



# Javascript

# JavaScript

JavaScript, often abbreviated as JS, is a lightweight, high-level, **interpreted programming language** which enables dynamic interactivity.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web.

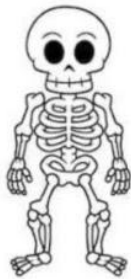
**JavaScript is mainly used for web-based applications and web browsers.** But JavaScript is also used beyond the Web in software, servers and embedded hardware controls.

JavaScript is not Java.

**HTML**



Skeleton



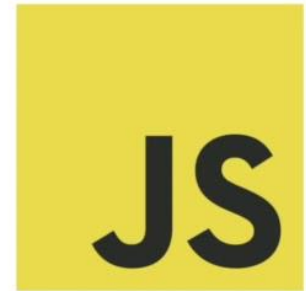
**CSS**



Style

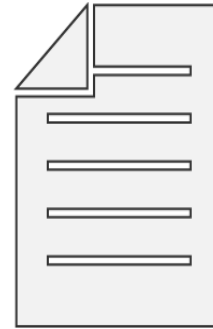


**JS**



Functionality





**Page**

**HTML**



Structure

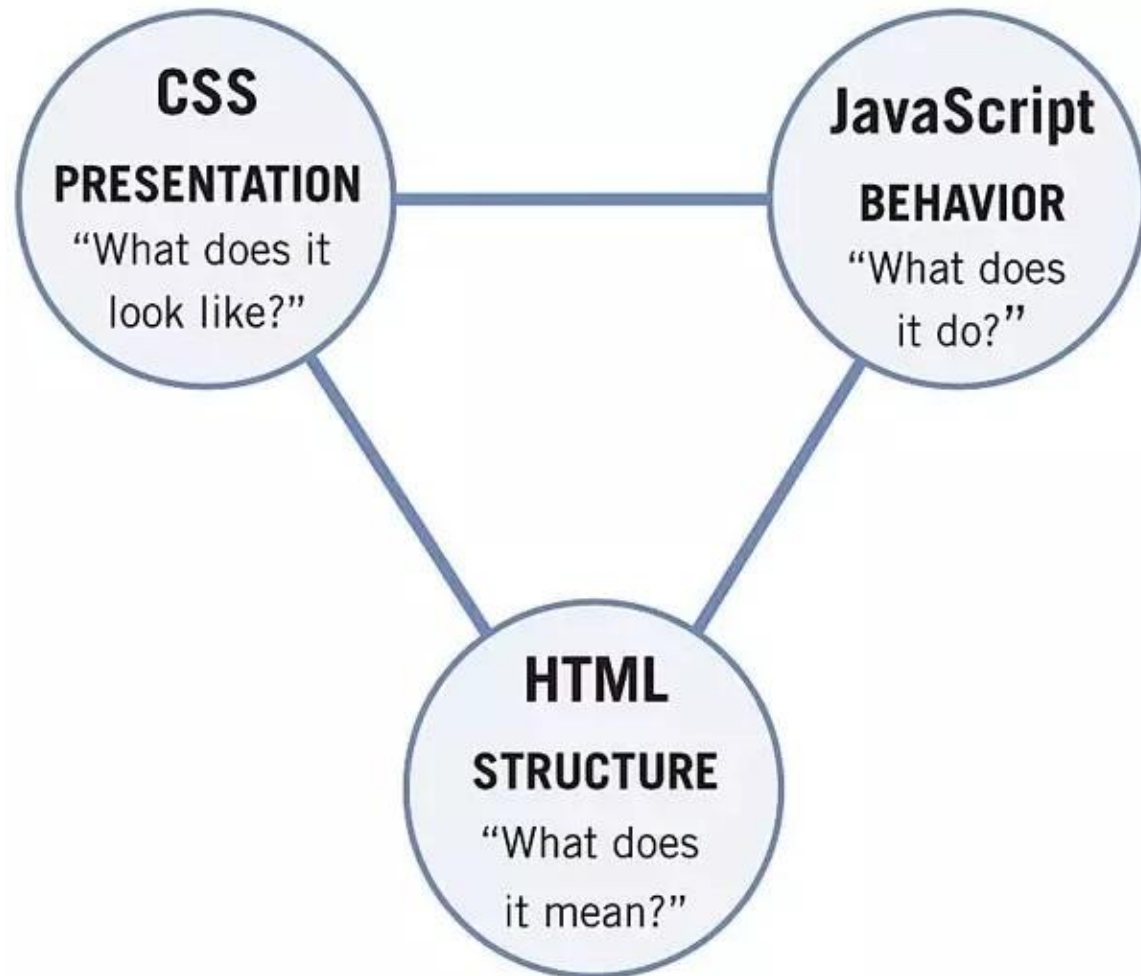
**CSS**



Styling

**JS**

Functionality



# Example

Oops!

We couldn't save your profile as entered. Please take a look at the following:

- Login has already been taken
- Email address doesn't match confirmation

Desired Username

Must be at least 4 characters

Email

Retype Email

Password

Retype Password

# DOM

The Document Object Model or DOM is created by the browser when a webpage is loaded. In graphical form, its like **a tree of elements also called nodes**, which are used to represent every single element on the page.

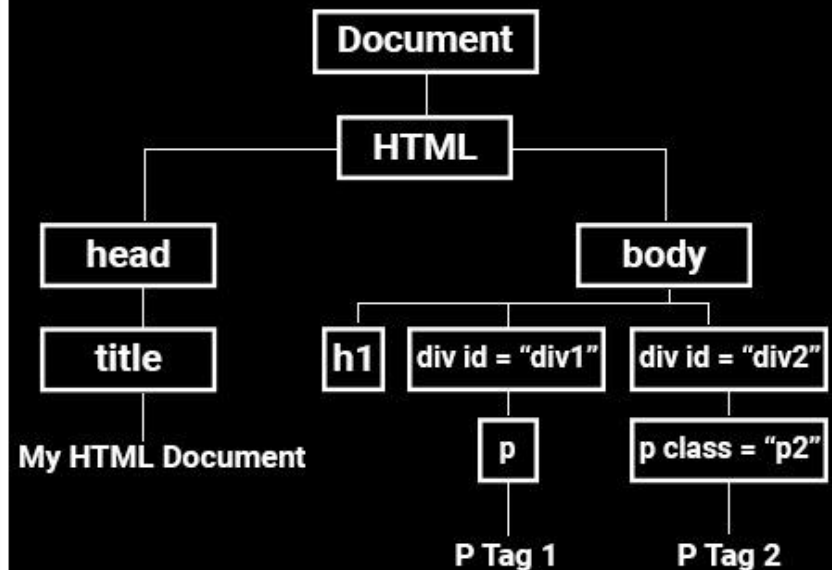
The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.

# How JS interacts with DOM


## HTML Document

```
index.html x
1 <html>
2   <head>
3     <title>My HTML Document</title>
4   </head>
5
6   <body>
7     <h1>Heading</h1>
8     <div id="div1">
9       <p>P Tag 1</p>
10    </div>
11    <div id="div2">
12      <p class="p2">P Tag 2</p>
13    </div>
14  </body>
15 </html>
```

## Document Object Model (DOM)







The Document Object Model (DOM) defines the logical structure of documents and the way a document is accessed and manipulated.

The name “Document Object Model” was chosen because it is an “object model” is used in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the above diagram do not represent a data structure, they represent objects, which have functions and identity.

JavaScript interacts with HTML document indirectly by interacting with the DOM. With the document object model, JavaScript gets all the power it needs to create dynamic HTML:

JavaScript can change all the HTML elements in the page

JavaScript can change all the HTML attributes in the page

JavaScript can change all the CSS styles in the page

JavaScript can remove existing HTML elements and attributes

JavaScript can add new HTML elements and attributes

JavaScript can react to all existing HTML events in the page

JavaScript can create new HTML events in the page

# JAVASCRIPT : Key Advantages

Javascript is the most popular programming language in the world and that makes it a programmer's great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS, Reactjs, Angular JS etc.

Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.

Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.

JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.

Due to high demand, there is tons of **job growth and high pay** for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.

Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

# Application of JavaScript

**Client side validation** - This is really important to verify any user input before submitting it to the server and Javascript plays an important role in validating those inputs at front-end itself.

**Manipulating HTML Pages** - Javascript helps in manipulating HTML page on the fly. This helps in adding and deleting any HTML tag very easily using javascript and modify your HTML to change its look and feel based on different devices and requirements.

**User Notifications** - You can use Javascript to raise dynamic pop-ups on the webpages to give different types of notifications to your website visitors.

**Back-end Data Loading** - Javascript provides Ajax library which helps in loading back-end data while you are doing some other processing. This really gives an amazing experience to your website visitors.

**Presentations** - JavaScript also provides the facility of creating presentations which gives website look and feel. JavaScript provides RevealJS and BespokeJS libraries to build a web-based slide presentations.

**Server Applications** - Node JS is built on Chrome's Javascript runtime for building fast and scalable network applications. This is an event based library which helps in developing very sophisticated server applications including Web Servers.

# Simple javascript program

```
<script language = "javascript" type = "text/javascript">  
    document.write("Hello JavaScript!");  
</script>
```

There are three ways of executing JavaScript on a web browser.

1. Inside <HEAD> tag
2. Within <BODY> tag
3. In an External File

## Inside <HEAD> tag

```
<html>
  <head>
    <script type = "text/javascript">
      alert("Hello JS");
    </script>
  </head>
<body>
</body>
</html>
```

## Within <BODY> tag

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello JS");
    </script>
  </body>
</html>
```

# External File

```
<html>  
  <body>  
    <script type="text/javascript" src="js/script.js">  
  </script>  
  </body>  
</html>
```

```
Js/script.js      //External File Name  
document.write("Hello JS");
```

# Javascript programming

JavaScript programs consist of a series of instructions known as statements, just as written paragraphs consist of a series of sentences. a JavaScript statement often ends in a semicolon (;).

```
// A single JavaScript statement  
const now = new Date();
```

```
// Get the current timestamp and print it to the console  
const now = new Date(); console.log(now);
```

```
// Two statements separated by newlines  
const now = new Date();  
console.log(now);
```

# Comments in JavaScript

## Single line

Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.

Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.

JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.

The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

```
<script language = "javascript" type = "text/javascript">  
  <!--  
    // This is a comment. It is similar to comments in C++  
  
    /*  
     * This is a multi-line comment in JavaScript  
     * It is very similar to comments in C Programming  
     */  
    //-->  
</script>
```



# Variables and Data Types

A variable is a "named storage" for data. We can use variables to store amount, visitors, and other data.

To create a variable in JavaScript, use the let keyword.

The statement below declares a variable with the name "message":

```
let message;
```

Storing data to the variable

```
let message;  
message = 'Welcome to the web technology';
```

We can also declare multiple variables in one line:

```
let user = 'John', age = 24, message = 'Hello';
```

# Variables and DataTypes

The var keyword is almost the same as let. It also declares a variable, but in a slightly different

```
var message = 'Hello';
```

let allows us to declare variables that are limited to the scope of a block statement, or expression on which it is used, unlike the var keyword, which declares a variable globally, or locally to an entire function regardless of block scope.

# Variables and DataTypes

JavaScript has three primitive data types –

Numbers, eg. 123, 123.45 etc.

Strings of text e.g. "This text is string" etc.

Boolean e.g. true or false.

JavaScript also defines two trivial data types, null and undefined, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as object.

# JavaScript Keywords

Keywords are reserved words that are part of the syntax in the programming language. For example,

```
const a = 'Hello';
```

Here, `const` is a keyword that denotes that `a` is a constant.

Keywords cannot be used to name identifiers.

In other words Keywords are reserved words in JavaScript that you cannot use to indicate variable labels or function names.

Here is the list of keywords available in JavaScript.

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

## Flow controls


The if statement is probably one of the most frequently used statements in JavaScript. The if statement executes a statement or block of code if a condition is satisfied. The following is the simple form of the if statement:

```
if( condition )  
    statement;
```

The condition can be any valid expression. In general, the condition evaluates to a Boolean value, either true or false.

In case the condition evaluates to a non-Boolean value, JavaScript implicitly converts its result into a Boolean value by calling the Boolean() function.

If the condition evaluates to true, the statement is executed. Otherwise, the control is passed to the next statement that follows the if statement.



```
let x = 25;  
if( x > 10 )  
    console.log('x is greater than 10');
```

This example first initializes the variable x to 25. The `x > 10` expression evaluates to true, therefore the script shows a message in the console window

In case we have more than one statement to execute, we need to use curly braces as follows:

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

In case we want to execute another statement when the condition evaluates to false, we use the else as follows:

```
if( condition ) {  
    // statement  
} else {  
    // statement (when condition evaluates to false)  
}
```

## The switch statement

The switch statement is a flow-control statement that is similar to the if else statement. We use the switch statement to control the complex conditional operations.

The syntax of the switch statement:

```
switch (expression) {  
    case value_1:  
        statement_1;  
        break;  
    case value_2:  
        statement_2;  
        break;  
    case value_3:  
        statement_3;  
        break;  
    default:  
        default_statement;  
}
```



```
var day = 3; var dayName;
switch (day) {
  case 1:
    dayName = 'Sunday';
    break;
  case 2:
    dayName = 'Monday';
    break;
  case 3:
    dayName = 'Tuesday';
    break;
  case 4:
    dayName = 'Wednesday';
    break;
  case 5:
    dayName = 'Thursday';
    break;
  case 6:
    dayName = 'Friday';
    break;
  case 7:
    dayName = 'Saturday';
    break;
  default:
    dayName = 'Invalid day';
}
console.log(dayName);
```

# Types of Loops in JavaScript

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. JavaScript supports five different types of loops:

## 1) While Loop

The while loop loops through a block of code as long as the specified condition evaluates to true. As soon as the condition fails, the loop is stopped.

```
while(condition) {  
    // Code to be executed  
}  
  
var i = 1;  
while(i <= 5) {  
    document.write("<p>The number is " + i + "</p>");  
    i++;  
}
```

## 2) do while Loop

The do-while loop is a variant of the while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to true.

```
do {  
    // Code to be executed  
} while(condition);
```

```
var i = 1;  
do {  
    document.write("<p>The number is " + i + "</p>");  
    i++;  
}  
while(i <= 5);
```

### 3) For Loop:

The for loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times. Its syntax is:

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

```
for(var i=1; i<=5; i++) {  
    document.write("<p>The number is " + i + "</p>");  
}
```

### 4) For in Loop

The for-in loop is a special type of a loop that iterates over the properties of an object, or the elements of an array.

```
for (variable in object) {  
    // Code to be executed  
}
```

```
var person = {"firstname": "John", "lastname": "nash", "age": "64"};  
for(var prop in person) {  
    document.write("<p>" + prop + " = " + person[prop] + "</p>");  
}
```

## 5) for of loop

A newly introduced for-of loop which allows us to iterate over arrays or other iterable objects (e.g. strings) very easily. Also, the code inside the loop is executed for each element of the iterable object


```
let greet = "Hello World!";
```

```
for(let character of greet) {  
    document.write(character); // H,e,l,l,o, ,W,o,r,l,d,  
}
```

## Javascript Function

A function is a group of statements that perform specific tasks and can be kept and maintained separately from main program. Functions provide a way to create reusable code packages which are more portable and easier to debug.

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now you can perform the same task by calling this function wherever you want within your script without having to copy and paste the same block of code again and again.
- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.



The declaration of a function start with the function keyword, followed by the name of the function you want to create, followed by parentheses i.e. () and finally place your function's code between curly brackets {}. Here's the basic syntax for declaring a function:

```
function functionName() {  
    // Code to be executed  
}
```

// Defining function

```
function welcome() {  
    document.write("Hello, welcome to the webtechnology!");  
}
```

// Calling function

```
welcome(); // Outputs: Hello, welcome to the webtechnology!
```

## Adding Parameters to Functions

You can specify parameters when you define your function to accept input values at run time. The parameters work like placeholder variables within a function; they're replaced at run time by the values (known as argument) provided to the function at the time of invocation.

Parameters are set on the first line of the function inside the set of parentheses, like this:

```
function functionName(parameter1, parameter2, parameter3) {  
    // Code to be executed  
}
```

// Defining function

```
function showFullname(firstName, lastName) {  
    document.write(firstName + " " + lastName);  
}
```

// Calling function

```
showFullname("John", "Nash"); // outputs: John Nash
```



## Default Values for Function Parameters

```
function welcome(name = 'Guest') {  
    document.write('Hello, ' + name);  
}
```

```
welcome(); // Outputs: Hello, Guest  
welcome('John'); // Outputs: Hello, John
```

## Returning Values from a Function

A function can return a value back to the script that called the function as a result using the return statement. The value may be of any type, including arrays and objects.

```
// Defining function  
function getSum(num1, num2) {  
    var total = num1 + num2;  
    return total;  
}
```

```
// Displaying returned value  
document.write(getSum(5, 20)); // Outputs: 25
```

# Popup Boxes

## **Alert Box**

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

### Syntax

```
window.alert("sometext");
```

The window.alert() method can be written without the window prefix.

```
alert("I am an alert box!");
```

# Confirm Box

A confirm box is often used if we want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
window.confirm("sometext");
```

The window.confirm() method can be written without the window prefix.

Example

```
if (confirm("Press a button!")) {  
    txt = "You pressed OK!";  
} else {  
    txt = "You pressed Cancel!";  
}
```

# Prompt Dialog Box

The prompt dialog box is used to prompt the user to enter information. A prompt dialog box includes a text input field, an OK and a Cancel button.

We can create prompt dialog boxes with the `prompt()` method. This method returns the text entered in the input field when the user clicks the OK button, and null if user clicks the Cancel button. If the user clicks OK button without entering any text, an empty string is returned. For this reason, its result is usually assigned to a variable when it is used.

```
var name = prompt("Enter your name ?");  
if(name.length > 0 && name != "null") {  
    document.write("Hi, " + name);  
} else {  
    document.write("Hi, Guest!");  
}
```

# Arrays in JavaScript

Arrays are complex variables that allow us to **store more than one value** or a group of values under **a single variable name**. JavaScript arrays can store any valid value, including strings, numbers, objects, functions, and even other arrays, thus making it possible to create more complex data structures such as an array of objects or an array of arrays.

## Syntax:

```
var myArray = [element0, element1, ..., elementN];
```

```
var colorArray = ["Red", "Green", "Blue"];
```

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
```

```
var cities = ["London", "Paris", "Kathmandu"];
```

```
var person = ["John", "Nash", 54];
```

# Accessing Array Elements

Array elements can be accessed by their index using the square bracket notation. An index is a number that represents an element's position in an array.

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
```

```
document.write(fruits[0]);
```

```
document.write(fruits[1]);
```

```
document.write(fruits[2]);
```

```
document.write(fruits[3]);
```

```
document.write(fruits[4]);
```

```
document.write(fruits[fruits.length - 2]);
```

```
document.write(fruits[fruits.length - 1]);
```

## Getting the Length of an Array

```
document.write(fruits.length);
```

# Accessing Array Elements using Loops

We can use for loop to access each element of an array in sequential order:

```
var fruits = ["Apple", "Banana", "Mango", "Orange", "Papaya"];
```

```
for (var i = 0; i < fruits.length; i++) {  
    document.write(fruits[i] + "<br>");  
}
```

```
for (var fruit of fruits) {  
    document.write(fruit + "<br>");  
}
```

```
for (var i in fruits) {  
    document.write(fruits[i] + "<br>");  
}
```

# Adding New Element to an Array

We can use the `push()` method to add a new element at the end of an array.

```
var colors = ["Red", "Green", "Blue"];  
colors.push("Yellow");
```

Similarly, to add a new element at the beginning of an array use the `unshift()` method.

```
var colors = ["Red", "Green", "Blue"];  
colors.unshift("Yellow");
```

Adding multiple elements at once using the `push()` and `unshift()`

```
var colors = ["Red", "Green", "Blue"];  
colors.push("Pink", "Voilet");  
colors.unshift("Yellow", "Grey");
```



# Removing Elements from an Array

To remove the last element from an array you can use the pop() method. This method returns the value that was popped out.

```
var colors = ["Red", "Green", "Blue"];  
var last = colors.pop();
```

```
document.write(last);
```

Removing the first element from an array using the shift() method

```
var colors = ["Red", "Green", "Blue"];  
var first = colors.shift();
```

# JavaScript Objects

JavaScript is an object-based language and in JavaScript almost everything is an object or acts like an object. A JavaScript object is just a collection of named values. These named values are usually referred to as properties of the object. An array is a collection of values, where each value has an index (a numeric key) that starts from zero and increments by one for each value. An object is similar to an array, but the difference is that you define the keys yourself, such as name, age, gender, and so on.

Object-Oriented Languages (OOP) follow all the concepts of OOPs whereas, Object-based languages don't follow all the concepts of OOPs like inheritance and polymorphism.

Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.

Examples for Object Oriented Languages include Java, C#  
whereas Object-based languages include Javascript, VB etc.

# Creating Objects

An object can be created with curly brackets {} with an optional list of properties. A property is a "key: value" pair, where the key (or property name) is always a string, and value (or property value) can be any data type, like strings, numbers, Booleans or complex data type like arrays, functions, and other objects. Additionally, properties with functions as their values are often called methods to distinguish them from other properties.

```
var person = {  
  name: "John Nash",  
  age: 54,  
  gender: "Male",  
  displayName: function() { alert(this.name)}  
};
```

# Accessing Object's Properties

To access or get the value of a property, you can use the dot (.), or square bracket ([]) notation,

```
var person = {  
  name: "John Nash",  
  age: 54,  
  gender: "Male",  
  displayName: function() {  
    alert(this.name);  
  }  
};
```

// Dot notation

```
document.write(person.name);
```

// Bracket notation

```
document.write(person["gender"]);
```

# Looping Through Object's Properties

We can iterate through the key-value pairs of an object using the for...in loop. This loop is specially optimized for iterating over object's properties.

```
for(var i in person) {  
    document.write(person[i] + "<br>")  
}
```

# Deleting Object's Properties

The delete operator can be used to completely remove properties from an object. Deleting is the only way to actually remove a property from an object.

Setting the property to undefined or null only changes the value of the property. It does not remove property from the object.

```
delete person.age;  
alert(person.age);
```

# Calling Object's Methods

You can access an object's method the same way as you would access properties—using the dot notation or using the square bracket notation.

```
person.displayName();
```

```
person["displayName"]();
```



# Javascript Class

A JavaScript class is a blueprint for creating objects. A class encapsulates data and functions that manipulate data.

```
class Person {  
  constructor(name) {  
    this.name = name;  
  }  
  getName() {  
    return this.name;  
  }  
}  
  
let person = new Person("John");  
person.getName();
```

A JavaScript class is not an object. It is a template for JavaScript objects.



# Constructor in JavaScript

The constructor in JavaScript **is a special method used to create and initialize objects** within a class.

There are two types of constructors in JavaScript

## Built-in Constructors

These are the readily available constructors that come bundled with the execution environment. The user simply needs to invoke them, Examples of built-in constructors **are Array, Date, and Object**.

## User-defined Constructors

These are the constructors declared and defined by the programmer to be used throughout the application. we can also define properties and methods of our **own custom types**. They are also known as custom constructors. By convention, all JavaScript constructors are sentence cased. To use them that this function must be invoked using the new keyword.

Whenever a new object of the class type is declared, the new keyword returns a reference to the newly created object. That object is accessed using this keyword inside the constructor to initialize the properties of the object.

```
class Employee{  
    constructor(id, name){  
        this.id = id;  
        this.name = name;  
    }  
}  
var emp1 = new Employee(123, "John");  
console.log(emp1.name);
```

The Object constructor is called directly when an object of the class Object is created. This creates an object of class Object if null or undefined parameters are passed as arguments. Otherwise, an object of the type of given parameters is created.

```
var now = new Date();  
console.log(now);
```

# Built-in Objects

## **String:**

String object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

```
var str = new String("string");
```

## **charAt()**

Returns the character at the specified index.

```
var str = new String("This is a sample text");  
str.charAt(0);  
str.charAt(1);
```

## **concat()**

This method adds two or more strings and returns a new single string.

```
var str1 = new String("This is string one");  
    var str2 = new String("This is string two");  
    var str3 = str1.concat(str2);
```

## **indexOf()**

This method returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex or -1 if the value is not found.

```
var str1 = new String("This is line one");  
    var index = str1.indexOf("line");
```

## **slice()**

```
string.slice( beginslice [, endSlice] );
```

beginSlice – The zero-based index at which to begin extraction.

endSlice – The zero-based index at which to end extraction. If omitted, slice extracts to the end of the string.

```
var str = "This is a text in line one";  
    var sliced = str.slice(3, 7);  
    var sliced = str.slice(3, -5);
```

## **split()**

This method splits a String object into an array of strings by separating the string into substrings.

```
var str = "This is a sample text in a line one";  
    var splitted = str.split(" ", 3);
```

## **substr()**

This method returns the characters in a string beginning at the specified location through the specified number of characters.

```
string.substr(start, length);
```

start – Location at which to start extracting characters (an integer between 0 and one less than the length of the string).

length – The number of characters to extract.

```
var str = "This is a sample text from paragraph one";  
    str.substr(1,2);
```

# Math Object

The math object provides properties and methods for mathematical constants and functions. Unlike other global objects, Math doesn't have constructors. All the properties and methods of Math are static and can be called by using Math as an object without creating it.

Thus, you refer to the constant pi as Math.PI and you call the sine function as Math.sin(x), where x is the method's argument.

The syntax to call the properties and methods of Math are as follows

```
var piVal = Math.PI;  
var sineVal = Math.sin(45);
```

### **Math.sqrt(n)**

The JavaScript Math.sqrt() method returns the square root of the given number.

syntax: Math.sqrt(16);

### **Math.random()**

The JavaScript Math.random() method returns the random number between 0 and 1.

syntax Math.random();

### **Math.floor(n)**

The JavaScript math.floor(n) method returns the lowest integer for the given number. For example 3 for 3.7, 4 for 4.9 etc.

syntax: Math.floor(4.6);

### **Math.ceil(n)**

The JavaScript math.ceil(n) method returns the largest integer for the given number. For example 4 for 3.7, 5 for 4.9 etc.

syntax: Math.ceil(4.6);

### **Math.round(n)**

The JavaScript math.round(n) method returns the rounded integer nearest for the given number. If fractional part is equal or greater than 0.5, it goes to upper value 1 otherwise lower value 0. For example 4 for 3.7, 3 for 3.3, 6 for 5.9 etc.

syntax: Math.round(4.3);

## Regular Expression in JavaScript

- Regular expressions are patterns used to match character combinations in strings.
- In JavaScript, regular expressions are also objects.
- Regular expressions can be used to perform all types of text search and text replace.
- There are two main string methods: `search()` and `replace()`.
- `search()` method uses an expression to search for a match, and returns the position of the match.
- `replace()` method uses an expression to return a modified string where the pattern is replaced.



## Regular Expression in JavaScript

```
<body>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var str = "The quick brown fox jumps over the lazy dog";
```

```
var n = str.search(/brown/i);
```

```
document.getElementById("demo").innerHTML = n;
```

```
</script>
```

```
</body>
```

# Regular Expression in JavaScript

```
<body>
<button onclick="myFunction()">Click to replace</button>
<p id="demo">The quick brown fox jumps over the lazy dog</p>
<script>
function myFunction()
{
var str = document.getElementById("demo").innerHTML;
var txt = str.replace(/quick/i,"slow");
document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
```

## Form

To create a form in HTML, you use the <form> element:

```
<form action="/process.php" method="post" id="registerform">  
</form>
```

The <form> element has two important attributes: action and method.

action specifies a URL that processes the form submission. In this example, the action is the /process.php URL.

method specifies the HTTP method to submit the form with. The method is either post or get.

Typically, you use the get method when you want to retrieve data from the server and the post method when you want to change something on the server.

# Form Method Property

- It is used to change the method for sending form data.
- The method property sets or returns the value of method attribute in a form.
- The syntax for setting the method property is:  
`formObject.method = get | post`
- `get` – appends the form-data to the URL:  
`URL?name=value&name=value` (this is default).
- `post` – sends the form-data as an HTTP post transaction.

JavaScript uses the `HTMLFormElement` object to represent a form. The `HTMLFormElement` has the following properties that correspond to the HTML attributes:

`action` – is the URL that processes the form data. It is equivalent to the `action` attribute of the `<form>` element.

`method` – is the HTTP method which is equivalent to the `method` attribute of the `<form>` element.

The `HTMLFormElement` element also provides the following useful methods:


`submit()` – submit the form.

`reset()` – reset the form.

### **Referencing forms**

To reference the `<form>` element, you can use DOM selecting methods such as `getElementById()`:

```
const form = document.getElementById('registerform');
```



An HTML document can have multiple forms. The `document.forms` property returns a collection of forms (`HTMLFormControlsCollection`) on the document:

`document.forms`

Code language: JavaScript (javascript)

To reference a form, you use an index. For example, the following statement returns the first form of the HTML document:

`document.forms[0]`

# JavaScript Window


JavaScript Window is a global Interface (object type) which is used to control the browser window lifecycle and perform various operations on it.

A global variable window, which represents the current browser window in which the code is running, is available in our JavaScript code and can be directly accessed by using window literal.

It represents a window containing a webpage which in turn is represented by the document object. A window can be a new window, a new tab, a frame set or individual frame created with JavaScript.

The properties of a window object are used to retrieve the information about the window that is currently open, whereas its methods are used to perform specific tasks like opening, maximizing, minimizing the window, etc.

```
let h = window.outerHeight
let w = window.outerWidth
window.document.write("Height and width of the window are: "+h+" and "+w)
```



```
<button onclick="createWindow()">Open a Window</button>  
<button onclick="closewin()">Close the Window</button>  
<button onclick="showAlert()">Show Alert in Window</button>
```

```
<script>  
var win;  
// Open window  
function createWindow() {  
    win = window.open("", "My Window", "width=500, height=200");  
}  
  
function showAlert() {  
    if(win == undefined) {  
        // show alert in main window  
        window.alert("First create the new window, then show alert in it");  
    }  
    else {  
        win.alert("This is an alert");  
    }  
}  
  
// Close window  
function closewin(){  
    win.close();  
}
```



## Window Object in JavaScript

- The window object represents the browser's window.
- Two properties window.innerHeight and window.innerWidth can be used to determine the size (in pixels) of browser window.

```
<body>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var w = window.innerWidth;
```

```
var h = window.innerHeight;
```

```
var x = document.getElementById("demo");
```

```
x.innerHTML = "Browser inner window width: " + w + ", height: " +  
    h + ".";
```

```
</script>
```

```
</body>
```

# JavaScript Document Object

JavaScript Document object is an object that provides access to all HTML elements of a document. When an HTML document is loaded into a browser window, then it becomes a document object.

The document object stores the elements of an HTML document, such as HTML, HEAD, BODY, and other HTML tags as objects.

A document object is a child object of the Window object, which refers to the browser.

`document.getElementById(id)`

`document.getElementsByName(name)`

`document.getElementsByTagName(tagname)`

## Date Object in JavaScript

- The Date object is used to work with dates and times.
- Date objects are created with the Date() constructor.
- Date object methods in JavaScript:
  - getDate() – returns the day of the month (from 1 – 31)
  - getDay() – returns the day of the week (from 0 – 6)
  - getFullYear() – returns the year (four digits)
  - getHours() – returns the hour (from 0 – 23)
  - getMilliseconds() – returns the milliseconds (from 0 – 999)
  - getMinutes() – returns the minutes (from 0 – 59)
  - getMonth() – returns the month (from 0 – 11)
  - getSeconds() – returns the seconds (from 0 – 59)

# Date Object in JavaScript

```
<html>
<body>
<h2>JavaScript new Date()</h2>
<p id="demo"></p>
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
//uncomment following lines to check the output
//document.getElementById("demo").innerHTML = d.getDate();
//document.getElementById("demo").innerHTML = d.getDay();
//document.getElementById("demo").innerHTML = d.getFullYear();
//document.getElementById("demo").innerHTML = d.getHours();
//document.getElementById("demo").innerHTML = d.getMilliseconds();
//document.getElementById("demo").innerHTML = d.getMinutes();
//document.getElementById("demo").innerHTML = d.getMonths();
//document.getElementById("demo").innerHTML = d.getSeconds();
</script>
</body>
</html>
```

# JavaScript Events

An event is something that happens when user interact with the web page, such as when he clicked a link or button, entered text into an input box or textarea, made selection in a select box, pressed key on the keyboard, moved the mouse pointer, submits a form, etc. In some cases, the Browser itself can trigger the events, such as the page load and unload events.. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

There are several ways to assign an event handler. The simplest way is to add them directly to the start tag of the HTML elements using the special event-handler attributes. For example, to assign a click handler for a button element, we can use onclick attribute.

```
<button type="button" onclick="alert('Hello World!')">Click Me</button>
```

Some of the HTML events and their event handlers are:

**onclick** > When mouse click on an element

```
function clickevent()  
{  
    document.write("You clicked a button");  
}
```

```
<input type="button" onclick="clickevent()" value="click here"/>
```

**mouseover** > When the cursor of the mouse comes over the element

```
function mouseoverevent()  
{  
    alert("This is an example of mouse over");  
}
```

```
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
```

**load** > When the browser finishes the loading of the page

```
<body onload="window.alert('Page is loaded');">
```

## The Change Event (onchange)

The change event occurs when a user changes the value of a form element.

You can handle the change event with the onchange event handler. The following example will show you an alert message when you change the option in the select box.

```
<select onchange="alert('You have changed the selection!');">  
  <option>Select</option>  
  <option>Male</option>  
  <option>Female</option>  
</select>
```

## Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;

if (name==null || name==""){
    alert("Name can't be blank");
    return false;
}else if(password.length<6){
    alert("Password must be at least 6 characters long.");
    return false;
}
}
</script>
```





```
<body>
```

```
<form name="myform" method="post" action="login.php" onsubmit="return  
    validateform()" >
```

```
Name: <input type="text" name="name"><br/>
```

```
Password: <input type="password" name="password"><br/>
```

```
<input type="submit" value="login">
```

# Error Handling

JavaScript is a loosely-typed language. It does not give compile-time errors. So some times you will get a runtime error for accessing an undefined variable or calling undefined function etc.

JavaScript provides error-handling mechanism to catch runtime errors using try-catch-finally block, similar to other languages like Java or C#.

Any code that might possibly throw an error should be placed in the try block of the statement, and the code to handle the error is placed in the catch block, as shown here:

```
try {  
    // Code that may cause an error  
} catch(error) {  
    // Action to be performed when an error occurs  
} finally {  
    // code to be executed regardless of an error occurs or not  
}
```

try: wrap suspicious code that may throw an error in try block.

catch: write code to do something in catch block when an error occurs. The catch block can have parameters that will give you error information. Generally catch block is used to log an error or display specific messages to the user.

finally: code in the finally block will always be executed regardless of the occurrence of an error. The finally block can be used to complete the remaining task or reset variables that might have changed before error occurred in try block.

```
try
{
    var result = Add(10, 20);
}
catch(ex)
{
    alert(ex);
}
```



Throw : Use throw keyword to raise a custom error.

```
try
{
    throw "Error occurred";
}
catch(ex)
{
    alert(ex);
}
```

# JavaScript Error Handling – try, catch, & throw (Example)

```
<body>
<p>Please input a number between 5 and 10:</p>
<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>
<script>
function myFunction() {
var message, x;
message = document.getElementById("p01");
message.innerHTML = "";
x = document.getElementById("demo").value;
try {
if(x == "") throw "is empty";
if(isNaN(x)) throw "is not a number";
x = Number(x);
if(x > 10) throw "is too high";
if(x < 5) throw "is too low";
}
catch(err) { message.innerHTML = "Input " + err; }
finally { document.getElementById("demo").value = ""; }
}
</script>
</body>
```



# Cookies

Web Browsers and Servers use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website, it is required to maintain session information among different pages. For example, one user registration ends after completing many pages. But how to maintain users' session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

## Handling Cookies in JavaScript

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem “how to remember information about the user?”
  - o When a user visits a web page, his/her name can be stored in a cookie.
  - o Next time the user visits the page, the cookie remembers his/her name.
- Cookies are saved in name-value pairs like: username = John Nash
- When a browser requests a web page from a server, cookies belonging to the page are added to the request; this way the server gets the necessary data to remember information about users.
- JavaScript can create, read, and delete cookies with the `document.cookie` property.

## Handling Cookies in JavaScript

Examples of cookies are:

- o Name Cookie – the first time a visitor arrives to your web page, he/she must fill in his/her name. The name is then stored in a cookie. Next time the visitor arrives at your page, he/she could get a welcome message like “Welcome John Doe!”. The name is retrieved from the stored cookie.
- o Password Cookie – the first time a visitor arrives to your web page, he/she must fill in a password. The password is then stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie.
- o Date Cookie – the first time a visitor arrives to your web page, the current date is stored in a cookie. Next time the visitor arrives at your page, he/she could get a message like “Your last visit was on Tuesday July 30, 2019!”. The date is retrieved from the stored cookie.



## Create a Cookie with JavaScript

- With JavaScript, a cookie can be created like this:

```
document.cookie = "username=John Nash";
```

- You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

```
document.cookie = "username=John Nash; expires=Thu, 18 Dec 2021 12:00:00 UTC";
```

- With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie = "username=John Nash; expires=Thu, 18 Dec 2021 12:00:00 UTC; path=/";
```

## Read a Cookie with JavaScript

- With JavaScript, a cookie can be read like this:

```
var x = document.cookie;
```

## Change a Cookie with JavaScript

- With JavaScript, you can change a cookie the same way as you create it:

```
document.cookie = "username=John Nash; expires=Thu, 18  
Dec 2021
```

```
12:00:00 UTC; path="/";
```

- The old cookie is overwritten

## Delete a Cookie with JavaScript

- Deleting a cookie is very simple as you don't have to specify a cookie value when you delete it.

- Just set the expires parameter to a passed date:

```
document.cookie = "username; expires=Thu, 01 Jan 1970  
00:00:00 UTC; path="/";
```