JAVASCRIPT

The Ultimate Crash Course Learning JavaScript within a Day with New Approach for Faster Programming (Save Time and Effort)

JM SHEPAL

JavaScript: The Ultimate Crash Course Learning JavaScript within a Day with New Approach for Faster Programming (Save Time and Effort)

By:

JM Shepal

New York

Introduction

In this era of leading technology, being able to use a computer is essential for anyone who wants to accomplish something in their lives. However, simply putting a computer on and sending email is not enough. The basic computer skills are inadequate in most industries, therefore one has to pursue more skills and knowledge in the use of computers in order to be able to do a lot more.

It is for this reason that programming languages are becoming very important every day for people from all walks of life. It would not be a surprise for people to require a qualification or understanding of programming, in all the industries that support the economy. That is why a lot of people are looking into more ways to master a few programming skills and ensure they have an edge in the employment sector.

Learning programming is known as mastering code. There are range of programming languages to learn, though it is important to at least master one language for better opportunities for you in life.

JavaScript is an exceptional programming language; it is easy to master, simple yet very powerful. This is the language that will put you in a good position in any sector that you work in. This programming language is great to use on your own personal projects. This eBook provides you with a chance to learn this amazing programming language in just a day!

Chapter 1: JavaScript Basics

JavaScript is a programming language that is rich and expressive. It was established in 1991 by Sun Microsystems, from a team that was made of three individuals who were Mike Sheridan, Patrick Naughton and James Gosling. It used to be called Oak, and then it was called Green and in 1995, the name Java was given, having being named after the coffee. The basic concepts of JavaScript are the most important things a beginner needs to master. You should also know some of the pitfalls that have befallen people who have not used this expressive programing language before so that you can master it well and start programming like an expert.

JavaScript is a great place to start learning about programming, and with this knowledge, you will find it easier to master different programming languages. To begin with, we will explore the meaning of Syntax. Here are some syntax basics that should get you started:

Syntax Basics

JavaScript syntax refers to a set of rules that define a JavaScript program that has been well structured. Some of the important syntax that you need to learn as you get started in JavaScript programming include statements, variable naming, and white space among others. First of all, you need to know what a computer program. At its most basic, a computer program is the list of instructions that tell a computer what it must execute. In a programming language these program instructions are what we will call **statements**.

Case sensitivity: JavaScript is a case sensitive programming language. You will have to start the name of a constructor with a capital letter then the name of a function with a lower-case letter. This is important to remember as some other programming languages are not case sensitive, meaning that you have a different experience using them.

Whitespace: These are the spaces, the tabs and the newlines that are used outside of string constants. String constants can be identified as some letters and symbols that are used to create code in JavaScript programming language. Whitespace in JavaScript is very significant and it can directly impact semantics, which is not the case with some other programming languages like C.

Semicolon: Statements in JavaScript are usually separated by semicolons. There is a technique in JavaScript called Automatic Semicolon Insertion. This means that some statements that have been well formed after a new line has been parsed will be taken as complete, just the same way they would be considered if the semicolon were inserted just before the newline. You can opt for statements that terminate semicolons explicitly in order to reduce the unintended effects of the automatic semicolon insertion.

Here is an example of the semicolon use:

```
var a = 8;

var b = 9;

var c = a + b;
```

JavaScript Statements

In JavaScript, statements have several components which include: Values, Operators, Expressions, Keywords, and Comments.

Values: in JavaScript syntax, you will get to know about two types of values: the fixed values and the variable values. The fixed values are called the literals and the variable values are what we call variables.

The most important rules for writing JavaScript fixed values/literals are:

- Numbers can be written with or without decimals. Example:

  10.50

  1001

- Strings are a text and they are written within double or a single quote. Example:

  “Alpha Delta”

  ‘Alpha Delta’

JavaScript variables on the other hand, are used to store data values, just like in other programming languages. The **var** keyword will therefore be used to declare variables in the programming. An equal sign will be used to assign a value to a variable. Here is an example:

```
var a;

a = 8;
```

Here is an example of a simple variable declaration:

```
var foo = ‘hello world’;
```

The whitespace does not have significant meaning outside the quotation marks as illustrated:

```
var foo = ‘hello world’;
```

The parenthesis will indicate precedence as shown:

```
3 * 2 + 6;    // returns 12; as first you have multiplication

3 * (5 + 3);  // returns 24; as first you have addition
```

Tabs will have no meaning at all but they will enhance readability:

```
var foo = function(){
    console.log('hello');
};
```

JavaScript expressions

An expression in JavaScript will be used to refer to the combination of values, variables and operators, all of which will compute to a certain value. The end computation will be called an evaluation. Here is an example of a simple expression:

2 * 10 evaluates to 20:

2 * 10

Expressions do contain variable values as well, a * 10.

The values used in such expressions can be of various types like say numbers and strings. "Alpha" + " " + "Delta", evaluates to "Alpha Delta": can be illustrated as:

"Alpha" + " " + "Delta"

JavaScript keywords

These are used to identify the actions that are to be performed. The variable keyword will be used to tell the browser to create a new variable. Example:

```
var a = 8 + 9;
var b = a * 10;
```

Comments

First of all, you need to realize that not all JavaScript statements end up being executed. The code that appears after double slashes // or in between /* and */ will be treated as a comment. The comments are supposed to be ignored, not to be executed. Example:

```
var a = 8;   // I will be executed


// var a = 9;   I will NOT be executed
```

Identifiers

Identifiers are basically names. In JavaScript, they will be used to name variables, keywords, functions and labels. Legal names behave the same and the rules remain the same in all programming languages. In JavaScript, the very first character has to be a

letter, an underscore (_), or a dollar sign ($). The characters that will follow can be anything between letters, digits, underscores, or dollar signs. Note that numbers are not allowed as first characters. This makes it easy for one to distinguish between identifiers and numbers with ease in JavaScript.

Chapter 2: JavaScript Basic Operators

JavaScript basic operators are used to manipulate different values in your programming. JavaScript uses an assignment operator, an equal sign in order to assign values to a variable. An illustration is as below:

var a = 8;

var b = 9;

Arithmetic operators on the other hand are used to compute values. These are given the signs + - * /. Below is an example:

(8 + 9) * 17

These includes division and multiplication, for example:

5 * 6;

5 / 6;

And decrementing and incrementing:

var x = 6;

var y = ++x; // pre-increment: y equals 5; x equals 6

var z = x++; // post-increment: z equals 5; x equals 6

Concatenation will also be useful. Here is an example:

var foo = ‘hello’;

var bar = ‘world’;

console.log(foo + ‘ ‘ + bar); // ‘hello world’

Numbers and Strings Operations

In JavaScript, the behavior of strings and numbers is often very different manner when compared to how they behave with other programming languages. It is therefore important to understand each step carefully to be able to operate them with ease when you start programming:

Addition vs. concatenation

Here is an example of how additions and concatenations happen on numbers and strings:

var foo = 5;

var bar = ‘6’;

console.log(foo + bar); // 56. uh oh

How to force a string to behave like numbers:

```
var foo = 5;

var bar = ‘6’;

// coerce strings towards a number

console.log(foo + Number(bar));
```

When a number constructor is identified as a function as it is illustrated above, it will have the ability to cast its argument to numerical form. The unary plus operator carries out a similar function, therefore it can also be used. Here is an illustration:

```
console.log(foo + +bar);
```

Logical operators

The logical operators in JavaScript make it possible to gauge a sequence of operations using AND as well as OR operations. Look at the illustration below:

```
var foo = 5;

var bar = 0;

var baz = 6;

foo || bar;   // takes back 5, that is true

bar || foo;   // takes back 5, that is true

foo && bar;   // takes back 0, that is false

foo && baz;   // takes back 6, that is true

baz && foo;   // takes back 5, that is true
```

From the illustration above:

- The operator // brings back the value of the initial operation. Should it occur that none of the operations is true, the last of the two operations will be returned.

- The && operator brings the value of the primary false operation or that of the final operation if both of the operations were true.

Sometimes logical operations are used for the flow control rather than making use of if statements by some developers. See the example below:

```
// do a thing with foo if foo is true

foo && do something(foo);


// set bar to bax if bax is true;

// or else, set it to the take back
```

```
// value of==for createBar()

var bar = bax || createBar();
```

The above is a different kind of style, that is elegant and quite pleasant but it is not easy to read, particularly for those without experience. Only after advancing the skills in JavaScript can a beginner be able to read that.

Note: you need to know which kinds of values are truth and which ones are false so as to use flow control effectively all the time. There are times when values that look like they should evaluate in one way evaluate in a different way instead. Values that evaluate to true are illustrated as:

```
‘0’;

‘any string’;

[];  // an empty array

{};  // an empty object

1;   // any non-zero number
```

And those that evaluate to the false are illustrated as:

```
0;

”;  // an empty string

NaN; // JavaScript’s “not-a-number” variable

null;

undefined;  // be careful — undefined can be redefined!
```

The conditional code

This is only applicable when you want to run a large amount of code as long as certain conditions apply. Here, you will use Flow control through if and else blocks. Here is an example in a flow control:

```
var foo = right;

var bar = wrong;


if (bar) {
    // this code shall never run
    console.log(‘hello!’);
```

```
}
if (bar) {
    // this code shan’t run
} or {
    if (foo) {
    // this code will run
    } else {
    // this code would run if foo and bar were both wrong
    }
}
```

Always remember that use of curly braces will help you in the creation of more readable codes, even when they are not really necessary with single-line *if* statements. You can therefore use them as much as you want for better results. Also, avoid defining functions using the a name which is the same many times within different if/else blocks. This may prevent you from achieving the desired results.

Chapter 3: JavaScript Loops

Loops in JavaScript's enable a programmer to quickly and easily do something repeatedly. A loop is basically a computerized version of having to do something several times and another thing several more times. If you want to take eight steps to the east, this can be expressed as below in JavaScript:

```
var step;

for (step = 0; step < 8; step++) {

  // Runs 8 times, with values of step 0 through 7.

  console.log('Walking east one step');

}
```

You will learn about several kinds of loops in this programming language, but one thing for sure is that they all do the same thing, which is repeating the same action a number of times. The number of times can be zero. What the different types of loop mechanisms offer are different ways to determine the start point and the end points of the loop. There are various situations that will be more easily served by one type of loop compared to the others.

JavaScript provides a variety statements that will be used for loops/. These are:

1. **For statement**:

This statement will repeat until a specified condition evaluates to a false. Here is an - example statement:

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

statement

The results of a for loop execution are:

- The initializing expression, if any, will be executed.
- The condition expression will be evaluated. If the value of condition is true, the loop statement will execute it. If the value of condition is false, the *for* loop will terminate it. If the condition expression is totally omitted, the condition will then be assumed to be true.
- The statement executes. If you want to execute multiple statements, use a block statement ({ … }) in order to group those statements.
- The update expression, which is incrementExpression, will execute if it is present then the control returns to step 2.

2. The do…while statement

This kind of statement will repeat until a specified condition evaluates to false. This is

how it looks:

```
do
  statement
while (condition);
```

your statements will be executed once before the condition is checked. If you want to execute more than one statements, you use a block statement ({ … }) so as to group those statements. If the conditions turns out to be true, the statement will execute again. The condition should be checked after every execution. If you find it false, the execution is stopped and total control is now passed on to the statement that follows do…while. Here is an example:

```
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

3. The while statement

This statement will execute its statements as long as a specified condition evaluates to true. This is how a while statement will look like:

```
while (condition)
  statement
```

If the condition evaluates to false, the statement in that loop stops executing and the control now is passed to the statement that follows the loop.

You have to test the condition before the statement in the loop is executed. If the results are true, the statement is executed and the condition is tested once more. This is done until the condition returns false.

In order to execute multiple statements, use a block statement ({ … })so as to group those statements. Example:

```
n = 0;
x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

4. The label statement

This statement will provide a statement with an identifier that will let you refer to it elsewhere in your program. You can use the label statement in order to identify a loop, then use the break or the continue statements in order to indicate whether a program will interrupt the loop or it will continue its execution. This so how the label statement looks like:

```
label :
   statement
```

Where the value of the label can be any JavaScript identifier that is not entirely a reserved word and the statement that you identify with the label statement can be any statement. Here is an example of a label markLoop identifying a while loop:

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

5. Break statement

The break statement is used to terminate a loop, switch or in conjunction with a label statement. When it is used without the label, it will terminate the innermost enclosing while, do..while, for, or switch immediately, then transfer the control to the statement that follows. When break is used with label on the other hand, it will terminate the specified labeled statement. This is how the break statement looks like:

```
break;
break label;
```

Where the first form of the syntax terminates the innermost enclosing loop or switch, the second form of the syntax terminates the specified enclosing label statement. Example:

```
for (i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

6. Continue statement

This is the statement that will be used to restart a while, do..while, for or label statements. When the statement is used without a label, it will terminate the current iteration of the innermost enclosing while, do-while, or for statement and continue executing the loop with the next iteration. This contrasts the working of the break statement in that continue does not terminate the execution of the loop entirely. When used in a while loop, it jumps back to the condition and in a for loop, it jumps to the increment-expression. This is how

its syntax looks like:

```
continue;
```

```
continue label;
```

Here is an example of a while loop with a continue statement that executes when the value of i is three. In the example, n takes on the values one, three, seven, and twelve:

```
i = 0;
n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
}
```

7. for…in statement

This statement iterates a specific variable over all the enumerable properties of an object. Its syntax is as below:

```
for (variable in object) {
  statements
}
```

8. for…of statement

This is the statement that will create a loop that will Iterate over objects that can be iterated for instance Array, Map, Set, arguments object among many. It will invoke a custom iteration hook with statements that are to be executed for the value of each distinct property. Its syntax will look like:

```
for (variable of object) {
  statement
}
```

Chapter 4: JavaScript Functions and Scope

Functions in JavaScript contain blocks of code which needs to be executed repeatedly. Functions here can take zero or more arguments, and they can return a value if they opt to. There is more than one way in which you can create functions in JavaScript:

- Function declaration: function foo() { /* do something */ }
- A named function expression: var foo = function() { /* do something */ }

Here are examples of different types of functions:

i) A simple function will be illustrated as:

```
var greet = function(person, greeting) {
    var text = greeting + ‘, ‘ + person;
    console.log(text);
};
greet(‘Rebecca’, ‘Hello’);
```

ii) A function that returns a value will be illustrated as:

```
var greet = function(person, greeting) {
    var text = greeting + ‘, ‘ + person;
    return text;
};
console.log(greet(‘Richard’,‘hello’));
```

iii) A function that returns another function will be illustrated as:

```
var greet = function(person, greeting) {
    var text = greeting + ‘, ‘ + person;
    return function() { console.log(text); };
};
```

```
var greeting = greet(‘Richard’, ‘Hello’);
greeting();
```

The Self-Executing Anonymous Functions

This is a common pattern in JavaScript today. The pattern creates a function expression, then immediately executes the function. This is a pattern that will be very useful if you do not want to create a mess in the global namespace with your code. Note that all the variables that will be declared inside of the function will be visible on the outside. This is how the self-executing anonymous function will look like:

```
(function(){
    var foo = 'Hello world';
})();
console.log(foo);   // undefined!
```

JavaScript functions as arguments

Functions are very important in JavaScript; they are treated as first-class citizens and this means that they can be assigned to variables with ease or they can be passed over to the other functions as arguments. The syntax for passing an anonymous function as an argument will be:

```
var myFn = function(fn) {
    var result = fn();
    console.log(result);
};
myFn(function() { return 'hello world'; });   // logs 'hello world'
```

while the syntax for passing a named function as an argument will be:

```
var myFn = function(fn) {
    var result = fn();
    console.log(result);
};
var myOtherFn = function() {
    return 'hello world';
};
myFn(myOtherFn);   // logs 'hello world'
```

Testing

There is a way one can test the type of variable in JavaScript. This is shown by use of typeof operator in order to determine the type of a specific value. Below is an illustration

of how one can test the type of various variables in programming:

```
var myFunction = function() {
console.log(‘hello’);
};

var myObject = {
foo : ‘bar’
};

var myArray = [ ‘a’, ‘b’, ‘c’ ];

var myString = ‘hello’;

var myNumber = 3;

typeof myFunction;   // takes back ‘function’
typeof myObject;     // takes back ‘object’
typeof myArray;      // takes back ‘object’ — careful!
typeof myString;     // takes back ‘string’;
typeof myNumber;     // takes back ‘number’

typeof null;         // takes back ‘object’ — careful!


if (myArray.push && myArray.slice && myArray.join) {
// probably an array
// (this is called “duck typing”)
}

if (Object.prototype.toString.call(myArray) === ‘[object Array]’) {
// Definitely an array!
```

// This is widely considered as the most robust way

// to determine if a specific value is an Array.

Scope

Scope is basically a variable that is available in a certain code at a given time. If you are unable to understand scope, you will have constant issues with debugging, therefore this is very important. When you declare a variable inside a function using the var keyword, it will only be available to the code inside of that function and the code outside the function will not be able to access the variable. Again, functions that have been defined inside that function will be able to access the declared variable.

Those variables that will be declared inside a function without the var keyword are not local to the function as this means that JavaScript will go back the scope chain up to the window scope in order to find the point where the variable was defined at first. If there was no declaration at the beginning, the variable will then be declared in the global scope, and this can have great expected consequences.

The example below shows functions that have access to variables defined in the same scope:

```
var foo = ‘hello’;


var sayHello = function() {
    console.log(foo);
};
sayHello();         // logs ‘hello’
console.log(foo);   // also logs ‘hello’
```

This example shows that a code outside the scope in which a variable was defined does not have access to the variable

```
var sayHello = function() {
    var foo = ‘hello’;
    console.log(foo);
};
sayHello();         // logs ‘hello’
console.log(foo);   // doesn’t log anything
```

Chapter 5: JavaScript Features

JavaScript is a very powerful programming language. It is quite popular as a client scripting language for different web browsers. It can be used in any web application in order to implement simple but very important features, roller of images. It has been used for many years now to add beautiful effects to web pages because of its powerful features, and this is something that makes it one of the best programming languages in the entire world today. Here are some of the best features that should prompt you to learn this great programming language:

JavaScript's browser support

With JavaScript, one does not need to install flash plugin in their browser, like it is required when one wants to access any flash content. This is because all browsers have fully accepted JavaScript as their main scripting language, therefore they provide an integrated support for it. What a programmer would have to do is just to handle some of the tasks that depend on DOM of different browsers properly and you will be free to use JavaScript on your browser.

JavaScript as a functional programming language

Programmers using JavaScript are at liberty to code in a functional programming style, which is easy and more exciting than any other style. This is because of a number of reasons. A function in JavaScript can be assigned to variables, just the same way as any other type of data. A function can also accept another function as its parameter. It can also return a function. You can also have functions that do not have any names as well. This is the ability that many programmers would love to enjoy as they work.

JavaScript can be used on both the client and on the server side

JavaScript has access to the document model object of any browser therefore one can easily change the structure of web pages at runtime. This is what gives users of the programming language more control over their browsers. You can use the language to add different effects to webpages. The programming language can also be used on the server side as well, for instance it is used in Alfresco to create web scripts. This is what makes it very easy for a programmer to add custom tasks to Alfresco.

Its ability to detect the user's browser and OS

This is something that will help a programmer to be able to perform operations that are dependent on the platform when it is required.

The ability to detect the user's browser and OS allows your script to perform platform-

dependent operations, if necessary.

JavaScript’s support for objects

JavaScript is a programing language that is oriented by objects but it handles objects and inheritance in a much different way when compared to how other programming languages that are object oriented do. What it does is that it offers immense support to object oriented concepts but then it remains simple to learn and use. That is why this programming language can be used to execute both the simple tasks and the complex tasks. This is also what has enables it to stay top in the list as one of the most preferred programming languages in the industry today. It is now the best language to learn for people who are interested in computer programming due to its support for object oriented concepts and function concepts. It is also easy to use and you only need a browser and a text editor to enjoy what it has to offer.

Conclusion

Many people believe that one has to actually go back to school in order to learn a programming language and other computer skills, but this is not true at all. The internet has brought a lot of possibilities in our lives and so much information is out there. This eBook can get you started in JavaScript without necessarily attending classes, and you can learn so much more with time to program like an expert.

Other people believe that programming languages are meant for certain kinds of people, which is also a myth. JavaScript is a programming language that can be learned by any kind of person, for any reason at all. You do not have to be a mathematic genius or a science guru to be able to program like an expert. This is something that anyone can do with so much ease. Like you have already seen, it is a very easy programming language to learn, which opens opportunities for all kinds of people.

An effective way to learn programming language is to learn the basics, then practice as much as you can. Practice is what makes one a pro and coding gets better with a lot of practice. You have to dedicate a considerable amount of time to it as well, for the skills to be well mastered. Check out what other people are doing and try new ways of programming and see how much you will have achieved in a short period of time.