreak);
Document.write (“ (a || b) = > “);
Result = (a || b);
Document.write(result);
Document.write(linebreak);
Document.write( “! (a && b) => “);
Result = (! (a && b));
```

```
Document.write(result);
Document.write(linebreak);
// - - >
</script>
</body>
</html>
```

The results for this example is going to be:

(a && b) = false

(a || b) = true

!(a && b) = true

## *Bitwise*

- Bitwise AND takes the Boolean value and the operation for every bit with the individual arguments.
- Bitwise OR takes the Boolean and it is performed or the operation of every integer is performed.
- Bitwise XOR is a Boolean exclusive and that is what will be performed or the operation for the integers. Either the operand on the right or the one on the left can be exclusively true, both cannot be.
- Bitwise not is a unary operation that is going to reverse all bits that can be found in the operand.
- A left shift is when the first operand moves to the left however many bits are in the second operand.
- A right shift is going to be a binary shift to the right the number of bits that is found in the right operand.
- A right shift with zero is going to make the operand move to the right but the bits on the left are always going to be zero.

**Example**

```
<html>
<body>
<script type = “text/javascript”>
<! –
Var a = 2; // bit is going to be presented as 10
Var b = 3; // bit will be presented as 11
Var linebreak = “<br />”;
Document.write (“ (a & b => “);
Result = (a & b);
Document.write(result);
```

```
Document.write(linebreak);
Document.write (“ (a | b) => “);
Result = ( a | b);
Document.write(result);
Document.write(linebreak);
Document.write(“ (a ^ b) => “);
Result = (a ^ b);
Document.write(result);
Document.write(linebreak);
Document.write(“ (~b) => “);
Result = (~b);
Document.write(result);
Document.write(linebreak);
Document.write(“ (a << b) => “);
Result (a << b);
Document.write(result);
Document.write(linebreak);
Document.write(“ (a >> b) => “);
Result = (a >> b);
Document.write(result);
Document.write(linebreak);
//- - >
</script>
</body>
</html>
```

Results:

(a & b) = 2

(a | b) = 3

(a ^ b) = 1

(~b) = -4

(a << b) = 16

(a >> b) = 0

## *Assignment*

- Simple assignment will assign the value on the right to the left.
- Add and assign will take the right operand, add it to the left, and then assign it to the left.
- Subtract and assign is going to subtract the right value from the left and assign it to the left.
- Multiply and assign will multiply the operands together and then assign the value to the left.
- Divide and assign will divide the left from the right and assign the value to the left.
- Modulus and assign will take the modulus from both operands and assign the remaining value to the left.
- Bitwise operators are going to become <<=, >>=, &=, |= and ^=

**Example**

```
<html>
<body>
<script type = “text/javascript”>
<! - -
Var a = 33;
Var b = 10;
Var linebreak = “<br />”;
Document.write(“value of a => (a = b) => “);
Result = (a = b);
Document.write(result);
Document.write(linebreak);
Document.write (“Value of a => (a += b) => “);
```

```
Result = (a += b);
Document.write(result);
Document.write(linebreak);
Document.write (“Value of a=> (a *= b) => “);
Result = (a *= b);
Document.write(result);
Document.write(linebreak);
Document.write(“ Value of a => (a /= b) => “);
Result = (a /= b);
Document.write(result);
Document.write(linebreak);
Document.write (“Value of a => (a %= b) => “);
Result = (a %= b);
Document.write(result);
Document.write(linebreak);
// - ->
</script>
</body>
</html>
```

Output:

```
Value of a => (a =b) = 10
Value of a => (a += b) = 20
Value of a => (a -=b) = 10
Value of a => (a *= b) = 100
Value of a => ( a /= b) = 10
```

Value of a => (a %= b) = 0

## *Conditional*

- The conditional operator will evaluate the expression for a true or false value before the statements are executed depending on what the result is once it has been evaluated.
- Conditional statements will be true if the condition is true, then the statement's value is going to be x. if it is not, then you will move on to the other statement.

**Example**

```
<html>
<body>
<script type = "text/javascript">
<! - -
Var a = 10;
Var b = 20;
Var linebreak = "<br />";
Document.write (" ((a >b) ? 100 : 200) => ");
Result = (a > b) ? 100 : 200;
Document.write(result);
Document.write(linebreak);
Document.write (" (( a < b) ? 100 : 200) => ");
Result = (a < b) ? 100 : 200;
Document.write(result);
Document.write(linebreak);
// - - >
</script>
</body>
```

</html>

The results are:

(( a > ) ? 100 : 200 => 200

((a < b) ? 100 : 200 => 100

## *Typeof*

- Typeof operators are unary operators. It is placed before a single operand. The operand can be any type.
- The value will become a string that is going to show the data types that have been used.
- Typeof operators evaluate strings, numbers, and Boolean only if the value is true or false based on the evaluation of them.
- These are the values that are returned with typeof operators: Number = number, Object = object, Null = object, String = string, Function = function, Undefined = undefined.

**Example**

```
<html>
<body>
<script type “text/javascript”>
<! - -
Var a = 10
Var b = “String”;
Var linebreak = “<br />”;
Result = (typeof b == “string” ? “B is String” : “B is Numeric”);
Document.write( “Result => “);
Document.write(result);
Document.write(linebreak);
Result = (typeof a == “string” ? “A is String” : “A is Numeric”);
Document.write(“result => “);
Document.write(result);
Document.write(linebreak);
```

```
// - ->
</script>
</body>
</html>
```

Results:

Result => B is String

Result => A is Numeric

# Page Printing with JavaScript

You may find that you want a button that allows you to print the page's contents to an actual printer. JavaScript has a function that makes this possible. The window.print() function is going to print whatever page you are currently viewing. This function is also considered an onclick event.

**Example**

```
<html>
<head>
<script type = "text/javascript">
<! - -
// - - >
</script>
</head>
<body>
<form>
<input type = "button" value = "Print" onclick = "window.print()" />
</form>
</body>
<html>
```

This is going to aide you in getting a printout of what you were looking at, but it is not recommended to use when you want to print a page. There is a printer friendly page that will not have any images such as the ads that usually appear on the sides of web pages.

In order to make a page printer friendly, you need to:

- Make a copy of the text on the page without any of the images that you do not want.
- If you do not want extra copies then you are going to need to make the

printable text with the appropriate comments.

- <! – PRINT STARTS HERE - - > <! - - PRINT ENDS HERE - - >

If those steps do not work, then use the toolbar on the browser to print the page. The file path is file – print – okay

# Conclusion

Thank for making it through to the end of *JavaScript: Programming Basics for Absolute Beginners,* let's hope it was informative and able to provide you with all of the tools you need to achieve your goals whatever it may be.

The next step is to take your new-found knowledge and apply it to creating JavaScript coding.

There is always more for you to learn and if you are interested, you are going to be able to go online and find books and other resources that you can use to further your knowledge so that you are able to do more with JavaScript.

Do not be discouraged if your script does not do what you want it to the first time, it is going to take patience and practice for you to get your script correct.

Finally, if you found this book useful in any way, a review on Amazon is always appreciated!

# About the Author

Nathan Clark is an expert programmer with nearly 20 years of experience in the software industry.

With a master's degree from MIT, he has worked for some of the leading software companies in the United States and built up extensive knowledge of software design and development.

Nathan and his wife, Sarah, started their own development firm in 2009 to be able to take on more challenging and creative projects. Today they assist high-caliber clients from all over the world.

Nathan enjoys sharing his programming knowledge through his book series, developing innovative software solutions for their clients and watching classic sci-fi movies in his free time.

To learn programming from an expert, look out for more of Nathan's books in store and online.