# Basic Javascript

Learning the basics of web development?
Start with Basic Javascript.

Krishna Prasad Timilsina

bikranshu.t@gmail.com

**JavaScript** is a ***client-side*** as well as a server-side scripting language that can be inserted into HTML pages and is understood by web browsers.

The ***script*** element can either contain JavaScript directly (**internal**) or link to an external resource via a src attribute (**external**).
Eg.

```
<script>
        alert("Hello, world.");  // JavaScript code here.
</script>

<script src="script.js"></script>
```

**Points to Remember :**

- ➢ JavaScript code must be written within the ***<script>*** tag.
- ➢ ***HTML5*** standard does not require ***type="text/javascript"*** attribute, whereas prior html standards require type attribute.
- ➢ The ***<script>*** tag can be added into the ***<head>*** or ***<body>*** tag.
- ➢ The script included in ***<head>*** tag may not be able to access DOM elements because ***<head>*** loads before ***<body>***. Write script before ending of ***</body>*** tag if script code needs to access DOM elements.
- ➢ JavaScript is a ***case sensitive*** scripting language. It means functions, variables and keywords are case sensitive. For example, VAR is different from var, John is not equal to john.

**Variable:**

- Variable is a container for storing data values.
- Variables can be used to store strings and numbers.
- JavaScript uses the reserved keyword ***var*** to declare a variable.
- A variable must have a unique name.

Syntax:

```
var variableName;
```

There are two parts to creating a variable; ***declaration*** and ***initialization***. Once it's created, you can assign (or set) its value.
Eg.

```
var name;                    // A single declaration
var name, age, gender;  // Multiple declarations with a single var keyword
var name = "Tom";       // Variable declaration and initialization
var name =  "Tom", age = 20;  // Variable declaration and initialization in one statement
name = "Andy";          // Variable assignment
```

***Note:*** The ***equal sign (=)*** is used in JavaScript statements to assign values to variables: it is the ***assignment operator***.

There are two types of variables in JavaScript : ***local variable*** and ***global variable***.

- ➢ A JavaScript ***local variable*** is declared inside a block or function. It is accessible within the function or block only.

  Eg.

```
function abc(){
        var x = 10;  // local variable
}
```

- ➢ A JavaScript ***global variable*** is accessible from everywhere in JavaScript code. A variable i.e. declared outside the function or declared with a window object is known as a global variable.

  Eg.

```
var data = 200;  // global variable
function a(){
        document.writeln(data);    // accessing global variable
}

window.age = 90;   // global variable
function n(){
        alert(window.age);  // accessing global variable
}
```

***Note:***

- A ***local*** variable takes precedence over a ***global*** variable with the same name.
- Function parameters are always ***local*** to that function.

**Data Types:**

Data Type indicates characteristics of data. It tells the compiler whether the data value is numeric, alphabetic, date etc.so that it can perform the appropriate operation.

**1.Primitive Data Types:**

| **DataType** | **Description** |
|---|---|
| string | represents a sequence of characters e.g. "hello" |
| number | represents numeric values e.g. 100 |
| boolean | represents boolean value either false or true |
| undefined | represents undefined value |
| null | represents null i.e. no value at all |

**String:**

The JavaScript string is an object that represents a sequence of characters. The string may contain letters,words, space, numbers, symbols. JavaScript string must be enclosed in double quotation marks ***(")*** or single quotation mark ***(')***.

**I. By string literal**

Syntax:

***var variableName = "string value";***

Eg.

```
var x = "This is string literal";  // typeof x will return string
```

**II. By using an Object constructor**

Syntax:

***var variableName = new String("string value");***

Eg.

```
var y = new String("hello string");  // typeof y will return object
```

**Number:**

The JavaScript number object enables you to represent a numeric value. It may be an integer or floating-point. Numbers must NOT be wrapped in quotation marks.

**I. By string literal**

Syntax:

***var variableName =  number;***

Eg.

```
var x = 102;    // integer value  // typeof x returns number
var y = 102.7;    // floating point value
var z = 13e4;    // exponent value, output: 130000
```

**II. By using an Object constructor**

Syntax:

***var variableName = new Number(value);***

Eg.

```
var z = new Number(16);    // typeof z returns object
```

***Note:*** If a ***number*** can't be converted to a number(such as ***undefined***), it returns ***NaN(Not a Number)*** that can be checked by the **isNaN*()*** method.

**Boolean:**

A JavaScript Boolean is an object that represents value in two states: true or false. The default value of a JavaScript Boolean object is false.  If a value parameter is ***omitted*** or is ***0***, ***-0***, ***null***, ***false***, ***NaN***, ***undefined***, or ***the empty string ("")***, the object has an initial value of ***false***.

**I. By string literal**

Syntax:

***var variableName = value;***

Eg.

```
var x = false;  // typeof x returns boolean
```

**II. By using an Object constructor**

Syntax:

***var variableName = new Boolean(value);***

Eg.

```
var y = new Boolean(false); // typeof y returns object
```

Instead of using the words ***true*** and ***false***, JavaScript also allows you to use the number ***1*** for ***true*** and the number ***0*** for ***false***.

Eg.

```
var x = 1;  // Using the number 1 is the same as using the value of true
var y = 0;  // Using the number 0 is the same as using the value of false
```

**Undefined:**

A variable without a value has the value ***undefined***. The ***typeof*** is also ***undefined***. The meaning of undefined ***"value is not assigned"***. If a variable is declared, but not assigned, then its value is exactly ***undefined***

Eg.

```
var car;    // value is undefined, type is undefined
```

**Null:**

A ***null*** means that the variable has no value. It is not a space, nor is it a zero; it is simply nothing.

Syntax:

***var variableName = null;***

Eg.

```
var myVar = null;
```

***null*** is assigned to a variable ***myVar***. It means we have declared a variable but have not assigned any value yet, so the value is absent.

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = null; // Now value is null, but the type is still an
object

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
person = undefined; // Now both value and type is undefined
```

***Note:*** *console.log(*typeof null); *// object*

**Q. Difference between undeclared and undefined variables?**
***Undeclared*** variables are those that do not exist in a program and are not ***declared***. If the program tries to read the value of an ***undeclared variable***, then a runtime error is encountered.

***Undefined*** variables are those that are ***declared*** in the program but have not been given any value. If the program tries to read the value of an ***undefined variable***, an undefined value is returned.

**2. Non-primitive Data Type:**

***1. Array:***

An array is a special type of variable, which can hold more than one value at a time. Array indexes start with 0.

**I. By array literal**

Syntax:

***var arrayName = [item1, item2, ...];***

Arrays use numbers to access its "elements". You access an array with the following syntax:

***arrayName[indexNumber]***

Eg.

```
var person = ["John", "Doe", 46];
console.log( person[0])  // John
```

**II. By creating an instance of Array directly (using new keyword)**

Syntax:

***var arrayName = new Array();***

Eg.

```
var employee = new Array();
employee[0] = "Arun";
employee[1] = "Varun";
employee[2] = "John";
```

**III. By using an Array constructor (using new keyword)**

Eg.

```
var employee = new Array("Jai","Vijay","Smith");
```

***Note:*** The ***length*** property of an array returns the length of an array (the number of array elements).

***2. Object:***

Objects are variables too. But objects can contain many values. A JavaScript object is a collection of named properties and methods - a function.

**I. By object literal**

Syntax:

***var objectName = {};*** *// an empty object*

***var objectName = {propertyName: propertyValue, …};***

You can access object properties with the following syntax:

***objectName.propertyName***

**OR**

***objectName["propertyName"]***

Eg.

```
var person = {firstName:"John", lastName:"Doe", age:46};
console.log(person.firstName);  // John
```

You can add new properties to an existing object by simply giving it a value.

Eg.

```
person.nationality = "Nepali";
```

The ***delete*** keyword deletes a property from an object. The ***delete*** keyword deletes both the value of the property and the property itself.

Eg.

```
delete person.age;   // or delete person["age"];
```

You access an object method with the following syntax:

***objectName.methodName()***

Eg.

```
var user = {
        name: "Yoda",
        age: 899,
        talk: function () { alert("Hello"); }
    };

user.talk();
```

***Note:*** If you access a method without ***()***, it will return the function definition.

You can add a new method to an existing object with the following syntax:

Eg.

```
user.fullName = function () {
        return “Ram Sharma”;
```

```
};
```

II. **By using an Object constructor**

Syntax:

***var objectName = new Object();***

Eg.

```
var employee = new Object();
employee.id = 101;
employee.name = "Ram Sharma;
employee.salary = 50000;
```

JavaScript does **not** support arrays with **named indexes**.
In JavaScript, **arrays** always use **numbered indexes**.
In JavaScript, **objects** always use **named indexes**.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

**Adding Variables to Text Strings:**

You use the ***addition operator (+)*** to add the value of the variable to the string.

Eg.

```
<script type="text/javascript">
   var car="Corvette";
   document.write("I like driving my " + car);
   document.write("I like driving my "+ car +" every day!");
</script>
```

**Function:**

Functions are reusable blocks of code that carry out a specific task. To create a function, use the ***function*** keyword.

***1. Function Declaration (aka Function Statement) Syntax:***


Eg.

```
function sum (a, b) {
    return a + b;
```

```
};
```

***a*** and ***b*** are the function's ***parameters***, and the value it returns is signified by the ***return*** keyword. The ***return*** keyword also stops execution of the code in the function; nothing after it will be run.

```
console.log(sum(1, 2));  // 3
```

***2. Function Expression (aka Function Literal) Syntax:***

JavaScript allows us to assign a function to a variable and then use that variable as a function. It is called function expression.

Eg.

```
var add = function (x, y) {
    return x + y;
};

console.log(add(2, 3));   // 5
```

**Q. Which Is Better: A Function Declaration or a Function Expression?**

They are basically the same, but ***function declarations*** have two advantages over function expressions:

- They are ***hoisted***, so you can call them before they appear in the source code.
- They have a name, so it is useful for debugging.

***Arguments Object:***

All the functions in JavaScript can use arguments by default. An argument object includes the value of each parameter.

The arguments object is an array-like object. You can access its values using an index similar to an array. However, it does not support array methods.

Eg.

```
function showMessage(firstName, lastName) {
    console.log("Hello " + firstName + " " + lastName);

    console.log("Hello " + arguments[0] + " " + arguments[1]);

    for(var i = 0; i < arguments.length; i++){
        console.log(arguments[i]);
    }
}
showMessage("Bill", "Gates");
showMessage(100, 200);
```

***Note:*** JavaScript is a loosely-typed language. A function parameter can hold values of any data type.

**Q. "Parameter" Versus "Argument"**

***Parameters*** are used to define a function. They are also called formal parameters and formal arguments. In the following example, param1 and param2 are parameters:

```
function foo(param1, param2) {
    // code goes here
}
```

***Arguments*** are used to invoke a function. They are also called actual parameters and actual arguments. In the following example, 3 and 7 are arguments:

```
foo(3, 7);
```

**Operators:**

JavaScript operators are symbols that are used to perform operations on operands.

**1. Arithmetic Operators**

| Operator | Description | Example |
|---|---|---|
| + | Addition | 10+20 = 30 |
| - | Subtraction | 20-10 = 10 |
| * | Multiplication | 10*20 = 200 |
| / | Division | 20/10 = 2 |
| % | Modulus (Remainder) | 20%10 = 0 |
| ++ | Increment | var a=10; a++; Now a = 11 |
| -- | Decrement | var a=10; a--; Now a = 9 |

**2. Comparison Operators**

| Operator | Description | Example |
|---|---|---|
| == | Is equal to | 10 == 20 = false |
| === | Identical (equal and of same type) | 10 == 20 = false |
| != | Not equal to | 10 != 20 = true |

| | | |
|---|---|---|
| !== | Not Identical | 20 !== 20 = false |
| > | Greater than | 20 > 10 = true |
| >= | Greater than or equal to | 20 >= 10 = true |
| < | Less than | 20 < 10 = false |
| <= | Less than or equal to | 20 <= 10 = false |

**3. Logical Operators**

| **Operator** | **Description** | **Example** |
|---|---|---|
| && | Logical AND | (10 == 20 && 20 == 33) = false |
| \|\| | Logical OR | (10 == 20 \|\| 20 == 33) = false |
| ! | Logical Not | !(10 == 20) = true |

**4. Assignment Operators**

| **Operator** | **Description** | **Example** |
|---|---|---|
| = | Assign | 10+10 = 20 |
| += | Add and assign | var a = 10; a+=20; Now a = 30 |
| -= | Subtract and assign | var a = 20; a-=10; Now a = 10 |
| *= | Multiply and assign | var a = 10; a*=20; Now a = 200 |
| /= | Divide and assign | var a = 10; a/=2; Now a = 5 |
| %= | Modulus and assign | var a = 10; a%=2; Now a = 0 |

**5. Special Operators**

| **Operator** | **Description** |
|---|---|
| (?:) | Conditional Operator returns value based on the condition. It is like if-else. |
| , | The Comma Operator allows multiple expressions to be evaluated as single statements. |
| delete | Delete Operator deletes a property from the object. |

| | |
|---|---|
| in | In Operator checks if object has the given property |
| instanceof | checks if the object is an instance of given type |
| new | creates an instance (object) |
| typeof | checks the type of object. |
| void | it discards the expression's return value. |
| yield | checks what is returned in a generator by the generator's iterator. |

Eg.

```
console.log(typeof 4.5)  // number
console.log(typeof "x")  // string
```

**Hoisting:**

In JavaScript, variable and function names can be used before declaring it. The JavaScript compiler moves all the declarations of variables and functions at the top so that there will not be any error. This is called ***hoisting***.

Eg.

```
x = 1;
console.log('x = ' + x);  // display x = 1
var x;
```

Hoisting is only possible with declaration but not the initialization. JavaScript will not move variables that are declared and initialized in a single line.

Eg.

```
console.log('x = ' + x);  // display x = undefined
var x = 1;
```

The JavaScript compiler moves the function definition at the top in the same way as variable declaration.

Eg.

```
console.log(sum(5, 5));  // 10
function sum(val1, val2)
{
        return val1 + val2;
}
```

***Note:*** JavaScript compiler does not move function expression.

**Loops:**

**1. for loop**

Syntax:

```
for (initialization; condition; increment)  {
    // code to be executed
}
```

Eg.

```
for (var i = 0; i < 9; i++) {
  console.log(i);
}
```

**2. for...in**

Syntax:

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

Eg.

```
var numObject = {a: 1, b: 2, c: 3};

for (var prop in numObject) {
  console.log("The property '" + prop + "' is " + numObject[prop]);
}
```

**3. Switch**

Eg.

```
switch (new Date().getDay()) {
        case 0:
        day = "Sunday";
        break;
        case 1:
        day = "Monday";
        case 2:
        day = "Tuesday";
        break;
        default:
        day = "Looking forward to the Weekend";
}
```

**Prototype:**

The prototype is an object that is associated with every ***function and object*** by default in JavaScript, where the function's prototype property is ***accessible and modifiable*** and the object's prototype property (aka attribute) is ***not visible***.

Every object which is created using ***literal syntax or constructor syntax with the new keyword***, includes ***__proto__*** property that points to the prototype object of a function that created this object.

Object's prototype property is invisible. Use ***Object.getPrototypeOf(obj)*** method instead of ***__proto__*** to access prototype objects.

***Note:*** The prototype property is a special type of enumerable object which cannot be iterate using ***for..in*** or ***foreach*** loop.

Use of Prototype Object:

1. Find properties and methods of an object
2. Implement inheritance in JavaScript.

**A Single Thread:**

The main JavaScript runtime is ***single threaded***. Two functions can't run at the same time. The runtime contains an ***Event Queue*** which stores a list of messages to be processed. There are no race conditions, no deadlocks. However, the code in the Event Queue needs to run fast. Otherwise the browser will become ***unresponsive*** and will ask to ***kill the task***.

**Simple Logging:**

The console API includes the following logging methods:

***1. console.error(object1, object2?, ...)***

Log the parameters to the console. In browsers, the logged content may be marked by an "error" icon and/or include a stack trace or a link to the code.

***2. console.info(object1?, object2?, ...)***

Log the parameters to the console. In browsers, the logged content may be marked by an "info" icon and/or include a stack trace or a link to the code.

***3. console.log(object1?, object2?, ...)***

Log the parameters to the console. If the first parameter is a printf-style format string, use it to print the remaining parameters.

***4. console.trace()***

Logs a stack trace (which is interactive in many browsers).

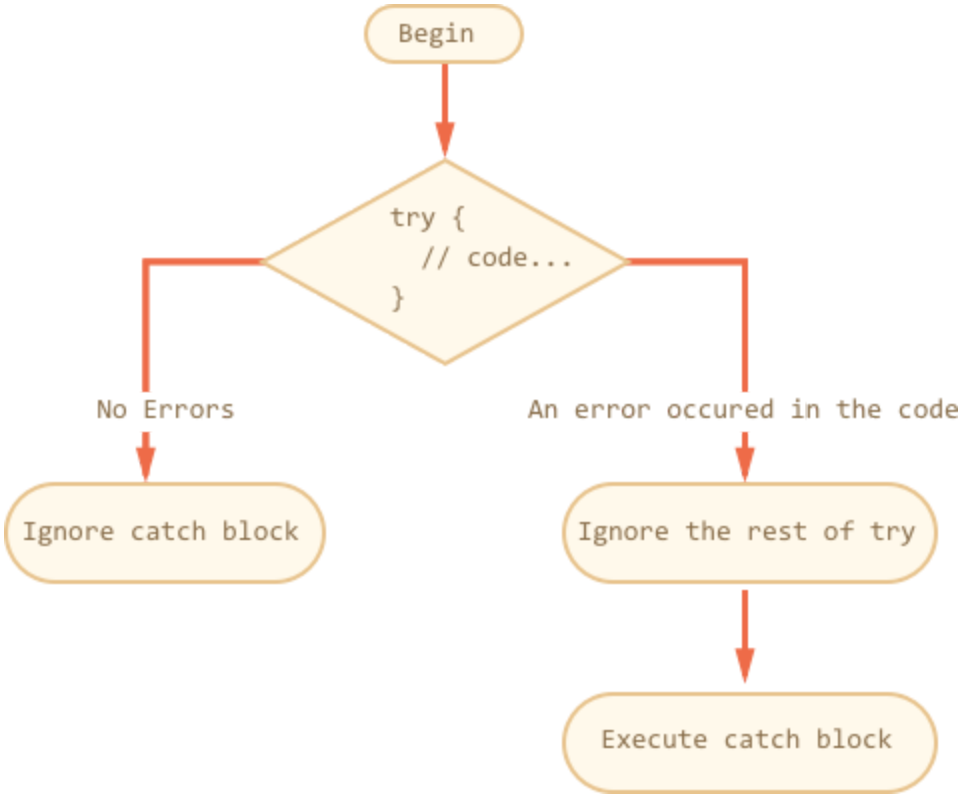
***5. console.warn(object1?, object2?, ...)***

Log the parameters to the console. In browsers, the logged content may be marked by a "warning" icon and/or include a stack trace or a link to the code.

**Error Handling, "try..catch":**

Syntax:

```
try {
    // code...
} catch (error) {
    // error handling
}
```




- First, the code in ***try {...}*** is executed.
- If there were no errors, then ***catch(error)*** is ignored: the execution reaches the end of ***try*** and then jumps over ***catch***.
- If an error occurs, then ***try*** execution is stopped, and the control flows to the beginning of ***catch(error)***. The ***error*** variable (can use any name for it) contains an error object with details about what's happened.

Error objects have the following properties:

- ➢ ***message*** – the human-readable error message.
- ➢ ***name*** – the string with error name (error constructor name).
- ➢ ***stack*** (non-standard) – the stack at the moment of error creation.

Eg.

```
try {
    lalala;   // error, variable is not defined!
} catch(err) {
  console.log(err.name);   // ReferenceError
  console.log(err.message);   // lalala is not defined
```

```
    console.log(err.stack);   // ReferenceError: lalala is not defined at ...
    // Can also show an error as a whole
    // The error is converted to string as "name: message"
    console.log(err);   // ReferenceError: lalala is not defined
}
```

**Data Storage:**

***1. Cookies***

HTTP cookies, commonly just called cookies, were originally intended to store session information on the client.

Link: https://github.com/js-cookie/js-cookie

***2. sessionStorage:***

The ***sessionStorage*** object stores data only for a session, meaning that the data is stored until the browser is closed. This is the equivalent of a session cookie that disappears when the browser is closed. Data stored on sessionStorage persists across page refreshes and may also be available if the browser crashes and is restarted, depending on the browser vendor.

Eg.

```
sessionStorage.setItem("name", "Nicholas");  // store data using method
sessionStorage.book = "Professional JavaScript"; // store data using property

var name = sessionStorage.getItem("name");  // get data using method
var book = sessionStorage.book;  // get data using property

delete sessionStorage.name;  // use delete to remove a value
sessionStorage.removeItem("book");  // use method to remove a value
```

You can iterate over the values in sessionStorage using a combination of the length property and key() method.

Eg.

```
for (var i=0, len = sessionStorage.length; i < len; i++){
   var key = sessionStorage.key(i);
   var value = sessionStorage.getItem(key);
   alert(key + "="" + value);
}
```

***3. localStorage:***

The ***localStorage*** object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Eg.

```
localStorage.setItem("name", "Nicholas");  // store data using method
localStorage.book = "Professional JavaScript";  // store data using property
```

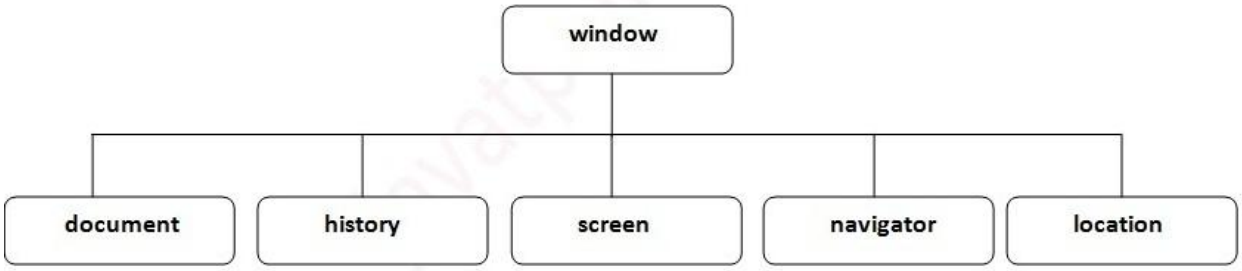
```
var name = localStorage.getItem("name");  // get data using method
var book = localStorage.book;  // get data using property

delete localStorage.name;  // use delete to remove a value
localStorage.removeItem("book");  // use method to remove a value
```

**Browser Object Model:**

The ***Browser Object Model (BOM)*** is used to interact with the browser. The default object of the browser is ***window,*** which means you can call all the functions of window by specifying window or directly.

Eg.

```
window.alert("hello javascript");
alert("hello javascript");
```




**1. Window Object**

The window object represents a window in the browser. An object of the window is created automatically by the browser.

Window is the object of the browser, it is not the object of javascript. The javascript objects are string, array, date etc.

| **Method** | **Description** |
|---|---|
| alert() | displays the alert box containing a message with the ok button. |
| confirm() | displays the confirm dialog box containing a message with ok and cancel buttons. |

| | |
|---|---|
| prompt() | displays a dialog box to get input from the user. |
| open() | opens the new window. |
| close() | closes the current window. |

Eg.

```
alert("Hello Alert Box");

var v= confirm("Are u sure?");
if(v==true){
    alert("ok");
}  else{
    alert("cancel");
}

var v= prompt("Who are you?");
alert("I am "+v);

open("http://www.javascript.com");
```

**Timing Events:**

The window object allows execution of code at specified time intervals.

- *setTimeout(function, milliseconds);*
  Executes a function only once, after waiting a specified number of milliseconds.
- *setInterval(function, milliseconds);*
  Same as setTimeout(), but repeats the execution of the function continuously.

Eg.

```
setTimeout(
    function(){
        alert("Welcome to javascript after 2 seconds")
    }, 2000);

setInterval(
    function(){
            alert("Welcome to javascript after 2 seconds")
    }, 2000);
```

**Stop the Timing Events Execution:**

The *clearTimeout()* method stops the execution of the function specified in setTimeout().

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

The *clearInterval()* method stops the executions of the function specified in the setInterval() method.

```
myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```

**2. History Object**

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

The history object is the window property, so it can be accessed by:

***window.history***

OR

***history***

| **Method** | **Description** |
|---|---|
| forward() | loads the next page. |
| back() | loads the previous page. |
| go() | loads the given page number. |

Eg.

```
history.back();   // for previous page
history.forward();   // for next page
history.go(2);   // for next 2nd page
history.go(-2);   // for previous 2nd page
```

**3. Navigator Object**

The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

The navigator object is the window property, so it can be accessed by:

***window.navigator***

OR

***navigator***

Eg.

```
console.log(navigator);
console.log(navigator.appCodeName);
console.log(navigator.appName);
console.log(navigator.appVersion);
console.log(navigator.cookieEnabled);
```

```
console.log(navigator.language);
console.log(navigator.userAgent);
console.log(navigator.platform);
console.log(navigator.onLine);
```

**4. Screen Object**

The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc

The screen object is the window property, so it can be accessed by:

***window.screen***

OR

***screen***

Eg.

```
console.log(screen);
console.log(screen.width);
console.log(screen.height);
console.log(screen.availWidth);
console.log(screen.availHeight);
console.log(screen.colorDepth);
console.log(screen.pixelDepth);
```

**5. Window Location:**

The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.

- ***window.location.href*** returns the href (URL) of the current page
- ***window.location.hostname*** returns the domain name of the web host
- ***window.location.pathname*** returns the path and filename of the current page
- ***window.location.protocol*** returns the web protocol used (http: or https:)
- ***window.location.assign*** loads a new document

Eg.

```
console.log(window.location.href);
console.log(window.location.hostname);
console.log(window.location.pathname);
console.log(window.location.protocol);
console.log(window.location.port);
console.log(window.location.assign("https://www.w3schools.com"));
```

**Document Object Model:**

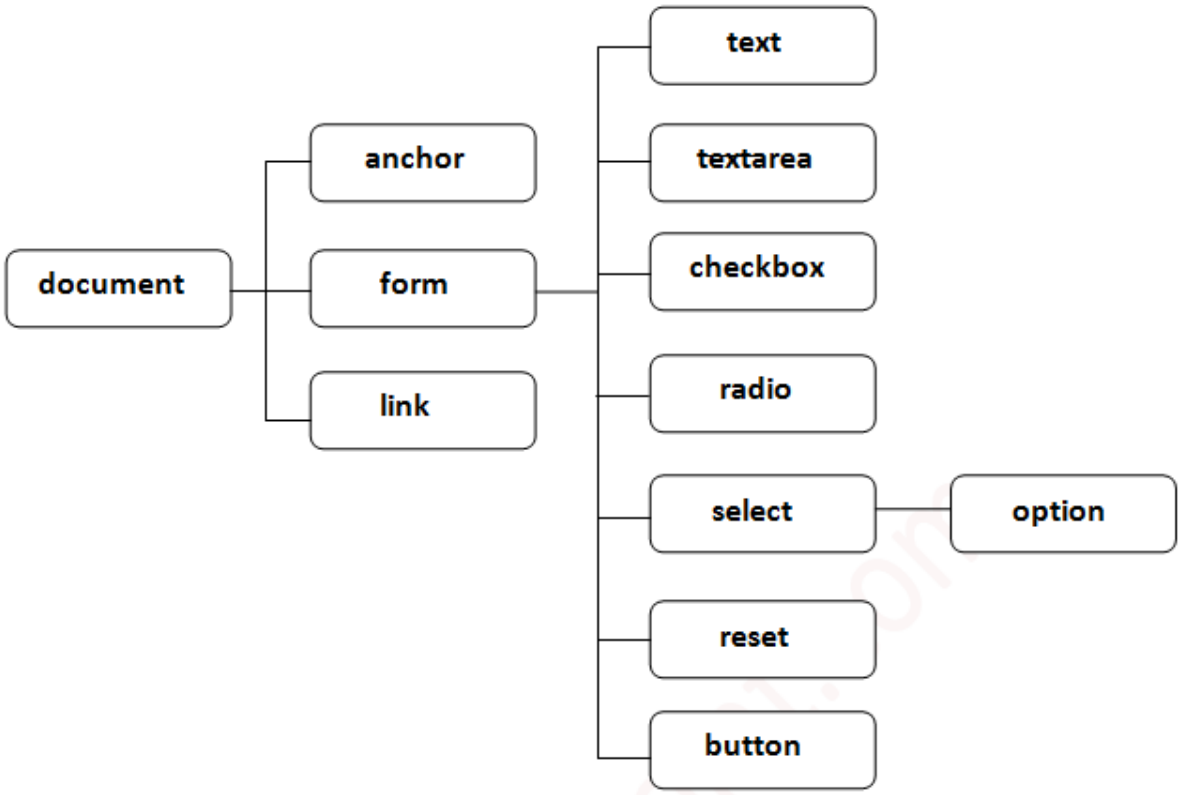
The ***document object*** represents the whole html document.

When a html document is loaded in the browser, it becomes a ***document object***. It is the root element that represents the ***html*** document. It has ***properties and methods***. By the help of document objects, we can add dynamic content to our web page.
The document object is the window property, so it can be accessed by:

***window.document***

OR

***document***




**Finding HTML Elements:**

***1. By ID***

*document.getElementById()* is a method for getting hold of an element by its ID.

Eg.

```
<script type="text/javascript">
   function getcube(){
      var number=document.getElementById("number").value;
      alert(number*number*number);
   }
</script>
<form>
    Enter No:<input type="text" id="number" name="number"/>
```

```
    <input type="button" value="cube" onclick="getcube()"/>
</form>
```

***2. By Name***

*document.getElementsByName()* returns all the elements of the specified name.

Eg.

```
var genders=document.getElementsByName("gender");
```

***3. By Tag Name***

*document.getElementsByTagName* works in much the same way as *getElementById*, except that it takes a tag name (a, ul, li, etc) instead of an ID and returns a *NodeList*

***3. By Class Name***

*document.getElementsByClassName* returns the same kind of *NodeList* as *getElementsByTagName*, except that you pass a class name to be matched, not a tag name.

***4. By CSS Selector***

*querySelector*, like *getElementById*, returns only one element whereas *querySelectorAll* returns a *NodeList*. If multiple elements match the selector you pass to *querySelector*, only the first will be returned.

Eg.

```
document.querySelector("body");  // get the body element
document.querySelector("#myDiv"); // get the element with the ID “myDiv”
document.querySelector(".selected"); // get first element with a class of “selected”
document.body.querySelector("img.button"); // get first image with class of “button”

// get all <em> elements in a <div> (similar to getElementsByTagName("em"))
document.getElementById("myDiv").querySelectorAll("em");
// get all elements with the class of "selected"
document.querySelectorAll(".selected");
// get all <strong> elements inside of <p> elements
document.querySelectorAll("p strong");
```

**Changing HTML Elements:**

**I. innerHTML**

The innerHTML property can be used to write the dynamic html on the html document.

Eg.

```
<script type="text/javascript" >
  function showCommentForm() {
      var data="Name:<input type='text' name='name'><br>Comment:<br>
      <textarea  rows='5' cols='80'></textarea><br>
      <input type='submit' value='Post Comment'>";
      document.getElementById('commentBox').innerHTML=data;
  }
```

```
</script>
<form name="comment">
   <input type="button" value="Send" onclick="showCommentForm()">
   <div id="commentBox"></div>
</form>
```

**II. innerText**

The innerText property can be used to write the dynamic text on the html document.

Eg.

```
<script type="text/javascript" >
function validate() {
     var msg;
    if(document.userForm.password.value.length > 5){
        msg="good";
    }  else{
       msg="poor";
   }
 document.getElementById(passwordBox).innerText = msg;
}

</script>
<form name="userForm">
   <input type="password" value="" name="password" onkeyup="validate()">
    Strength:<span id="passwordBox">no strength</span>
</form>
```

**III. document.write()**

In JavaScript, document.write() can be used to write directly to the HTML output stream.

Eg.

```
<script>
        document.write(Date());
</script>
```

**IV. Changing the Value of an Attribute**

Syntax:

***document.getElementById(id).attribute = new value***

Eg.

```
<!DOCTYPE html>
<html>
<body>
<img id="myImage" src="smiley.gif">
<script>
```

```
    document.getElementById("myImage").src = "landscape.jpg";
</script>
</body>
</html>
```

**V. Changing HTML Style**

Syntax:

***document.getElementById(id).style.property = new style***

Eg.

```
<html>
<body>
<p id="p2">Hello World!</p>
<script>
    document.getElementById("p2").style.color = "blue";
</script>
<p>The paragraph above was changed by a script.</p>
</body>
</html>
```

**HTML/DOM Events:**

| **Events** | **Description** |
|---|---|
| onclick | occurs when the element is clicked. |
| ondblclick | occurs when element is double-clicked. |
| onfocus | occurs when an element gets focus such as button, input, textarea etc. |
| onblur | occurs when form loses the focus from an element. |
| onsubmit | occurs when a form is submitted. |
| onmouseover | occurs when a mouse is moved over an element. |
| onmouseout | occurs when the mouse is moved out from an element (after moved over). |
| onmousedown | occurs when the mouse button is pressed over an element. |
| onmouseup | occurs when the mouse is released from an element (after the mouse is pressed). |

| | |
|---|---|
| onload | occurs when a document, object or frameset is loaded. |
| onunload | occurs when the body or frameset is unloaded. |
| onscroll | occurs when the document is scrolled. |
| onresized | occurs when a document is resized. |
| onreset | occurs when form is reset. |
| onkeydown | occurs when the key is being pressed. |
| onkeypress | occurs when the user presses the key. |
| onkeyup | occurs when the key is released. |

**Event Handler:**
An ***event handler*** is a predefined JavaScript property of an object (in most cases an element in the document) that is used to handle an event on a Web page.

An ***event*** is something that happens when the viewer of the page performs some sort of action, such as clicking a mouse button, clicking a button on the page, changing the contents of a form element, or moving the mouse over a link on the page. Events can also occur simply by the page loading or other similar actions.

***1. Using an Event Handler in an HTML Element***
Eg.

```
<body>
   <form>
        <input type="button" value="Click Me!" onclick="window.alert('Hi!');" />

         <!-- outputs "click" event -->
         <input type="button" value="Click Me" onclick="alert(event.type)"/>

         <!-- outputs "Click Me" value -->
         <input type="button" value="Click Me" onclick="alert(this.value)"/>
   </form>
</body>
```

***Note:*** A semicolon enables you to add additional JavaScript code after the alert.

***2. Using an Event Handler in the script code***
Eg.

```
<body>
```

```
    <form>
        <input type="button" value="Click Me!" id="btn_message" />
    </form>
<script type="text/javascript">
function message() {
        window.alert('Hi!');
        window.alert('Bye!');
}
document.getElementById("btn_message").onclick = message;
</script>
</body>
```

***3.The addEventListener() Method***

Syntax:

***element.addEventListener('eventType', functionName, true_or_false);***

There are two possible values for that optional last argument:

- ***false (default)*** - handler is set on the ***bubbling phase***.
- ***true*** - the handler is set on the ***capturing phase***.

Eg.

```
var b1 = document.getElementById("btn1");
b1.addEventListener('click', function() {
      alert("Hello");
}, false);
```

To remove an event, you would use the removeEventListener() method:

***element.removeEventListener('eventType', functionName, true_or_false);***

**Link:** https://codepen.io/studioadam/full/EKoxJP

**Forms and Form Fields:**

To access one of the forms using JavaScript, you can use any one of the following options:

- Use the forms array of the document object
- Name the form in the opening form tag and use that name to access the form
- Give the form an id in the opening form tag and access it using the document.getElementById() method

***1. Using the forms Array***

The forms array allows you to access a form using an index number in the array.

```
document.forms[0];   // access the first form in the document
document.forms[1];   // access the second form in the document
```

***2. Using Form Names***

To use a form name, you must add a name="formName" attribute to the opening form tag on the form you want to access.

***document.formName;***

***3. Using an ID***

***document.getElementById("formId");***

Eg.

```
<form action="#" name="loginForm" id="login">
  Name: <input type="text" name="username" id="username" value="ram"><br>
  Password: <input type="password" name="password" value="123"><br>
  <select id="gender">
      <option selected="selected" value="male">Male</option>
      <option value="female">Female</option>
  </select>
   <input type="checkbox" id="purple">
  <button type="submit">Login</button>
</form>

<script>
var form = document.querySelector("form");
console.log(form.elements[1].type);
console.log(form.elements.password.type);
console.log(form.name);

console.log(document.forms[0].elements[0]);
console.log(document.forms[0].elements[1].value);

console.log(document.loginForm.elements[0]);
console.log(document.loginForm.username);
console.log(document.loginForm.elements[0].value);
console.log(document.loginForm.username.value);

console.log(document.getElementById("login").name);
console.log(document.getElementById("username").value);
console.log(document.getElementById("gender").options);
console.log(document.getElementById("gender").options[1].value);
console.log(document.getElementById("gender").options[1].text);
var gender = document.getElementById("gender");
console.log(document.getElementById("gender").options[gender.selectedIndex].value);

console.log(document.body);

document.getElementById("purple").checked = true;
```

```
console.log(document.getElementById("purple").checked);

console.log(document.getElementById("gender").options[1].selected = true);

var form = document.getElementById("login");
var field = form.elements[0];
field.value = "Hari";  // change the value
field.focus();  // set focus to the field
field.disabled = true;  // disable the field

form.addEventListener("submit", function(event) {
        console.log("Saving value", form.elements.username.value);
        event.preventDefault();
});
</script>
```

**File fields:**

Eg.

```
<form action="/" method="post" onsubmit="return uploadImage();">
  <fieldset>
        <legend>Upload photo</legend>
        <input type="file" name="photo" id="photo">
        <input type="submit" value="Upload">
  </fieldset>
</form>

<script>
   function uploadImage() {
        var photo = document.getElementById("photo");
        var file = photo.files[0];

        console.log("File name: " + file.name);
        console.log("File size: " + file.size);
        console.log("File type: " + file.type);

        return false;
};
</script>
```

**Resource Link:**

https://javascript.info/
http://speakingjs.com/es5/index.html
https://frontendmasters.com/books/front-end-handbook/2018/what-is-a-FD.html