

UNIT-3 JAVASCRIPT

Introduction:

JavaScript is a scripting language developed by Developed by Brendan Eich at Netscape (European Computer Manufacturer's association (ECMA))

Scripting Language:

Language used to write programs that compute inputs to another language processor

JavaScript (JS) is a lightweight, interpreted, compiled programming language with first class functions.

JavaScript (JS) enables dynamic interactivity on websites when applied to an HTML document. JavaScript that runs at the client side (i.e. at the client's browser) is client side java script (CCJS) and JavaScript that runs at the server is server side java script (SSJS).

JavaScript is a case-sensitive language. This means the language considers capital letters to be different from their lowercase counterparts

Common Uses of JavaScript

- client-side validation

- Form validation

- Basic math calculations

- Dynamic content manipulation

- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box).

- Displaying clocks etc.

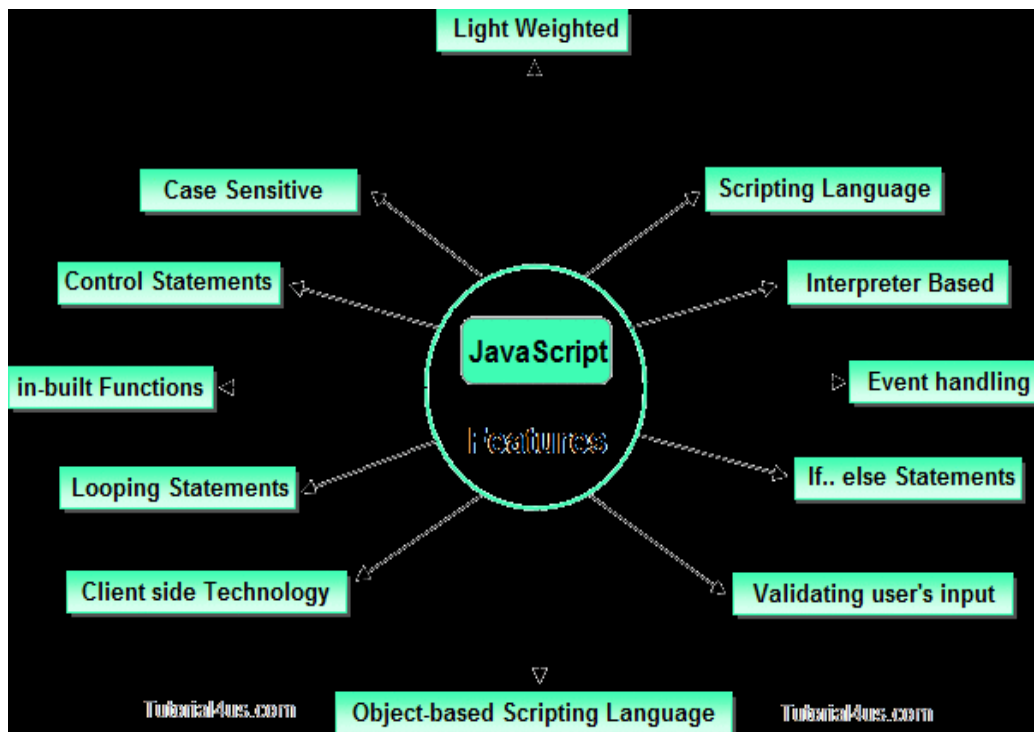
Why we Use JavaScript?

HTML can be used to design only a web page ,but cannot run any logic on web browser like addition of two numbers, check any condition, looping statements (for, while), decision making statement (if-else) at client side.

All these are not possible using HTML So to perform all these task at client side JavaScript is used.

Features of JavaScript

JavaScript is a client side technology, it is mainly used for gives client side validation, but it has lot of features which are given below:



Creating a java script: - html script tag is used to script code inside the html page. <script> </script> The script is containing 2 attributes. They are

1) Language attribute:-

It represents name of scripting language such as JavaScript, VbScript.

```
<script language="JavaScript"||>
```

2) Type attribute: - It indicates MIME (multi purpose internet mail extension) type of scripting code. It sets to an alpha-numeric MIME type of code.

```
<script type="text / JavaScript"||>
```

Location of script or placing the script: -

Script code can be placed in both head & body section of html page.

Ex: - <html>
<head>
<script type="text / JavaScript">
Script code here
</script>

</head>

```
<body>

<script type="text / JavaScript">

    Script code here

</script>

</body>

</html>
```

JavaScript Variables

Variables are used to store data values.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Examples:

1. `var x = 10;`
2. `var _name="cse";`

There are three keywords in JavaScript that can be used to declare variables: `let`, `var`, and `const`. Each keyword has different rules and implications for how the variables they create can be used.

1. **let:** The `let` keyword declares a block-scoped local variable, optionally initializing it to a value.

Block-scoped means that the variable is only available within the block it was declared in, which is usually denoted by curly braces {}.

2. **var:** The `var` keyword declares a function-scoped or global variable, optionally initializing it to a value.

Function-scoped means that the variable is only available within the function it was declared in. Global variables are available throughout your entire code.

3. **const:** The const keyword declares a block-scoped, immutable constant variable, i.e. a variable that can't be reassigned.

Constants are also called “immutable variables”, but that's a bit of a misnomer since they are actually variables – just ones that can't be reassigned.

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

JavaScript local variable

A **JavaScript local variable** is declared inside block or function. It is accessible within the function or block only. For example:

```
{  
var x=10;//local variable  
}
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

1. var data=200;//global variable

Difference between var, let, and const keyword in JavaScript

Scope

var	let	const
Variables declared with var are in the function scope.	Variables declared as let are in the block scope.	Variable declared as const are in the block scope.

Hoisting

Hoisting means that you can define a variable before its declaration.

var	let	const
Allowed	Not allowed	Not allowed

Reassign the value

To reassign a value is to reassign the value of a variable.

var	let	const
Allowed	Allowed	Not allowed

Redeclaration of the variable

The redeclaration of a variable means that you can declare the variable again.

var	let	const
Allowed	Not allowed	Not allowed

Data Types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.

- Primitive data type
- Non-primitive (reference) data type

Example:

- `var a=40;//holding number`
- `var b="Rahul";//holding string`

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine.

JavaScript primitive data types

- There are seven types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
bigint	To store large numeric value
Boolean	represents boolean value either false or true
Null	represents null i.e. no value at all
Undefined	represents undefined value
Symbol	

A **primitive** (primitive value, primitive data type) is data that is not an object and has no methods.

String

The string type is used to represent textual data. It is a set of "elements" of 16-bit unsigned integer values. Each element in the String occupies a position in the String.

The first element is at index 0, the next at index 1, and so on. The length of a String is the number of elements in it.

Example:

```
var str="cse";
```

```
document.write(str);
```

output: cse

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

output: c

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

output: 3

The `+` operator to append multiple strings together.

Example:

```
var str="cse"+"branch";
```

```
document.write(str);
```

output: csebranch

backslash character (`"\"`) can also be used at the end of each line to indicate that the string will continue on the next line.

Example:

```
var str="cse"+"branch\
```

```
in college";
```



```
document.writeln(str);
```

output: csebranch in college

Backticks: `Hello`

Backticks are generally used to include variables or expressions into a string.

Example:

```
Var name1='vijay';
```

```
Var name2="bhaskar"
```

```
Var result=`The names are ${name1} and ${name2}`
```

Some important methods used in strings are :

1. To extract a part of a string `substr()` method is used.
`Stringvariable.substr(startindex,lastindex).`

Example:

```
let text = "Hello world!";  
let result = text.substr(1, 4);
```

```
document.write(result);
```

output: ello

2. To combine two string concat() method is used. String1.concat(string2) .

Example:

```
var str="cse branch in college contains";
```

```
var str1="vits&vec students"
```

```
document.writeln(str.concat(str1));
```

output: cse branch in college containsvits&vec students

3. The charAt() method returns the character at a specified index (position) in a string.Example:

```
let text = "HELLO WORLD";
```

```
let letter = text.charAt(0);
```

```
document.write(letter);
```

output: H

Number

JavaScript has only one Number (numeric) data types. Number data type can store normal integer, floating-point values.

The number type has three symbolic values: +Infinity, -Infinity, and NaN (not-a-number).

The constants `Number.MAX_VALUE` or `Number.MIN_VALUE` can be used to specify largest and smallest values.

The number type has only one integer that has two representations: 0 is represented as `-0` and `+0`. ("`0`" is an alias for `+0`)

Example:

```
var num1 = 5;      // Numeric integer value
var num2 = 10.5;    // Numeric float value
var num3 = -30.47;  // Negative numeric float value
var num4 = 5E4;     // 5 x 10 Powers of 4 = 50000
var num5 = 5E-4     // 5 x 10 Powers of -4 = -50000
var num6 = 10 / 0;   // Number divide by zero, result is: Infinity
var num7 = 10 / -0; // Number divide by negative zero, result is: - Infinity
var num8 = 10, num9 = 12.5; // Multiple variable declare and initialize
```

Boolean

JavaScript Boolean type can have two value `true` or `false`. Boolean type is use to perform logically operator to determine condition/expression is true.

```
var val1 = true;
var val2 = false;
```

All the values below are considered `false` when they are converted/coerced to a boolean value.

1. `0` and `-0`

2. `null`
3. `false`
4. `NaN`
5. `undefined`
6. empty string: `"`

Null

JavaScript Null specifies variable is declare but values of this variable is empty.

Example:

```
var str = null;
```

Undefined

JavaScript uninitialized variables value are undefined.

Example:

```
var x;
```

```
document.write(x)
```

output: undefined

Symbol

A Symbol is a **unique** and **immutable** primitive value and may be used as the key of an Object property.

Example:

```
var s1 = Symbol();  
var s1 = Symbol('name');
```

Note:

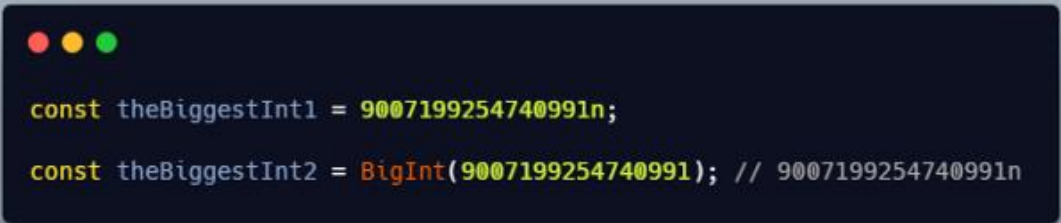
Symbols don't auto-convert to "strings"

Symbol value cannot convert to a number.

BigInt

BigInt type is a numeric primitive in JavaScript that can represent integers with arbitrary precision.

A BigInt is created by appending `n` to the end of an integer literal.



```
const theBiggestInt1 = 9007199254740991n;  
const theBiggestInt2 = BigInt(9007199254740991); // 9007199254740991n
```

`BigInt` is similar to a `Number`, but it still is different in some areas:

1. It cannot be used with methods in the built-in `Math` object.
2. It cannot be mixed with instances of `Number` in operations.
3. It is not strictly equal to a `Number` value

A `BigInt` behaves like a `Number` in cases where it is converted to a Boolean.

JavaScript non-primitive data types

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

OPERATORS AND EXPRESSIONS

An expression is formed with the combination of operators and operands.

An **operator** is a symbol that tells the compiler to perform certain mathematical and logical manipulations.

An **operand** is a data item on which the particular operation takes place.

Example: `x + y = 10;`

Here,

x, y and 10 are operands ; + and = are operators

JavaScript supports the following types of operators.

- Assignment Operators
- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Bitwise Operators
- Special Operators
- Assignment Operators

Assignment operators:

The Assignment operators are used to assign values to the operand. The following operators are known as JavaScript assignment operators:

Operator		Description	Example	
=		Assigns values from the right side operand to the left side operand	20+10 = 30	
+=		It adds the right operand to the left operand and assigns the result to the left operand	var a=20; a+=10; Now a = 30	
-=		It subtracts the right operand from the left operand and assigns the result to the left operand	var a=30; a-=10; Now a = 20	
=		It multiplies the right operand with the left operand and assigns the result to the left operand	var a=10; a=20; Now a = 200	

/=	It divides the left operand with the right operand and assigns the result to the left operand	var a=10; a/=2; Now a = 5
%=	It takes modulus using two operands and assigns the result to the left operand	var a=10; a%=2; Now a = 0

Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Adds two operands	10 + 20 = 30
-	Subtracts the second operand from the first	30 - 20 = 10
/	Divide the numerator by the denominator	20/10 = 2
*	Multiply two operands	5 * 5 = 25
**	Exponential	2***3=8
%	Outputs the remainder of an integer division	20 % 10 = 0
++	Increases an integer value by one	var a=20; a++; Now a = 21
-	Decreases an integer value by one	var a=20; a--; Now a = 19

Comparison Operators :

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Checks if two operands are equal or not. If yes, then the condition becomes true.	20==30 = false
===	Finds the identical (equal and of the same type)	10==20 = false
!=	Checks if two operands are equal or not. If the values are not equal, then the condition becomes true	20!=30 = true
!==	It implies that two values are not Identical	20!==20 = false

>	Checks if the value of the left operand is greater than the value of the right operand	30>10 = true
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand	20>=10 = true
<	This Checks if the value of the left operand is less than the value of the right operand	20<10 = false
<=	Checks if the value of the left operand is less than or equal to the value of the right operand	30<=10 = false

Logical Operators :

Logical operators are used to combine two or more relational expressions.

Operator	Description	Example	
&&	Logical AND – If both the operands are non-zero, then the condition becomes true	(10==20 && 20==33) = false	
	Logical OR – If any of the two operands are non-zero, then the condition becomes true.	(10==20 20==33) = false	
!	Logical NOT – Reverses the logical state of its operand.	!(10==20) = true	

Bitwise Operators:

The bitwise operators are used to perform bitwise operations on operands.

Operator	Description	Example	
----------	-------------	---------	--

&	Boolean AND operation on each bit of its integer arguments	$(10 == 20 \ \& \ 20 == 33) = \text{false}$	
 	It performs a Boolean OR operation on each bit of its integer arguments	$(10 == 20 \ \ 20 == 33) = \text{false}$	
^	This operator performs Bitwise XOR operation	$(10 == 20 \ ^ \ 20 == 33) = \text{false}$	
~	It is a unary operator and operates by reversing all the bits in the operand	$(\sim 10) = -10$	
<<	Moves all the bits in its first operand to the left by the number of places specified in the second operand.	$(10 << 2) = 40$	
>>	The left operand's value is moved right by the number of bits specified by the right operand.	$(10 >> 2) = 2$	
>>>	This operator is just like the >> operator, except that the bits shifted in on the left are always zero.	$(10 >>> 2) = 2$	

Special Operators:

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
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.

JavaScript Literals :

JavaScript Literals are the fixed value that cannot be changed.

No need to specify any type of keyword to write literals. Literals are often used to initialize variables in programs.

Names of variables are string literals.

Types of JavaScript Literals

Javascript literals hold different types of values

Integer Literals:

- It can be expressed in the decimal(base 10), hexadecimal(base 16) or octal(base 8) format.
- Decimal numeric literals consist of a sequence of digits (0-9) without a leading 0(zero).
- Hexadecimal numeric literals include digits(0-9), letters (a-f) or (A-F).
- Octal numeric literals include digits (0-7). A leading 0(zero) in a numeric literal indicates octal format.

Example 1:

```
var x=120 // decimal literal
```

```
var y=021434 // octal literal
```

```
var z=0x4567 // hexadecimal literal
```

Example 2:

```
<body>
<h1>JavaScript Numbers </h1>
<p> Number can be written of any base.</p>
Decimal number : <b id="no1"></b><br>
Octal number : <b id="no2"></b><br>
Hexadecimal number : <b id="no3"></b><br>
<script>
document.getElementById("no1").innerHTML = 100.25;
</script>
<script>
document.getElementById("no2").innerHTML = 073;
</script>
<script>
document.getElementById("no3").innerHTML = 0X8b;
```

</script>
</body>

JavaScript Numbers

Number can be written of any base.

Decimal number : 100.25

Octal number : 59

Hexadecimal number : 139

Floating Number Literals

Floating numbers are decimal numbers or fraction numbers or even can have an exponent part as well.

Example 1:

```
var x=78.90,
```

```
var y=-234.90,
```

```
var z=78.6e4 etc.
```

Example 2:

```
<h1>JavaScript Float </h1>
```

```
<p> Float Examples are : </p>
```

```
1. <b id="no1"></b><br>
```

```
2. <b id="no2"></b><br>
```

```
3. <b id="no3"></b><br>
```

```
<script>
```

```
document.getElementById("no1").innerHTML = 100.25;
```

```
</script>
```

```
<script>
```

```

document.getElementById("no2").innerHTML = -78.34;
</script>
<script>
document.getElementById("no3").innerHTML = 56e4;
</script>
</body>

```

JavaScript Float

Float Examples are :

1. 100.25
2. -78.34
3. 560000

String Literals:

A string literal is a combination of zero or more characters enclosed within a single(') or double quotation marks (").

Example 1:

```
var x='vijay'
```

```
var y="bhaskar"
```

Example 2:

```

<body>
<h1>JavaScript String </h1>
<p> String Examples are : </p>
1. <b id="no1"></b><br>
2. <b id="no2"></b><br>
3. <b id="no3"></b><br>
4. <b id="no4"></b><br>
<script>
var str = "This is first string";
document.getElementById("no1").innerHTML = str;
</script>

```

```
<script>
var strobj = new String("This is string store as object");
document.getElementById("no2").innerHTML = strobj;
</script>
<script>
var str = "This is first string";
document.getElementById("no3").innerHTML = str.length;
</script>
<script>
var str = "This is first string";
document.getElementById("no4").innerHTML = str+" This is second string";
</script>
</body>
```

JavaScript String

String Examples are :

1. **This is first string**
2. **This is string store as object**
3. **20**
4. **This is first string This is second string**

Array Literals:

An array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [] ' .

When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified. If no value is supplied it creates an empty array with zero length.

Example 1:

```
var color = [ ],
```

```
var fruits = ["Apple", "Orange", "Mango", "Banana"] (an array of four elements).
```

Example 2:

```

<body>
<h1>JavaScript Array </h1>
<p> Array Examples are : </p>
1. <b id="no1"></b><br>
2. <b id="no2"></b><br>
3. <b id="no3"></b><br>
4. <b id="no4"></b><br>
<script>
var fruits = ["Apple", "Orange", "Mango", "Banana"];
document.getElementById("no1").innerHTML = fruits;
</script>
<script>
document.getElementById("no2").innerHTML = fruits[0];
</script>
<script>
document.getElementById("no3").innerHTML = fruits[fruits.length - 1];
</script>
<script>
document.getElementById("no4").innerHTML = fruits.length;
</script>
</body>

```

JavaScript Array

Array Examples are :

1. **Apple,Orange,Mango,Banana**
2. **Apple**
3. **Banana**
4. **4**

Boolean Literals:

Boolean literals in JavaScript have only two literal values that are true and false.

Example 1:

```
var x=true // Boolean literal
```

```
var y=false // Boolean literal
```

Example 2:

```
<body>
<h1>JavaScript Boolean </h1>
<p> Boolean Examples are : </p>
<script>
document.write('Boolean(12) is ' + Boolean(12));
document.write('<br>');
document.write('Boolean("Hello") is ' + Boolean("Hello"));
document.write('<br>');
document.write('Boolean(2 > 3) is ' + Boolean(2 > 3));
</script>
</body>
```

JavaScript Boolean

Boolean Examples are :

```
Boolean(12) is true
Boolean("Hello") is true
Boolean(2 > 3) is false
```

Object Literals:

It is a collection of key-value pairs enclosed in curly braces({}). The key-value pair is separated by a comma.

Example 1

```
var userObject = { },
```

```
var games = {cricket :11, chess :2, carom: 4} // Object literal
```

Example 2:

```
<body>
<h1>JavaScript Object </h1>
<p> Object Examples are : </p>
<p id= "no1"> </p>
```



```
<script>
// Create an object:
var student = {firstName:"John", lastName:"D", "rno" : 23, "marks" : 60 };
// Displaying some data from the object:
document.getElementById("no1").innerHTML =
student.firstName + " got " + student.marks + " marks.";
</script>
</body>
```

JavaScript Object

Object Examples are :

John got 60 marks.

JavaScript Function :

A function is a self contained program segment (or) a sub program (or) a module that carries out some specific, well defined task.

Advantages:

1. Avoid repetition of codes.
2. Increases program readability.
3. Divide a complex problem into simpler ones.
4. Reduces chances of error.
5. Create, Modify and debugging a program becomes easier by using function.
6. It can be executed any number of times.

Function Types:

functions can be classified into two categories,

- Library functions
- User-defined functions

Library functions :

Functions that are defined by the compiler are known as library functions.
Example: `Math.sqrt()` is a function to calculate the square root of a number.

User defined functions:

User-defined functions are those functions which are defined by the user at the time of writing program.

These functions are made for code reusability, for saving time & space.

Declaring a Function

The syntax to declare a function is:

```
function nameOfFunction () {  
  
    // function body  
  
}
```

- A function is declared using the function keyword.
- The basic rules of naming a function are similar to naming a variable. It is better to write a descriptive name for your function. For example, if a function is used to add two numbers, you could name the function add or addNumbers.

The body of function is written within {}.

Example:

```
// declaring a function named greet()  
function greet() {  
    document.write("Hello there");  
}
```

Calling a Function

Calling a Function is used to execute the function.

```
// function call  
greet();
```

Example:

```
<html>

<head>

    <title>Functions!!!</title>

    <script type="text/javascript">

function myFunction()

{

    document.write("This is a simple function.<br />");

}

        myFunction();

    </script>

</head>

<body>

</body>

</html>
```

Function Parameters:

A function can also be declared with parameters. A parameter is a value that is passed when declaring a function.

```
function greet(name) {  
    // code  
}  
greet(name);  
// code
```

The diagram illustrates the relationship between a function declaration and its call. A teal arrow points from the closing curly brace of the function definition to the text "function call". Another teal arrow points from the "greet(name);" line back to the opening curly brace of the function definition. A third teal arrow points from the "greet(name);" line to the "function call" text. The text "function call" is positioned to the right of the code.

```
// program to add two numbers using a function
```

```
// declaring a function
```

```
function add(a, b) {  
    document.write(a + b);  
}
```

```
// calling functions
```

```
add(3,4);
```

```
add(2,9);
```

The `return` statement can be used to return the value to a function call.

```
// program to add two numbers
```

```
// declaring a function
```

```
function add(a, b) {  
    return a + b;  
}
```

```
// take input from the user
```

```
let number1 = parseFloat(prompt("Enter first number: "));
```

```
let number2 = parseFloat(prompt("Enter second number: "));
```

```
// calling function
```

```
let result = add(number1,number2);
```

```
// display the result
```

```
document.write("The sum is " + result);
```

Arguments Passing:

The following are the types of arguments that can be passed to a JavaScript Function:

1. Argument Passed by Value
2. Default Argument
3. Arguments Object
4. Objects Passed by Reference

Argument Passed by Value

In pass by value, values of the variables are passed to the arguments instead of address.

Example:

```
function mul(a, b) {  
    document.write(a * b);  
}  
  
// calling functions  
mul(3,4);  
mul(2,9);
```

Default Argument

The default arguments are those which are initialized with default values. The default values are used when the value of the argument is not passed.

Syntax

```
function Name(paramet1 = value1, paramet2 = value2 .. .) {  
    // statements  
}
```

Example

```
<script type="text/javascript">  
    function displayCountry(country = "Norway"){  
        document.write("I am from " + country);  
    }
```

```
displayCountry("Sweden")  
displayCountry("India")  
displayCountry()  
displayCountry("Brazil")  
</script>
```

OUTPUT:

I am from Sweden
I am from India
I am from Norway
I am from Brazil

Arguments Object

The arguments object is an inbuilt object which contains all the argument values passed to the function.

Properties

`length` → contains the number of arguments passed to the function

`callee` → contains the currently executing function reference

Example 1:

```
function test() {  
  document.write (arguments.callee);  
  document.write (arguments.length);  
}
```

`test(1,2,3);`

Output

`[Function: test] 3`

Example 2:

```
<script>  
  var sum = 0;  
  function addAll(){  
    for (var i = 0; i<arguments.length; i++){  
      sum+=arguments[i];  
    }  
    document.write("sum of numbers :"+sum);  
  }  
</script>
```

`addAll(1, 2, 3, 4, 5, 6, 7, 8, 9,10);`

`</script>`

Output: sum of numbers :55

Objects Passed by Reference

In pass by reference, address (or reference) of the variable is passed to the function as arguments.

if made any changes inside the function definition; those changes are effected on the original variable.

In Javascript objects and arrays are passed by reference.

<script>

```
function callByReference(varObj) {
```

```
    document.write("Inside Call by Reference Method");
```

```
    varObj.a = 100;
```

```
    document.write(varObj);
```

```
}
```

```
let varObj = {
```

```
    a: 1
```

```
};
```

```
document.write("Before Call by Reference Method");
```

```
document.write(varObj);
```

```
callByReference(varObj)
```

```
document.write("After Call by Reference Method");
```

```
document.write(varObj);
```

```
}
```

```
</script>
```

Before Call by Reference Method

```
{a: 1}
```

Inside Call by Reference Method

```
{a: 100}
```

After Call by Reference Method

```
{a: 100}
```

Note:

- In JavaScript, functions are called Function Objects because like objects
 1. functions also **have properties and methods.**
 2. Functions can be **stored in a variable or an array**
 3. Functions can be **passed as arguments to other functions**
 4. Functions can be **returned from functions.**

This concept is called 'functions as first-class citizens' . first-class citizenship simply means "being able to do what everyone else can do".

Anonymous Function

Writing a function without name is called anonymous function.

Example:

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

Function Expression

A function expression is defined by assigning a function definition to a javascript variable.

Example:

```
<script>
```

```
var add=function(a,b){  
  return a+b;
```



```
}  
let result=add(5,61);  
document.write("sum of 2 no's:"+result);  
</script>
```

Output: sum of 2 no's:66

immediately invoked function expression (IIFE)

JavaScript immediately invoked function expression is a function defined as an expression and executed immediately after creation. The following shows the syntax of defining an immediately invoked function expression:

```
(function(){  
    //...  
})();
```

Example:

```
<script>  
let sum = (function(a,b){  
    return a + b;  
})(10, 20);  
document.write("sum of 2 no's:"+sum);  
</script>
```

Output:

sum of 2 nos:30

Arrow Functions

Arrow functions are **anonymous functions** . They don't return any value and can declare without the function keyword.

Arrow functions cannot be used as the constructors.

Arrow functions allow us to write shorter function syntax.

Arrow functions are similar to lambda functions in some other programming languages, such as Python.

Syntax for defining the arrow function:

```
const functionName = (arg1, arg2, ?..) => {  
  //body of the function  
}
```

```
<script>  
const test=(a,b)=>a+b;  
let sum=test(10,21);  
document.write("sum of 2 no's:"+sum);  
</script>  
sum of 2 no's:31
```

Callback Functions

- A function which takes arguments as another function is known as callback functions.
- Functions that do this are called higher-order functions
- A JavaScript callback is a function which is to be executed after another function has finished execution.
- callback functions are useful to develop asynchronous JavaScript code.

```
<script>  
  
// function  
  
function greet(name, callback) {  
  
  document.write('Hi' + ' ' + name+"<br>");  
  
  callback();  
  
}  
  
// callback function
```

```
function callMe() {  
    document.write('I am callback function');  
}  
  
// passing function as an argument  
greet('bhaskar', callMe);  
</script>
```

Output:

Hi bhaskar
I am callback function

RECURSION

A function which calls itself is known as recursion.

Example:

```
function fact (n)  
{  
    if (n == 1)  
        return 1;  
    else  
        return (n * fact (n - 1));  
}
```

Objects:

JavaScript uses objects to perform many tasks and therefore is referred to as an **object-based programming language**.

An object can be created with figure brackets {...} with an optional list of *properties*. A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything.

Creating Empty Objects:

An empty object (“empty cabinet”) can be created using one of two syntaxes:

```
let user = new Object(); // "object constructor" syntax
let user = {}; // "object literal" syntax
```

Creating Objects With properties:

Syntax:

```
let objectName = {
  key1:value1;
  key2:value2;
  .....
};
```

Example:

```
let user1 = { // an object
  name: "bhaskar", // by key "name" store value "John"
  technology: javascript; // by key "age" store value 30
};
```

- Property values can be accessed using 2 ways:
 1. dot notation
 2. square bracket notation

Dot notation:

Dot notation is used to access single word properties.

Example:

```
<script type="text/javascript">
```

```
var user1 = {
  name: "bhaskar",
  technology: "javascript" ,
};

document.write("user name:"+user1.name+"<br>");
```

```
document.write("Technology is:"+user1.technology);  
</script>
```

Output:

```
user name: bhaskar  
Technology is: javascript
```

Square Bracket Notation:

Square Bracket Notation is used to access multiword properties.

Example:

```
<script type="text/javascript">  
var user1 = {  
  name: "bhaskar",  
  technology:"java script",  
  'work experience': "4 years",  
};  
document.write("user name:"+user1.name+"<br>");  
document.write("Technology is:"+user1.technology+"<br>");  
document.write("Experience:"+user1['work experience']);
```

output:

```
user name:bhaskar  
Technology is: java script  
Experience:4 years
```

The "for. In" loop

It is used to iterate through the properties of object.

Example:

```
<body>
```

```
<script type="text/javascript">
var user1 = {
  name: "bhaskar",
  technology:"java script",
  'work experience': "4 years",
};
for(let key in user1)
{
    document.write(key+":",user1[key]+"<br>");
}
</script>
```

Output:

```
name:bhaskar
technology:java script
work experience:4 years
```

delete can be used to remove an object (or) property.

Syntax:

delete objectName;

(or)

delete objectName.propertyName

Example:

To delete property technology in user1 object is as follows;

```
delete user1.technology;
```

Checking if a property exists

in operator is used to check if a property exists in an object or not.

Syntax:

PropertyName in objectName;

The result can be either true or false.

Example:

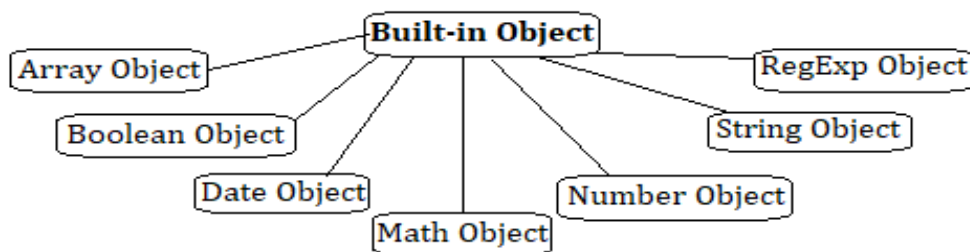
```
document.write('name' in user1+"<br>");
```

output: true;

Built-in Objects:

JavaScript has several built-in or core language objects. These built-in objects are available regardless of window content and operate independently of whatever page your browser has loaded.

- Built-in objects are not related to any Window or DOM object model.
- These objects are used for simple data processing in the JavaScript.



Math Object:

- Math object is a built-in static object.
- It is used for performing complex math operations.

Math Properties

| Math Property | Description |
|---------------|----------------------------------|
| SQRT2 | Returns square root of 2. |
| PI | Returns Π value. |
| E \ | Returns Euler's Constant. |
| LN2 | Returns natural logarithm of 2. |
| LN10 | Returns natural logarithm of 10. |
| LOG2E | Returns base 2 logarithm of E. |
| LOG10E | Returns 10 logarithm of E. |

Math Methods

| Methods | Description |
|---------|---|
| abs() | Returns the absolute value of a number. |
| acos() | Returns the arccosine (in radians) of a number. |
| ceil() | Returns the smallest integer greater than or equal to a number. |
| cos() | Returns cosine of a number. |
| floor() | Returns the largest integer less than or equal to a number. |
| log() | Returns the natural logarithm (base E) of a number. |
| max() | Returns the largest of zero or more numbers. |

| | |
|-------|--|
| min() | Returns the smallest of zero or more numbers. |
| pow() | Returns base to the exponent power, that is base exponent. |

Example 1: Simple Program on Math Object Methods

```
<body>
  <script type="text/javascript">

    var value = Math.abs(-20);
    document.write("ABS Test Value : " + value + "<br>");

    var value = Math.acos(-1);
    document.write("ACOS Test Value : " + value + "<br>");

    var value = Math.asin(1);
    document.write("ASIN Test Value : " + value + "<br>");

  </script>
</body>
```

Output

```
ABS Test Value : 20
ACOS Test Value : 3.141592653589793
ASIN Test Value : 1.5707963267948966
```

Example 2: Simple Program on Math Object Properties

```
<body>
  <script type="text/javascript">
    var value1 = Math.E
    document.write("E Value is : " + value1 + "<br>");

var value4 = Math.PI
    document.write("PI Value is : " + value4 + "<br>");
  </script>
</body>
```

output:

E Value is :2.718281828459045

PI Value is :3.141592653589793

Date Object:

- Date is a data type.
- Date object manipulates date and time.
- Date() constructor takes no arguments.
- Date object allows you to get and set the year, month, day, hour, minute, second and millisecond fields.

Syntax:

```
var variable_name=new Date();
```

Example:

```
var current_date=newDate();
```

Date Methods

Methods	Description
Date()	Returns current date and time.
getDate()	Returns the day of the month.
getDay()	Returns the day of the week.
getFullYear()	Returns the year.
getHours()	Returns the hour.
getMinutes()	Returns the minutes.
getSeconds()	Returns the seconds.

getMilliseconds()	Returns the milliseconds.
getTime()	Returns the number of milliseconds since January 1, 1970 at 12:00 AM.
getTimezoneOffset()	Returns the timezone offset in minutes for the current locale.
getMonth()	Returns the month.
setDate()	Sets the day of the month.
setFullYear()	Sets the full year.
setHours()	Sets the hours.
setMinutes()	Sets the minutes.
setSeconds()	Sets the seconds.
setMilliseconds()	Sets the milliseconds.
setTime()	Sets the number of milliseconds since January 1, 1970 at 12:00 AM.
setMonth()	Sets the month.
toDateString()	Returns the date portion of the Date as a human-readable string.
toLocaleString()	Returns the Date object as a string.
toGMTString()	Returns the Date object as a string in GMT timezone.
valueOf()	Returns the primitive value of a Date object.

Example : JavaScript Date() Methods Program

```

<script type="text/javascript">
var d = new Date();
document.write("<b>Locale String:</b> " + d.toLocaleString()+"<br>");

```

```
document.write("<b>Hours:</b> " + d.getHours()+"<br>");  
document.write("<b>Day:</b> " + d.getDay()+"<br>");  
document.write("<b>Month:</b> " + d.getMonth()+"<br>");  
document.write("<b>FullYear:</b> " + d.getFullYear()+"<br>");  
document.write("<b>Minutes:</b> " + d.getMinutes()+"<br>");  
</script>
```

Locale String: 11/26/2022 3:09:35 PM

Hours: 15

Day: 6

Month: 10

FullYear: 2022

Minutes: 9

String Object:

- String objects are used to work with text.
- It works with a series of characters.

Syntax:

```
var variable_name=new String(string);
```

Example:

```
var s=new String(string);
```

String Properties

Properties	Description
length	It returns the length of the string.
prototype	It allows you to add properties and methods to an object.
constructor	It returns the reference to the String function that created the object.

String Methods

Methods	Description
charAt()	It returns the character at the specified index.
charCodeAt()	It returns the ASCII code of the character at the specified position.
concat()	It combines the text of two strings and returns a new string.
indexOf()	It returns the index within the calling String object.
match()	It is used to match a regular expression against a string.
replace()	It is used to replace the matched substring with a new substring.
search()	It executes the search for a match between a regular expression.
slice()	It extracts a session of a string and returns a new string.
split()	It splits a string object into an array of strings by separating the string into the substrings.
toLowerCase()	It returns the calling string value converted lower case.
toUpperCase()	Returns the calling string value converted to uppercase.

Example:

```
<script type="text/javascript">
```

```
    var str = "Web Technologies";
```

```
    document.write("<b>Char At:</b> " + str.charAt(1)+"<br>");
```

```
    document.write("<b>CharCode At:</b> " + str.charCodeAt(2)+"<br>");
```

```
    document.write("<b>Index of:</b> " + str.indexOf("ch")+"<br>");
```

```
document.write("<b>Lower Case:</b> " + str.toLowerCase()+"<br>");  
document.write("<b>Upper Case:</b> " + str.toUpperCase()+"<br>");  
</script>
```

Output:

Char At: e

CharCode At: 98

Index of: 6

Lower Case: web technologies

Upper Case: WEB TECHNOLOGIES

Boolean Object:

- The Boolean object is wrapper class and member of global objects.
- It is used to convert a non-Boolean value to a Boolean value (true or false).
- If the Boolean object has no initial value or if it is 0, -0, null, "", false, undefined, or NaN, the object is set to false. Otherwise it is true (even with the string "false")!
- Properties of the Boolean object
 - Constructor - Returns the function that created the Boolean object.
 - Prototype - Add properties and methods to an object.
- Methods of the Boolean object
 - toSource() - Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
 - toString() - Returns a string of either "true" or "false" depending upon the value of the object.
 - valueOf() - Returns the primitive value of the Boolean object

Example:

```
<body>  
<h2>JavaScript Booleans</h2>  
<p>Display the value of Boolean(10 > 9):</p>  
<p id="demo"></p>  
<script>  
document.getElementById("demo").innerHTML = Boolean(10 > 9);  
</script>
```

JavaScript Booleans

Display the value of Boolean(10 > 9):

true

Number Object:

- The Number objects represents numerical date, either integers or floating-point numbers.
- A Number objects are created using the Number() constructor var num = new number(value);
- Properties of Number object
 - Constructor - Returns the function that created the Number object.
 - MAX VALUE - Returns maximum numerical value possible in JavaScript.
 - MIN VALUE - Returns minimum numerical value possible in JavaScript.
 - NEGATIVE INFINITY - Represent the value of negative infinity.
 - POSITIVE INFINITY - Represent the value of infinity.
 - Prototype - Add properties and methods to an object.
- Methods of Number object
 - toExponential() - Converts a number into exponential notation.
 - toFixed() - Formats a number with a specific number of digits to the right of the decimal.
 - toLocaleString() - Returns a string value version of the current number in a format that may vary according to a browser's locale settings.
 - toPrecision() - Defines how many total digits to display of a number.
 - toString() - Returns the string representation of the number's value.
 - valueOf() - Returns the number's value.

```
<script>
var x=102;//integer value
var y=102.7;//floating point value
var z=13e4;//exponent value, output: 130000
var n=new Number(16);//integer value by number object
document.write(x+" "+y+" "+z+" "+n);
</script>
```

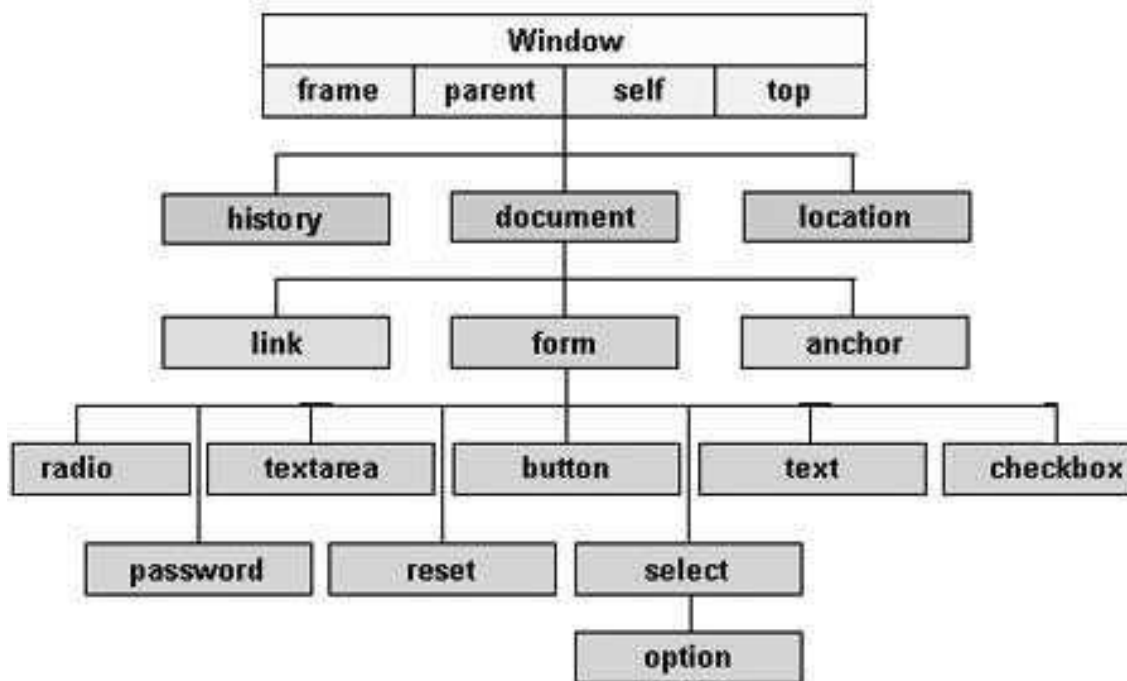
Output: 102 102.7 130000 16

Document Object Model (DOM) :

Document Object Model (DOM) is an application programming interface (API) for manipulating HTML documents.

The DOM represents an HTML document as a tree of nodes. The DOM provides functions that allows to add, remove, and modify parts of the document effectively.

DOM is cross-platform and language-independent ways of manipulating HTML and XML documents.



The DOM represents an HTML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
```

```
<head>
```

```
<title>JavaScript DOM</title>
```

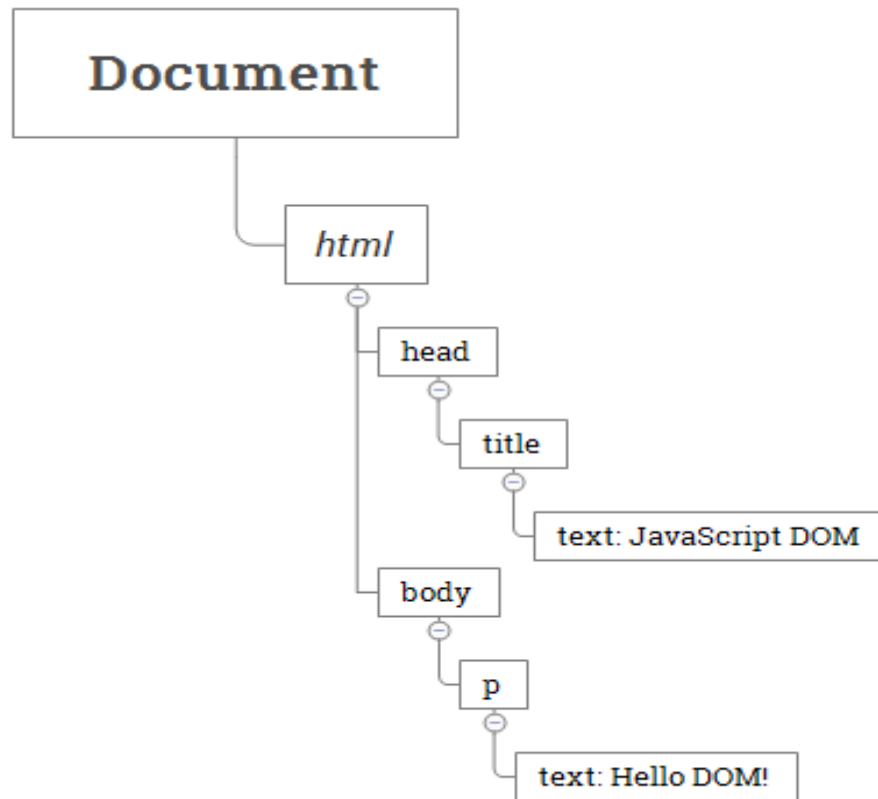
```
</head>
```

```
<body>
```

```
<p>Hello DOM!</p>
```


</body>

</html>



The document object has many properties and methods.

properties of document object are:

Properties	Description
cookie	returns a report that contains all the visible and unexpired cookies associated with the document

Properties	Description
domain	returns the domain name of the server from which the document has originated
lastModified	returns the date on which document was last modified
documentMode	returns the mode used by the browser to process the document
readyState	returns the loading status of the document.
referrer	returns the URL of the documents referred to in an HTML document
title	returns the name of the HTML document defined between the starting and ending tags of the TITLE element
URL	returns the full URL of the HTML document.

Methods of document object are:

Methods	Description	Syntax
open()	opens an HTML document to display the output	<pre>document.open(mimetype, replace)</pre> Copy
close()	closes an HTML document	<pre>document.close()</pre> Copy
write()	Writes HTML expressions or JavaScript code into an HTML document	<pre>document.write(exp1, exp2, ...)</pre> Copy
writeln()	write a new line character after each HTML expressions or JavaScript Code	<pre>document.writeln(exp1, exp2, ...)</pre> Copy
getElementById()	returns the reference of first element of an HTML document with the specified id.	<pre>document.getElementById(id)</pre> Copy
getElementsByName()	returns the reference of an element with the specified name	<pre>document.getElementsByName(name)</pre> Copy

Methods	Description	Syntax
getElementsByTagName()	returns all elements with the specified	

Example:

```

<!doctype html>
<head>
    <title>JS Document Methods Example</title>
</head>
<body>
    <h3>Document Methods Example</h3>
    <p id="demo">It Will change</p>

    <script>
        // Open document
        document.open();
        // writing
        document.write("Hello" + "<br>");
        document.getElementById("demo").innerHTML = "Set by ID";
    </script>
</body>
</html>

```

Output:

Document Methods Example

Set by ID

Hello

Example2:

```

<head>
    <script type="text/javascript">
        function check()
        {
            var username = document.getElementById("uname");
            var password = document.getElementById("pass");
            if(username.value=="abc" && password.value=="123")
                alert("Hello!!! You are successfully signed in");

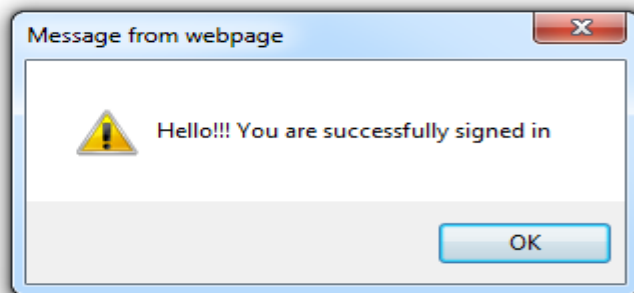
```

```
else  
  
    alert("Invalid Username and Password!!!");  
  
}  
</script>  
</head>  
<body>  
<h2>Login</h2>  
<form>  
    Username:<input type = "text" id="uname"><br>  
    Password:<input type = "password" id="pass"><br>  
    <input type="button" onClick="check()" Value="Sign In">  
</form>  
</body>
```

Login

Username:

Password:



window Object:

The window object provides methods for manipulating browser windows.

window object properties are:

Property	Description
closed	returns a boolean value that specifies whether a window has been closed or not
document	specifies a document object in the window.
history	specifies a history object for the window.
frames	specifies an array of all the frames in the current window
defaultStatus	specifies the default message that has to be appeared in the status bar of Window.
innerHeight	specifies the inner height of the window's content area.
innerWidth	specifies the inner width of the window's content area.

Property	Description
length	specifies the number of frames contained in the window.
location	specifies the location object for window
name	specifies the name for the window
top	specifies the reference of the topmost browser window.
self	returns the reference of current active frame or window.
parent	returns the parent frame or window of current window.
status	specifies the message that is displayed in the status bar of the window when an activity is performed on the Window.
screenleft	specifies the x coordinate of window relative to a user's monitor screen

Property	Description
scrollTop	Specifies the y coordinate of window relative to a user's monitor screen
screenX	specifies the x coordinate for window relative to a user's monitor screen
screenY	Specifies the y coordinate for window relative to a user's monitor screen

window object methods are:

Method	Description
alert()	specifies a method that displays an alert box with a message an OK button.
blur()	specifies a method that removes the focus from the current window.
clearInterval()	specifies a method that clears the timer, which is set by using setInterval() method.

Method	Description
close()	specifies a method that closes the current window.
confirm()	specifies a method that displays a dialogue box with a message and two buttons OK and Cancel
focus()	specifies a method that sets the focus on current window.
open()	specifies a method that opens a new browser window
moveTo()	specifies a method that moves a window to a specified position
moveBy()	specifies a Method that moves a window relative to its current position.
prompt()	specifies a method that prompts for input.
print()	specifies a method that sends a print command to print the content of the current window.

Method	Description
setTimeout()	specifies a method that evaluates an expression after a specified number of milliseconds.
setInterval()	specifies a method that evaluates an expression at specified time intervals (in milliseconds)
resizeBy()	specifies the amount by which the window will be resized
resizeTo()	used for dynamically resizing the window
scroll()	scrolls the window to a particular place in document
scrollBy()	scrolls the window by the given value
stop()	this method stops window loading

Example:
<body>

<h3>Perform various operations on Window object</h3>

```
<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();
}
</script>
```

</body>

Perform various operations on Window object

Open a Window

Close the Window

Show Alert in Window

Arrays:

an array is an ordered list of values. Each value is called an element specified by an index.

characteristics:

An array can hold values of mixed types. i.e, array stores elements with the types number, string, boolean, and null.

The size of an array is dynamic and auto-growing.

Creating JavaScript arrays

There are 3 ways to construct array in JavaScript

1. By array literal
2. By creating instance of Array directly (using new keyword)
3. By using an Array constructor (using new keyword)

JavaScript array literal:

Syntax:

```
var arrayname=[value1,value2.....valueN];
```

Example:

```
var emp=["vijay","bhaskar","ramu"];
```

JavaScript Array directly:

new keyword is used to create instance of array.

Syntax:

```
var arrayname=new Array();
```

Example:

```
var emp = new Array();  
emp[0] = " vijay ";  
emp[1] = " bhaskar ";  
emp[2] = " ramu ";
```

JavaScript array constructor :

create instance of array by passing arguments in constructor

Syntax:

```
var arrayname=new Array(arguments);
```

Example:

```
var emp = new Array("vijay","bhaskar","ramu");
```

properties of the Array object are:

Sr.No	Property & Description
1	Constructor Returns a reference to the array function that created the object.
2	Index The property represents the zero-based index of the match in the string.
3	Input This property is only present in arrays created by regular expression matches.
4	Length Reflects the number of elements in an array.
5	prototype The prototype property allows you to add properties and methods to an object.

Example:

```
<script>
var emp=new Array("Jai","Vijay","Smith");
for (i=0;i<emp.length;i++)
{
document.write(emp[i] + "<br>");
}
</script>
```

Array Methods

Methods	Description
concat()	It returns a new array object that contains two or more merged arrays.
fill()	It fills elements into an array with static values.

filter()	It returns the new array containing the elements that pass the provided function conditions.
For Each()	It invokes the provided function once for each element of an array.
includes()	It checks whether the given array contains the specified element.
indexOf()	It searches the specified element in the given array and returns the index of the first match.
map()	It calls the specified function for every array element and returns the new array
pop()	It removes and returns the last element of an array.
push()	It adds one or more elements to the end of an array.
reverse()	It reverses the elements of given array.
shift()	It removes and returns the first element of an array.
slice()	It returns a new array containing the copy of the part of the given array.

sort()	It returns the element of the given array in a sorted order.
splice()	It add/remove elements to/from the given array.
toLocaleString()	It returns a string containing all the elements of a specified array.
toString()	It converts the elements of a specified array into string form, without affecting the original array.
unshift()	It adds one or more elements in the beginning of the given array.

Example:

```

<script type="text/javascript">
let fruits = ["Apple", "Orange", "Pear"];
document.write( fruits.pop()+"<br>" );
document.write( fruits+"<br>");
fruits.push("Pear");
document.write( fruits+"<br>" );
fruits.shift()
document.write( fruits+"<br>" );
fruits.unshift('Apple');
document.write( fruits +"<br>");
</script>

```



Pear

Regular Expression:

- A Regular Expression is an object that describes a pattern of characters.
- A Regular Expression (also “**regexp**”, or just “**reg**”) is a sequence of characters that forms a search pattern.
- Regular Expression are useful for searching and replacing the characters in the string that match a pattern.
- For example, regular expressions can be used to validate form fields like email addresses and phone numbers, to perform all types of text search and text replace operations, for counting specific characters in a string.
- A regular expression is very similar to a mathematical expression, except it tells the browser how to manipulate text rather than numbers by using special symbols as operators.

Creating Regular Expression

- A regular expression could be defined by using two ways:

1) Using RegExp constructor

Syntax - `var pattern = new RegExp(pattern [, flags]);`

Example - `var re1 = new RegExp("xyz", "i");`

2) Using literal

Syntax - `var pattern = /pattern/flags;`

Example - `var re2 = /xyz/;`

Language of Regular Expression

- The language of regular expression consists of following:

- 1) Character Classes (Brackets)
- 2) Metacharacters
- 3) Quantifiers

1) Character Classes (Brackets)

- Characters can be grouped by putting them in Square brackets.
- Brackets are used to find a range of characters.
- [...] matches any one of the characters between the brackets
- [^...] matches any one character, but not one of those inside the brackets
- (x|y) matches any of the alternatives specified.

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x)	A grouping or subpattern, which is also stored for later use
(x y)	Find any of the alternatives specified

2) Metacharacters

- Metacharacters are characters with a special meaning within the language of regular expression.

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character

\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NULL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd

3)Quantifiers

- Quantifiers match a number of instances of a character, group, or character class in a string.
- The following table list the quantifiers:

Quantifier	Description
*	Match zero or more times.
+	Match one or more times.
?	Match zero or one time.
{ n }	Match exactly <i>n</i> times.
{ n , }	Match at least <i>n</i> times.
{ n , m }	Match from <i>n</i> to <i>m</i> times.

RegExp class functions:

Here is a list of the methods associated with RegExp along with their description.

1) exec() method

- The exec method searches string for text that matches regexp. If it finds a match, it returns an array of results; otherwise, it returns null.

Syntax

```
RegExpObject.exec( string );
```

- It returns the matched text if a match is found, and null if not.

2) test() method

- The test method searches string for text that matches regexp. If it finds a match, it returns true; otherwise, it returns false.

Syntax

```
RegExpObject.test( string );
```

- It returns the matched text if a match is found, and null if not.

String class functions:

1) match(pattern)

Searches for a matching pattern. Returns an array holding the results, or null if no match is found

2) replace(pattern1, pattern2)

Searches for pattern1. If the search is successful pattern1 is replaced with pattern2

3) search(pattern)

Searches for pattern in the string. If the match is successful, the index of the start of the match is returned. If the search fails, the function returns -1.

Sample Programs

1. Write a JavaScript to test if string contains the letter 'a' or 'c' or both.

```
<html>

<body>

<script language="javascript">

var input = prompt('Enter some text')

re = /[ac]/

if(re.test(input)) {

    alert('The string contains letter a or c or both')

}

else {

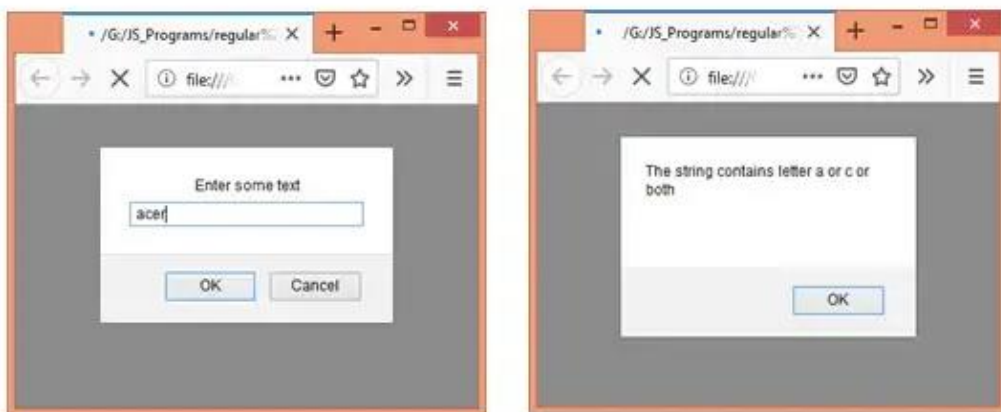
    alert('String does not contain a or c or both')

}

</script>

</body>

</html>
```



2. Develop a program to check for valid email id entered by user.

```
<html>

<head>

<title>Check Email ID</title>

<script>

function checkEmail()

{

var email = document.getElementById('email').value

var regex = /^[a-zA-Z0-9_\.]+\@([a-zA-Z0-9_\.]+\.[a-zA-Z]{2,5})$/

var res = regex.test(email)

if(!res) {

alert('Please enter valid email id')

}

else {

alert('Thank you')

}

}

</script>

</head>

<body>

<form name="myform" action="#" method="post">

Enter Email ID <input type="text" id="email" /><br/>
```

```

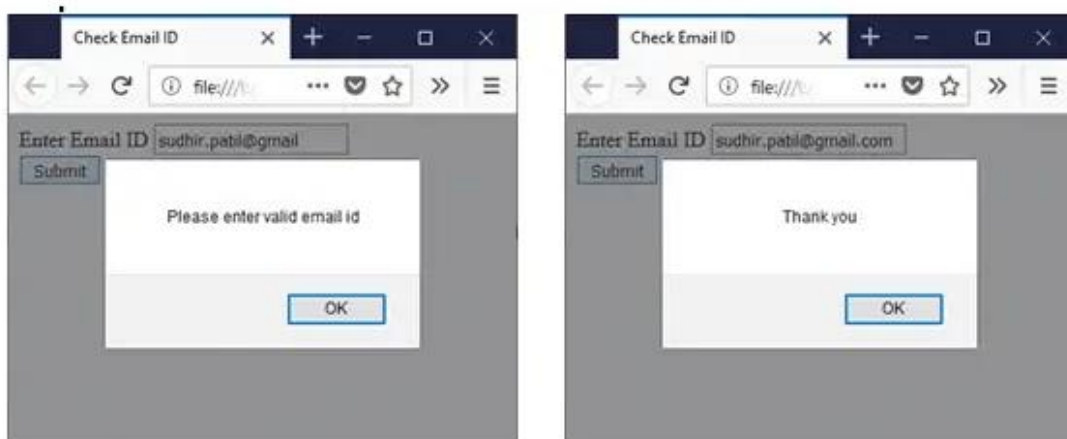
<input type="button" value="Submit" onclick="checkEmail()"/>

</form>

</body>

</html>

```



3. Write a webpage that accepts Username and adhaarcad as input texts. When the user enters adhaarcad number ,the JavaScript validates card number and displays whether card number is valid or not. (Assume valid adhaar card format to be nnnn.nnnn.nnnn or nnnn-nnnn-nnnn).

```

<html>

<head>

<script>

function check()

{

var card_no = document.getElementById('card_no').value

var re1 = /\d{4}\.\d{4}\.\d{4}/

var re2 = /\d{4}-\d{4}-\d{4}/

```

```
var res = re1.test(card_no) || re2.test(card_no)

if(res)

    alert('Valid format')

else

    alert('Invalid format')

}

</script>

</head>

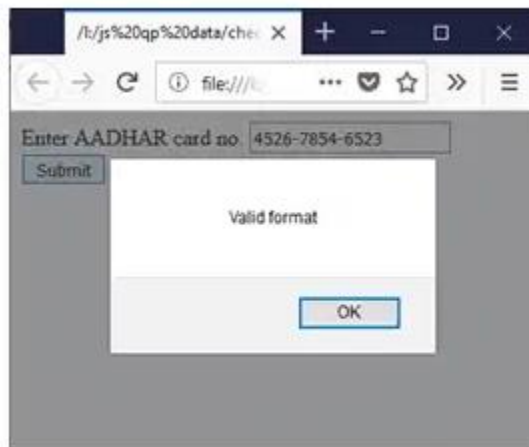
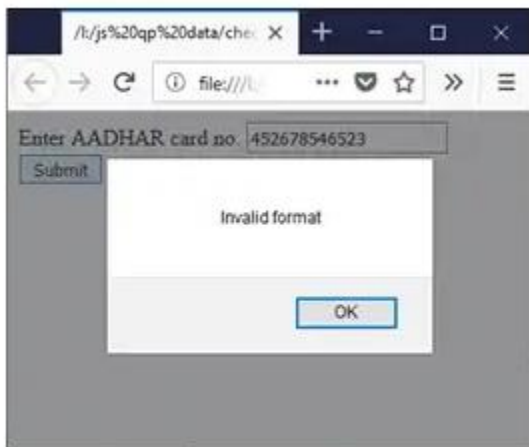
<body>

Enter AADHAR card no. <input type="text" id="card_no"/><br>

<input type="button" value="Submit" onclick="check()"/>

</body>

</html>
```



Exceptions:

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions. When an error occurs within a method, the method creates an object and hands it off to the runtime system.

A few reasons why exceptions occur are listed below:

- Dividing a number by zero: This results in infinity, thus throwing an exception.
- When a requested file does not exist in the system.
- When the user provides the wrong input.
- When the network drops during communication.

JavaScript error types

Different errors may occur while executing a JavaScript code. There are three types of errors.

1. **Syntax Errors**: These are errors that cannot be interpreted by the computer. These errors stop the program from working.

In JavaScript, these errors are:

- Spelling errors (wrong spelling such as fiction instead of function).
 - The omission of important characters, such as not using a semicolon to end a statement.
 - Use of the wrong indentation.
2. **Runtime Errors**: These errors take place during execution. The errors get detected when your program runs. It crashes or raises an exception. Thus, exception handlers handle exception errors.

These errors are often caused by:

- The program not being able to find data because it does not exist.
- The data being an invalid type of data.

3. **Logical Errors**: These types of errors do not throw an error or an exception at all. This is because they result from the code not doing what the developer intends it to. It's challenging to find logical errors. They can only be found through thorough testing.

Error objects:

When a runtime error occurs, it stops the code and raises an error object.

The error object has two properties:

- **Name**: It gives the error name.
- **Message**: It sets or returns the error message in the form of a string.

JavaScript uses six types of error objects. These error objects are the foundation of exception handling.

- **EvalError**: The EvalError function indicates the error that occurred in the eval() function. It's a global function that evaluates the JavaScript string. JavaScript does not throw this exception anymore.
- **RangeError**: RangeError exceptions occur when a numeric value is outside the specified range.
- **ReferenceError**: A ReferenceError exception occurs when undeclared variables are used. These exceptions commonly occur due to spelling errors on variables.
- **Syntax Error**: A Syntax Error exception occurs when JavaScript language rules get broken.
- **TypeError**: A TypeError exception occurs when a value is different from the one expected.
- **URIError**: A URIError exception is raised by encodeURIComponent() and decodeURI() methods.

How to handle exceptions in JavaScript

JavaScript handles exceptions in try-catch-finally statements and throw statements.

- A **try-catch-finally** statement is a code or program that handles exceptions.
- The **try** clause runs the code that generates exceptions.
- The **catch** clause catches exceptions that are thrown.
- A **finally** clause always gets executed.
- The **throw** statement generates exceptions.

Syntax:

```
try
{
    // code that may throw an error
}
catch(ex)
{
    // code to be executed if an error occurs
}
finally{
    // code to be executed regardless of an error occurs or not
}
```

Example:

```
<script>
    const numerator= 100, denominator = 'a';

    try {
        document.write(numerator/denominator);
        document.write(a+"<br>");
    }
    catch(error) {
        document.write('An error caught'+"<br>");
        document.write('Error message: ' + error+"<br>");
    }
    finally {
        document.write('Finally will execute every time'+"<br>");
    }
}
```

</script>

Output:

NaN

An error caught

Error message: ReferenceError: 'a' is undefined

Finally will execute every time

Example 2:

```
<body>
<h2>JavaScript try catch</h2>
<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() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>
</body>
```

JavaScript try catch

Please input a number between 5 and 10:

Test Input

Input is too high

EVENT HANDLING:

JavaScript Events:

The change in the state of an object is known as an Event.

- In html, there are various events which represents that some activity is performed by the user or by the browser.
- process of reacting over the events is called Event Handling.
- JavaScript handles the HTML events via Event Handlers.

For example, when a user clicks over the browser

- Some of the HTML events and their event handlers are:

Mouse events

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element

mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Keyboard events

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Form events

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form

blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

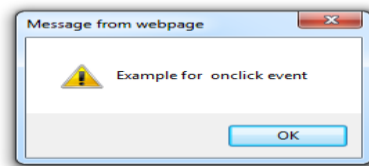
Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Example For OnClick Event:

```
<body> <form>
<input type="button" onclick="clিকেভেন্ট()" value="click"/>
</form>
<script language="Javascript" type="text/Javascript">
function clিকেভেন্ট()
{
    alert("Example for onclick event" );
}
</script>
</body>
```

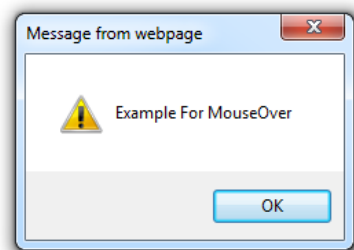
click



Example For mouseOver Event:

```
<body>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
<script language="Javascript" type="text/Javascript">
function mouseoverevent()
{
    alert("Example For MouseOver");
}
</script>
</body>
```

Keep cursor over me



Example For focus Event:

```
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script language="Javascript" type="text/Javascript">
function focusevent()
{
    document.getElementById("input1").style.background=" yellow";
}
</script>
</body>
```

Enter something here



Definition jQuery :

jQuery is a fast, small, and feature-rich JavaScript library.

jQuery is a JavaScript library created to simplify manipulation, HTML DOM tree traversal, and event handling, CSS animation, and Ajax.

jQuery makes a web developer's life easy. It provides many built-in functions using which we can accomplish various tasks easily and quickly.

jQuery Important Features

- **DOM Selection:**

jQuery provides Selectors to retrieve DOM element based on different criteria like tag name, id, css class name, attribute name, value, nth child in hierarchy etc.

- **DOM Manipulation:**

DOM elements can be manipulated using various built-in jQuery functions. For example, adding or removing elements, modifying html content, css class etc.

- **Special Effects:**

special effects can be applied to DOM elements like show or hide elements, fade-in or fade-out of visibility, sliding effect, animation etc.

- **Events:**

jQuery library includes functions which are equivalent to DOM events like click, dblclick, mouseenter, mouseleave, blur, keyup, keydown etc. These functions automatically handle cross-browser issues.

- **Ajax:**

jQuery also includes easy to use AJAX functions to load data from servers without reloading whole page.

- **Cross-browser support:**

jQuery library automatically handles cross-browser issues, so the user does not have to worry about it. jQuery supports IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+.

jQuery Library:

There are two ways to start using jquery library. Those are:

- 1) Download jQuery library directly from the official website : jquery.com
- 2) Include jQuery Library from CDN.

Example :

Java script Program to fade in and out an image on button click using jquery.

```
<head>

<script>

    function func1()

    {

        $("#img1").fadeToggle();

    }

</script>

</head>

<body>

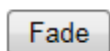
    <button onclick="func1()">Fade</button>

    <script src="https://code.jquery.com/jquery-3.6.1.js" integrity="sha256-3zlB5s2uwoUzrXK3BT7AX3FyvojsraNFx_Cc2vC/7pNI=" crossorigin="anonymous">

</script>

</body>
```

Output:



click the button Fade to get the image



Again click the button Fade to close the image



Validation – jQuery:

Using a library to do form validations can save lots of development time. jQuery Form validation library is the most popular validation library.

How JQuery Validate work?

- The jQuery Validate is a feature provided by jQuery that makes it easier for the creator to validate the user or visitor's input data while keeping code free of all the mesh of validation rules from JavaScript code.
- It consists of validation functions or in simple words, rules set into groups called "modules", which makes it possible to fetch them and consider only those rules or functions, which are needed to check or validate a particular field of the entire form.

Procedure to apply JQuery Validation

Step 1: Include the required jQuery Files.

The first step is to add the required jQuery files so that we are able to use jQuery Validate().

There are two ways, either you can add directly from a source (like CDN) or download it in local machine from a source (jquery.com or Bower or npm) and then add to your code.

Step 2: Create the HTML Form.

The second step is to create HTML form.

Step 3: Create styling for form and form fields.

The third step is completely optional but it will make it more interactive for a user.

Step 4: Call the jQuery Validate()

The fourth step is where we will select html form and call the jQuery Validate().

Example:

```
<head>
```

```
<title>jQuery Validate Demo</title>
```

```
<!-- SRC attribute can be changed according to your preference or location of JS file -->
```

```
<script src="https://cdn.jsdelivr.net/jquery/1.12.4/jquery.min.js"></script>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
```

```
validate/1.19.1/jquery.validate.min.js"></script>
```

```
<style type="text/css">
```

```
/* Styles */
```

```
* {
```

```
box-sizing: border-box;
```

```
}
```

```
label {
```

```
padding: 11px 12px 11px 2px;
```

```
display: inline-block;
```

```
}
```

```
input[type=text], input[type=email], input[type=password], textarea
```

```
{
```

```
border: 2px solid #ccc;
```

```
resize: vertical
```

```
padding: 12px;
```

```
border-radius: 4px;
```

```
width: 100%;
```

```
}
```

```
.container
```

```
{
```

```
border-radius: 5px;
```

```
background-color: #f2f2f2;
```

```
padding: 20px;
```

```
}
```

```
form
```

```
{
```

```
padding: 20px;
```

```
background: #2c3e50;
```

```
color: #fff;
```

```
border-radius: 4px;
```

```
webkit-border-radius: 4px;
```

```
border-radius: 4px;
```

```
}
```

```
/* Set a style for the submit button */
```

```
.registerbtn {
```

```
background-color: #4CAF50;
```

```
color: white;
```

```
padding: 12px 20px;
```

```
margin: 10px 0;
```

```
border: none;
```

```
border-radius: 4px;
```

```
cursor: pointer;
```

```
float: right;
```

```
}
```

```
@media screen and (max-width: 560px) {
```

```
.row {
```

```
margin-top: 0;
```

```
width: 100%;
```

```
}
```

```
}
```

```
form .error {
```

```
color: #ff0001;
```

```
}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<div class="container">
```

```
<form class="form" id="MyForm" method="get" action="" name="MyForm">
```

```
<fieldset>
```

```
<legend>Registration Form</legend>
```



```
<div class="row">
```

```
<label for="FName">First Name</label>
```

```
<input id="FName" name="FName" type="text" minlength="2">
```

```
</div>
```

```
<div class="row">
```

```
<label for="LName">Last Name</label>
```

```
<input id="LName" name="LName" type="text">
```

```
</div>
```

```
<div class="row">
```

```
<label for="Email">E-Mail</label>
```

```
<input id="Email" type="email" name="Email">
```

```
</div>
```

```
<div class="row">
```

```
<label for="Pwd">Password</label>
```

```
<input type="password" name="Pwd" id="Pwd" >
```

```
</div>
```

```
<div class="row">
```

```
<label for="Cmt">Your comment</label>
```

```
<textarea id="Cmt" name="Cmt"></textarea>
```

```
</div>
```

```
<div class="row">
```

```
<input class="registerbtn" type="submit" value="Register">
```

```
</div>
```

```
</fieldset>
```

```
</form>
```

```
</div>
```

```
<script>
```

```
$(document).ready(function() {
```

```
$("#MyForm").validate({
```

```
rules: {
```

'FName': {

required: true,

minlength: 2

},

'LName': {

required: true,

minlength: 2

},

'Email':{

required: true,

email:true,

},

'Pwd':{

required: true,

},

```
},
```

```
messages: {
```

```
'FName': "Please enter a valid First Name.",
```

```
'LName': "Please enter a valid Last Name.",
```

```
'Email': "Please enter Email in proper format.",
```

```
'Pwd': "Please enter a Password",
```

```
}
```

```
});
```

```
});
```

```
</script>
```

```
</body>
```

```
</html>
```

Registration Form

First Name

Last Name

E-Mail

Password

Your comment

Register

When we click on the 'Register' button, it shows the validation for all the errors the user made.

Registration Form

First Name

Please enter a valid First Name

Last Name

Please enter a valid Last Name

E-Mail

Please enter Email in proper format

Password

Please enter a Password

Your comment

Register