JavaScript™

By
Agubata Odinaka

JavaScript Constants

JavaScript Functions

JavaScript Conditional Statements

Loops in JavaScript

JavaScript Arrays

Introduction

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. Simply put, JavaScript is a scripting language that adds interactivity to web pages.

JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript** but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java.

The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

JavaScript helps to reduce the amount of time the server will be queried by carrying out most of its validations on the client-side.

For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of using JavaScript

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.

- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Including JavaScript in HTML

There is a flexibility given to include JavaScript code anywhere in a HTML document. However the most preferred ways to include JavaScript in a HTML file are as follows:

- ❑ Script in <head>...</head> section.
- ❑ Script in <body>...</body> section.
- ❑ Script in <body>...</body> and <head>...</head> sections.
- ❑ Script in an external file and then include in <head>...</head> section.

Regardless of where it is included in a HTML file, JavaScript can be implemented using JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
   JavaScript code
</script>
```

For internal JavaScript code, the script will have the “type” attribute:

- ❖ type- This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

The example on the next slide shows javaScript code placed in the head and body tags of a HTML file.

```
<html>
        <head>
                <title>JavaScript Example</title>
                <script type="text/javascript">
                        document.write("Execute during page load from the head<br>");
                </script>
        </head>
        <body>
                <script type="text/javascript">
                        document.write("Execute during page load from the body<br>");
                </script>
        </body>
</html>
```

It is advantageous to group common functions in an external JavaScript file.

This permits the reuse of the functions in the file in multiple HTML pages.

For external JavaScript code, the script will have both the “type” attribute and the “src” attribute:

❖ src- The location of an external scripting file.

The *src* attribute specifies that the code is actually found in a file which should be loaded and then executed. The .js extension is normally used for JavaScript code files. The following example illustrates the use of these attributes.

JavaScript™

Create a new JavaScript file and call it script.js. Write the following JavaScript code inside the file and save:

```
document.write(“234”);
```

To include the file you just created in your HTML page, do the following:

```
<html>
        <head>
                <title>JavaScript Example</title>
                <script type="text/javascript" src=“script.js"> </script>
        </head>
        <body>
        </body>
</html>
```

Things to know about JavaScript

White Lines and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
var1 = 10

var2 = 20
```

But when formatted in a single line as follows, you must use semicolons:

```
var1 = 10; var2 = 20;
```

Note: It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE: Giving a variable and a function the same name in JavaScript may cause conflict in code.

Loosely Typed-Language

JavaScript is **a loosely-typed** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

```
var money=”twenty”;
var money=20;
```

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- ➢Writing into an alert box, using **window. alert()**.
- ➢Writing into the HTML output using **document. write()**.
- ➢Writing into the browser console, using **console.log()**.

Using Window.alert()

You can use an alert box to display data:

```
<html>
    <body>
            <h1>My First Web Page</h1>
            <p>My first paragraph.</p>
            <script type="text/javascript">
                    window.alert(“window alert”);
            </script>
    </body>
</html>
```

Using document.write()

For testing purposes, it is convenient to use **document.write()**:

```
<html>
	<body>
		<h1>My First Web Page</h1>
		<p>My first paragraph.</p>
		<script type="text/javascript">
			document.write("document write");
		</script>
	</body>
</html>
```

Using console.log()

In your browser, you can use the **console.log()** method to display data. Activate the browser console with F12, and select “web Console" in the menu.

```
<html>
        <body>
                <h1>My First Web Page</h1>
                <p>My first paragraph.</p>
                <script type=“text/javascript”>
                        console.log(“console log”);
                </script>
        </body>
</html>
```

JavaScript Data Types

There are six data types in JavaScript :

- **Numbers** – Integer or floating point numbers
- **Booleans** – Either true/false or a number (0 being false) can be used for boolean values
- **Strings** – Sequence of characters enclosed in a set of single or double quotes
- **Objects** – Entities that typically represents elements of a HTML page
- **Null** – No value assigned which is different from a 0
- **Undefined** – Is a special value assigned to an identifier after it has been declared but before a value has been assigned to it

JavaScript is a dynamically typed language. The data type of the identifier(variable or constant) is not assigned when the identifier is declared. When a value is assigned to the identifier the identifier takes on that type. The data type of the variable is not important until an operator is applied to the variable. The behavior of the operator is dependent of the data type being acted upon.

For example:

```
<script type="text/javascript">
        var a = "Sally";
        var b = 34;
        var c=20;
        window.alert(a+b);
        window.alert(b+c);
<script/>
```

It is better if we are consistent when assigning a data type to a variable. This leads to less confusing code.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container. Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
var money;
```

JavaScript keywords like “else”, ”break” etc. cannot be used as variable names.

JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For

Example:

```
var _34man=“Onyewu”;
var dog=“Tracy”;
```

Variable Scope

The scope of a variable is the region of your program in which it is defined.

JavaScript variables have only two scopes.

❖ **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.

❖ **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

```
<script type="text/javascript">
var myVar = "global";    // Declare a global variable
function checkscope( ) {
     var myVar = "local";       // Declare a local variable
     document.write(myVar);
}
checkscope( );
</script>
```

JavaScript

JavaScript Operators

JavaScript supports the following types of operators:

- ❑ Arithmetic Operators
- ❑ Comparison Operators
- ❑ Logical (or Relational) Operators
- ❑ Assignment Operators
- ❑ Conditional (or ternary) Operators

Arithmetic Operators

JavaScript supports the following arithmetic operators:

Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Example:

```
<script type="text/javascript">
        var a=10;
        var b=3;
        var result;
        result=a+b;
        document.write("a+b= "+result+"<br>");
        result=a-b;
        document.write("a-b= "+result+"<br>");
        result=a*b;
```

```
    document.write("a*b= "+result+"<br>");
    result=a/b;
    document.write("a/b= "+result+"<br>");
    result=a%b;
    document.write("a%b= "+result+"<br>");
    a++
    document.write("a++= "+a+"<br>");
    b--;
    document.write("b--= "+b+"<br>");
</script>
```

Comparison Operators

JavaScript supports the following comparison operators:

Operator	Description
==	Equal
===	Identical
!=	Not equal
!==	Not identical
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator

Example:

```
<script type="text/javascript">
        var a=10;
        var b=3;
        var result;
        result=(a==b);
        document.write(“a==b "+result+"<br>");
        result=(a!=b);
        document.write(“a!=b "+result+"<br>");
        result=(a>b);
        document.write(“a>b "+result+"<br>");
        result=(a<b);
        document.write("a<b= "+result+"<br>");
        result=(a==b);
        result ? document.write(“true”): document.write(“false”);
</script>
```

Logical Operators

JavaScript supports the following Logical operators:

Operator	Description
&&	Logical and
||	Logical or
!	Logical not

Example:

```
<script type="text/javascript">
        var a=true;
        var b=false;
        result=(a&&b);
        document.write(result);
        result=(a||b);
        document.write(result);
        result=!(a||b);
        document.write(result);
</script>
```

Assignment Operators

JavaScript supports the following assignment operators:

Operator	Description
=	Simple Assignment C=A+B
+=	Add and assignment C+=A => C=C+A
-=	Subtract and assignment C-=B => C=C-B
*=	Multiply and assignment C*=A => C=C*A
/=	Divide and assignment C/=B => C=C/B
%=	Modulus and assignment C%=A => C=C%A

Example:

```
<script type="text/javascript">
        var a=10;
        var b=3;
        document.write(a+”<br>”);
        a+=b;
        document.write(a+”<br>”);
        a-=b;
```

```
     document.write(a+”<br>”);
     a*=b;
     document.write(a+”<br>”);
     a/=b;
     document.write(a+”<br>”);
      a%=b;
    document.write(a+”<br>”);
</script>
```

Conditional Operators

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator	Description
?: (conditional)	If Condition is true? Then value X : Otherwise value Y

Example:

```
<script type="text/javascript">
    var a=10;
    var b=3;
    result = (a > b) ? 100 : 200;
    document.write(result);
<script/>
```

JavaScript Constants

In JavaScript, constants are declared with “const” keyword and assigned at the time of the declaration. A constant can be global or local to a function where it is declared.

Constants are read-only, therefore you can not modify them later on.

Naming a constant in JavaScript follows the same rule of naming a variable except that the ”const” keyword is always required, even for global constants.

If the keyword is omitted, the identifier is assumed to represent a variable.

Example:

```
<script type="text/javascript">
    const pi=3.142;
<script/>
```

You cannot declare a constant with the same name as a function

or variable in the same scope. The following statements create error.

Example:

```
    <script type="text/javascript">
        function abc()  {
            const abc = 15;
            var abc;
        }
<script/>
```

JavaScript Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Creating/defining a function

The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

```
<script type="text/javascript">
        function functionName(parameter_list){
                …
        }
</script>
```

Example:

```
<script type="text/javascript">
        function sayHello(){
                window.alert ("Hello there!");
        }
</script>
```

Calling a function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
        <head>
        </head>
        <body>
                <p>Click the following button to call the function</p>
                <input type="button" onclick="sayHello()" value="Say Hello">
        </body>
</html>
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

```
<script type="text/javascript">
    function sayHello(name, age){
        window.alert (name + " is " + age + " years old.");
    }
</script>
```

JavaScript

```
<html>
        <head>
        </head>
        <body>
                <p>Click the following button to call the function</p>
                <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
                <p>Use different parameters inside the function and then try...</p>
        </body>
</html>
```

The Return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

```
<script type="text/javascript">
        function multiply(first, last){
                var full;
                full = first * last;
                return full;
        }
        function secondFunction(){
                var result;
                result = multiply(2, 5);
                window.alert (result );
        }
<script/>
```

JavaScript

```
<script type="text/javascript">
        function multiply(first, last){
                var full;
                full = first * last;
                return full;
        }
        function secondFunction(){
                var result;
                result = multiply(2, 5);
                window.alert (result );
        }
<script/>
```

JavaScript

```
<html>
        <head>
        </head>
        <body>
                <p>Click the following button to call the function</p>
                <input type="button" onclick=“secondFunction(){" value="Say Hello">
                <p>Use different parameters inside the function and then try...</p>
        </body>
</html>
```

JavaScript Conditional Statements

In JavaScript we have the following conditional statements:

❑ **If statement-** used to specify a block of code to be executed, if a specified condition is true

❑ **else statement-** used to specify a block of code to be executed, if the same condition is false

❑ **else if statement**- used to specify a new condition to test, if the first condition is false

❑ **Switch-**used to specify many alternative blocks of code to be executed

If Statement

Use the **if** statement to specify a block of JavaScript code to be executed

if a condition is true.

Syntax

```
if (condition) {
   block of code to be executed if the condition is true
}
```

Example:

```
<script type="text/javascript">
    var check=1;
    if (check < 10) {
       window.alert(“Okocha”);
    }
</script>
```

else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   block of code to be executed if the condition is true
} else {
   block of code to be executed if the condition is false
}
```

Example:

```
<script type="text/javascript">
    var check=1;
    if (check > 10) {
       window.alert(“Okocha”);
    }else{
        window.alert(“Kanu Nwankwo”);
    }
</script>
```

JavaScript

else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  block of code to be executed if condition1 is true
} else if (condition2) {
  block of code to be executed if the condition1 is false and condition2 is true
} else {
  block of code to be executed if the condition1 is false and condition2 is false
}
```

Example:

```
<script type="text/javascript">
    var check=12;
    if (check < 10) {
       window.alert(“Okocha”);
    }else if(check>15){
            window.alert(“Kanu Nwankwo”);
    }else{
            window.alert(“Jordan”);
    }
</script>
```

Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        default code block
}
```

This is how it works:

The switch expression is evaluated once.

The value of the expression is compared with the values of each case.

If there is a match, the associated block of code is executed. If there is no match, the default statement is executed.

Example:

```
<script type="text/javascript">
var check=9;
    switch (check) {
       case 0:
          window.alert(“case 0”);
          break;
       case 5:
          window.alert(“case 5”);
          break;
```

Example:

```
          case 9:
            window.alert(“case 9”);
            break;
           default:
             window.alert(“default”);
          }
        </script>
```

Loops in JavaScript

JavaScript supports different kinds of loops:

- ❑ **for** - loops through a block of code a number of times
- ❑ **for/in** - loops through the properties of an object
- ❑ **while** - loops through a block of code while a specified condition is true
- ❑ **do/while** - also loops through a block of code while a specified condition is true

For Loop

The for loop is often the tool you will use when you want to create a loop. The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
   code block to be executed
}
```

Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

Example:

```
<script type="text/javascript">
    for (var counts = 0; counts < 5; counts++) {
       window.alert(counts);
    }
    </script>
```

For/in Loop

The JavaScript for/in statement loops through the properties of an object. The for/in loop has the following syntax:

```
for (statement ) {
    code block to be executed
}
```

Example:

```
<script type="text/javascript">
    var person = {name:"John", surname:"Doe", age:25};
    var i;
    for (i in person) {
       window.alert(person[i]);
    }
    </script>
```

while Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
   code block to be executed
}
```

Example:

```
<script type="text/javascript">
    var i=7;
    while (i < 10) {
       window.alert(i);
       i++;
    }
    </script>
```

Do/while Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
   code block to be executed
}
while (condition);
```

Example:

```
<script type="text/javascript">
    var i=7;
    do {
       window.alert(i);
       i++;
    }
    while (i < 10);
    </script>
```

Loop Control

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the look.

To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The Break Statement

The **break** statement, which was briefly introduced with the *switch* statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Example:

```
<script type="text/javascript">
    var x = 1;
    while (x < 20){
        if (x == 5){
                break; // breaks out of loop completely
        }
        window.alert(x);
        x++;
    }
</script>
```

The Continue Statement

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a **continue** statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example:

```
<script type="text/javascript">
    var x = 1;
    while (x < 7){
        if (x == 5){
                continue; // breaks out of loop completely
        }
        window.alert(x);
        x++;
    }
</script>
```

NB: this code may have a problem displaying because of the limitations of the window.alert display.

JavaScript Arrays

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
var car1 = "Saab";
var car2 = "Volvo";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes** and if you use a named index, JavaScript will redefine the array to a standard object.

After that, **all array methods and properties will produce incorrect results**.

Creating an Array

Arrays in JavaScript can be created in two different ways:

- ❑ Using an array literal.
- ❑ Using the javascript “new” keyword.

Using an Array Literal

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

var *array-name* = [*item1*, *item2*, ...];

Example:

var nums = [1,2,3];

Using the JavaScript Keyword "new"

Syntax:

var *array-name* = new Array(*item1*, *item2*, ...);

Example:

var nums = new Array(1,2,3);

NB: The two examples above do exactly the same. There is no need to use new Array().

For simplicity, readability and execution speed, use the first one (the array literal method).

Accessing the Elements of an Array

You refer to an array element by referring to the **index number**.

This statement accesses the value of the first element in nums:

```
var name = nums[0];
```

This statement modifies the first element in cars:

```
nums[0] = 10;
```

The elements of an array can easily be accessed with a for loop.

Example:

```
<script type="text/javascript">
var cars = ["Saab", "Volvo", "BMW"];
var len =cars.length;
for(var a=0; a<len; a++){
        window.alert(cars[a]);
}
</script>
```

Questions?

JavaScript™