HTML

1) <!DOCTYPE>

```
<html>
<head>
<title>Web page title</title>
</head>
<body>
<h1>Write Your First Heading</h1>
<p>Write Your First Paragraph.</p>
</body>
```

Write Your First Heading

Write Your First Paragraph.

2) <!DOCTYPE html>

```
<html>
<head>
</head>
<body>
   <h1> This is Style attribute</h1>
  <p style="height: 50px; color: blue">It will add style property in element</p>
   <p style="color: red">It will change the color of content</p>
</body>
</html>
```

This is Style attribute

It will add style property in element

It will change the color of content

3. <!DOCTYPE html>

```
<html>
 <head>
</head>
<body>
 <h1>Example of src attribute</h1>
```

HTML images can be diplayed with the help of image tag and its attribute src gives the sourc for that image

<img src="D://whitepeacock.jpg" height="400" width="600">

</body>

</html>

```
1.  <!DOCTYPE html>
2.  <html>
3.          <head>
4.     </head>
5.  <body>
6.     <div style="background-color: lightblue">This is first div</div>
7.     <div style="background-color: lightgreen">This is second div</div>
8.     <p style="background-color: pink">This is a block level element</p>
9.  </body>
10. </html>
```

Output

This is first div

This is second div

This is a block level element

JavaScript is *an object-based scripting language*. JavaScript is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
4. It is a light-weighted and interpreted language.
5. It is a case-sensitive language.
6. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
7. It provides good control to the users over the web browsers.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

```
<html>
<body>
<h2>Welcome to JavaScript</h2>
<script>
document.write("Hello JavaScript by JavaScript");
</script>
</body>
</html>
```

output

Welcome to JavaScript

Hello JavaScript by JavaScript.

The **document.write()** function is used to display dynamic content through JavaScript

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

3 Places to put JavaScript code

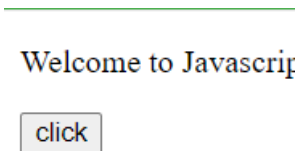
1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

```
<html>

<body>

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

```
 alert("Hello Javatpoint");
</script>
</body>
</html>
```

JavaScript Example : code between the head tag

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to Javascript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```



External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

```
1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>
```

```
function msg(){
 alert("Hello Javatpoint");
}
```

JavaScript Comment

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

JavaScript Single line Comment

```
<html>
<body>
<script>
// It is single line comment
document.write("hello javascript");  </script>  </body> </html>
```

```
<html>
<body>
<script>
var a=10;
var b=20;
var c=a+b;//It adds values of a and b variable
document.write(c);//prints sum of 10 and 20
</script>
</body> </html>
```

JavaScript Multi line Comment

```
<html>
<body>
<script>
/* It is multi line comment.
It will not be displayed */
document.write("example of javascript multiline comment");
</script>
</body>
</html>
```

Output

example of javascript multiline comment

JavaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

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.

```
<html>

<body>

<script>

var x = 10;

var y = 20;

var z=x+y;

document.write(z);

</script>

</body>

</html>
```

JavaScript local variable

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

```
1. <script>
2. function abc(){
3. var x=10;//local variable
4. }
5. </script>
```

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:

```
<html>

<body>

<script>

var data=200;//gloabal variable

function a(){

document.writeln(data);
```

```
}
function b(){
document.writeln(data);
}
a();//calling JavaScript function
b();
 </script>
</body>
</html>
```

Output

200 200

```
<script>
var value=50;//global variable
function a(){
alert(value);
}
}
</script>
```

To declare JavaScript global variables inside function, you need to use **window object**.

```
window. value=90;
```

```
<html>
<body>
<script>
function m(){
window.value=100;//declaring global variable by window object
}
```

```
function n(){
alert(window.value);//accessing global variable from other function
}  m(); n();
</script>
</body>
</html>
```

Before (2015), JavaScript did not have **Block Scope**.

JavaScript had **Global Scope** and **Function Scope**.

ES6 introduced the two new JavaScript keywords: `let` and `const`.

These two keywords provided Block Scope in JavaScript:

Example

Variables declared inside a { } block cannot be accessed from outside the block:

```
{
  let x = 2;
}
// x can NOT be used here
```

Global Scope

Variables declared with the `var` always have **Global Scope**.

Variables declared with the `var` keyword can NOT have block scope:

Example

Variables declared with `var` inside a { } block can be accessed from outside the block:

```
{
  var x = 2;
}
// x CAN be used here
```

Cannot be Redeclared

Variables defined with `let` **can not** be redeclared.

You can not accidentally redeclare a variable declared with `let`.

With `let` you **can not** do this:

```
let x = "John Doe";

let x = 0;
```

Variables defined with `var` **can** be redeclared.

With `var` you **can** do this:

```
var x = "John Doe";

var x = 0;
```

Redeclaring Variables

Redeclaring a variable using the `var` keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

> Redeclaring a variable using the `let` keyword can solve this problem.
>
> Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

Difference Between var, let and const

	Scope	Redeclare	Reassign	Hoisted	Binds

var	No	Yes	Yes	Yes	Yes
let	Yes	No	Yes	No	No
const	Yes	No	No	No	No

`let` and `const` have **block scope**.

`let` and `const` can not be **redeclared**.

`let` and `const` must be **declared** before use.

`let` and `const` does **not bind** to `this`.

`let` and `const` are **not hoisted**.

`var` does not have to be declared.

`var` is hoisted.

`var` binds to this.

Redeclaring

Redeclaring a JavaScript variable with `var` is allowed anywhere in a program:

Example

```
var x = 2;
// Now x is 2

var x = 3;
// Now x is 3
```

With `let`, redeclaring a variable in the same block is NOT allowed:

Example

```
var x = 2;   // Allowed
let x = 3;   // Not allowed

{
let x = 2;   // Allowed
let x = 3;   // Not allowed
}

{
let x = 2;   // Allowed
var x = 3;   // Not allowed
}
```

Redeclaring a variable with `let`, in another block, IS allowed:

Example

```
let x = 2;   // Allowed

{
let x = 3;   // Allowed
}

{
let x = 4;    // Allowed
}
```

Let Hoisting

Variables defined with `var` are hoisted to the top and can be initialized at any time.

Meaning: You can use the variable before it is declared:

Example

This is OK:

```
carName = "Volvo";
var carName;
```

Variables defined with `let` are also hoisted to the top of the block, but not initialized.

Meaning: Using a `let` variable before it is declared will result in a `ReferenceError`:

Example

```
carName = "Saab";
let carName = "Volvo";
```

ReferenceError: Cannot access 'carName' before initialization

Javascript Data Types

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

1. Primitive data type
2. Non-primitive (reference) data type

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. You need to use **var** here to specify the data type. It can hold any type of values such as numbers, strings etc.

1. var a=40;//holding number
2. var b="Rahul";//holding string

JavaScript primitive data types

There are five 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
Boolean	represents boolean value either false or true
Undefined	represents undefined value

Null	represents null i.e. no value at all

JavaScript non-primitive data types

Object

Array

RegExp

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers –

- **Encapsulation** – the capability to store related information, whether data or methods, together in an object.
- **Aggregation** – the capability to store one object inside another object.
- **Inheritance** – the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** – the capability to write one function or method that works in a variety of different ways.
- Object Properties
- Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.
- The syntax for adding a property to an object is –
- objectName.objectProperty = propertyValue;
- **For example** – The following code gets the document title using the **"title"** property of the **document** object.
- var str = document.title;

Object Methods

There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example – Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

```
document.write("This is test");
```

A javaScript object is an entity having state and behavior (properties and method). For example: car, pen, bike, chair, glass, keyboard, monitor etc.

JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

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

1) JavaScript Object by object literal

The syntax of creating object using object literal is given below:

```
object={property1:value1, property2:value2.....propertyN: valueN}
```

As you can see, property and value is separated by : (colon).

Let's see the simple example of creating object in JavaScript.

```
<script>
emp={id:102,name:"Shyam Kumar",salary:40000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

2)By creating instance of Object

The syntax of creating object directly is given below:

```
var objectname=new Object();
```

Here, **new keyword** is used to create object.

Let's see the example of creating object directly.

```
<script>
var emp=new Object();
emp.id=101;
emp.name="Ravi Malik";
emp.salary=50000;
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
```

3) By using an Object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.

The **this keyword** refers to the current object.

The example of creating object by object constructor is given below.

```
<script>
function emp(id,name,salary){
this.id=id;
this.name=name;
this.salary=salary;
}
e=new emp(103,"Vimal Jaiswal",30000);

document.write(e.id+" "+e.name+" "+e.salary);
</script>
```

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.

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

The syntax of creating array using array literal is given below:

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

As you can see, values are contained inside [ ] and separated by , (comma).

Let's see the simple example of creating and using array in JavaScript.

```
<html>
<body>
<script>
var emp=["Sonoo","Vimal","Ratan"];
for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br/>");
}
</script>
</body>
</html>
```

Sonoo
Vimal
Ratan

3) JavaScript Array directly (new keyword)

The syntax of creating array directly is given below:

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

Here, **new keyword** is used to create instance of array.

Let's see the example of creating array directly.

```
<html>
<body><script>
var i;
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";

for (i=0;i<emp.length;i++){
```

```
10. document.write(emp[i] + "<br>");
11. }
12. </script>  </body></html>
```

Arun
Varun
John

3) JavaScript array constructor (new keyword)

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

The example of creating object by array constructor is given below.

```
1. <script>
2. var emp=new Array("Jai","Vijay","Smith");
3. for (i=0;i<emp.length;i++){
4. document.write(emp[i] + "<br>");
5. }
6. </script>
```

```
Jai
Vijay
Smith
```

JavaScript String

The **JavaScript string** is an object that represents a sequence of characters.

There are 2 ways to create string in JavaScript

1. By string literal
2. By string object (using new keyword)

---

1) By string literal

The string literal is created using double quotes. The syntax of creating string using string literal is given below:

```
1. var stringname="string value";
```

Let's see the simple example of creating string literal.

```
1. <script>
2. var str="This is string literal";
3. document.write(str);
```

```
4. </script>
```

```
This is string literal
```

2) By string object (using new keyword)

The syntax of creating string object using new keyword is given below:

```
1. var stringname=new String("string literal");
```

Here, **new keyword** is used to create instance of string.

Let's see the example of creating string in JavaScript by new keyword.

```
1. <script>
2. var stringname=new String("hello javascript string");
3. document.write(stringname);
4. </script>
```

```
hello javascript string
```

1) JavaScript String charAt(index) Method

The JavaScript String charAt() method returns the character at the given index.

```
1. <script>
2. var str="javascript";
3. document.write(str.charAt(2));
4. </script>
```

v

2)JavaScript String concat(str) Method

The JavaScript String concat(str) method concatenates or joins two strings.

```
1. <script>
2. var s1="javascript ";
3. var s2="concat example";
4. var s3=s1.concat(s2);
5. document.write(s3);
6. </script>
```

```
javascript concat example
```

3)JavaScript String indexOf(str) Method

The JavaScript String indexOf(str) method returns the index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.indexOf("from");
document.write(n);
</script>
```

11

4) JavaScript String lastIndexOf(str) Method

The JavaScript String lastIndexOf(str) method returns the last index position of the given string.

```
<script>
var s1="javascript from javatpoint indexof";
var n=s1.lastIndexOf("java");
document.write(n);
</script>
```

16

5) JavaScript String toLowerCase() Method

The JavaScript String toLowerCase() method returns the given string in lowercase letters.

```
<script>
var s1="JavaScript toLowerCase Example";
var s2=s1.toLowerCase();
document.write(s2);
</script>
```

javascript tolowercase example

6) JavaScript String toUpperCase() Method

The JavaScript String toUpperCase() method returns the given string in uppercase letters.

```
<script>
var s1="JavaScript toUpperCase Example";
var s2=s1.toUpperCase();
document.write(s2);
</script>
```

```
JAVASCRIPT TOUPPERCASE EXAMPLE
```

7) JavaScript String slice(beginIndex, endIndex) Method

The JavaScript String slice(beginIndex, endIndex) method returns the parts of string from given beginIndex to endIndex. In slice() method, beginIndex is inclusive and endIndex is exclusive.

```
<script>
var s1="abcdefgh";
var s2=s1.slice(2,5);
document.write(s2);
</script>
```

```
cde
```

8)JavaScript String trim() Method

The JavaScript String trim() method removes leading and trailing whitespaces from the string.

```
<script>
var s1="    javascript trim    ";
var s2=s1.trim();
document.write(s2);
</script>
```

```
javascript trim
```

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

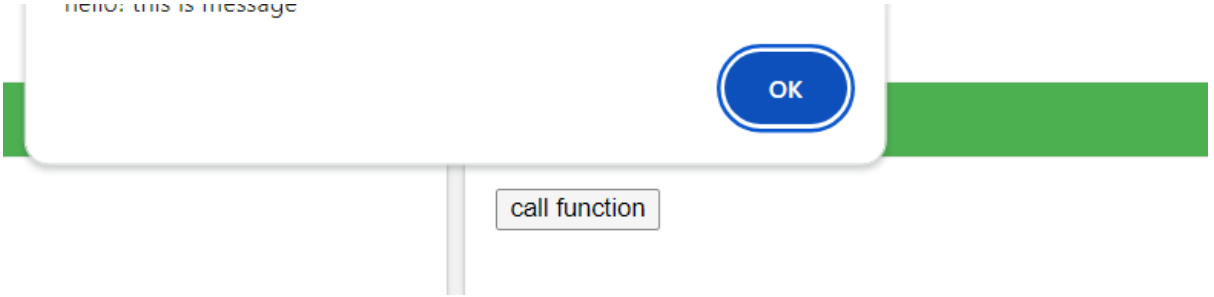
Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability**: We can call a function several times so it save coding.
2. **Less coding**: It makes our program compact. We don't need to write many lines of code each time to perform a common task

JavaScript Function Example

```
<html>
<body>
<script>
function msg(){
alert("hello! this is message");
}
</script>
<input type="button" onclick="msg()" value="call function"/>
</body> </html>
```

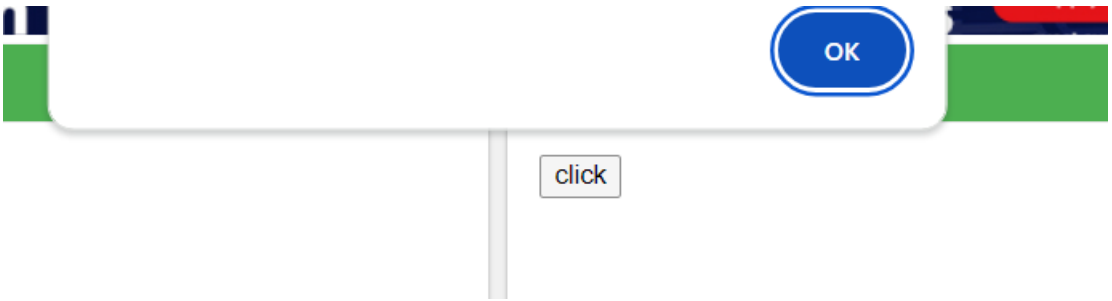


JavaScript Function Arguments

We can call function by passing arguments. Let's see the example of function that has one argument.

```
1. <script>
2. function getcube(number){
3. alert(number*number*number);
4. }
5. </script>
6. <form>
7. <input type="button" value="click" onclick="getcube(4)"/>
```

8. </form>



Function with Return Value

We can call function that returns a value and use it in our program. Let's see the example of function that returns value.

```
1. <script>
2. function getInfo(){
3. return "hello javatpoint! How r u?";
4. }
5. </script>
6. <script>
7. document.write(getInfo());
8. </script>
```

hello javatpoint! How r u?

JavaScript Function Object

In JavaScript, the purpose of **Function constructor** is to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

Syntax

```
1. new Function ([arg1[, arg2[, ....argn]],] functionBody)
```

Example 1

Let's see an example to display the sum of given numbers.

```
1. <script>
2. var add=new Function("num1","num2","return num1+num2");
3. document.writeln(add(2,5));
4. </script>
```

Example 2

```
<script>
var pow=new Function("num1","num2","return Math.pow(num1,num2)");
document.writeln(pow(2,3));
</script>
```

8

User-Defined Objects

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1

Try the following example; it demonstrates how to create an Object.

```
<html>
   <head>
      <title>User-defined objects</title>
      <script type = "text/javascript">
```

```
         var book = new Object();    // Create the
object
         book.subject = "Perl";      // Assign
properties to the object
         book.author  = "Mohtashim";
      </script>
   </head>

   <body>
      <script type = "text/javascript">
         document.write("Book name is : " +
book.subject + "<br>");
         document.write("Book author is : " +
book.author + "<br>");
      </script>
   </body>
</html>
```

Output

Book name is : Perl
Book author is : Mohtashim

JavaScript Operators

There are following types of operators in JavaScript.

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Special Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30

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

JavaScript Comparison Operators

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

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false

||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators.

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

JavaScript Special Operators

The following operators are known as JavaScript 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 Loops

The **JavaScript loops** are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

---

1) JavaScript For loop

The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

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

Let's see the simple example of for loop in javascript.

```
<html>

<body>

<script>

for (i=1; i<=5; i++)

{

document.write(i + "<br/>")

}

</script>

</body>

</html>
```

JavaScript while loop

The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

```
1. while (condition)
2. {
3.     code to be executed
4. }
```

```
<html>

<body>

<script>

var i=11;

while (i<=15)

{

document.write(i + "<br/>");

i++;

}

</script>

</body>

</html>
```

11
12
13
14
15

3) JavaScript do while loop

The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

```
1. do{
2.     code to be executed
3. }while (condition);
```

```
<!DOCTYPE html>

<html>

<body>

<script>

var i=21;

do{

document.write(i + "<br/>");

i++;

}while (i<=25);

</script>

</body>

</html>
```

```
21
22
23
24
25
```

4) JavaScript for in loop

The **JavaScript for in loop** is used *to iterate the properties of an object*. We will discuss about it later.

```
<!DOCTYPE html>

<html>
```

```
<body>
<h1>JavaScript Statements</h1>
<h2>The for...in Loop</h2>
<script>
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
for (let x in person) {
  text += person[x] + " ";
}
document.write(text);
</script>
</body>
</html>
```

JavaScript Statements

The for...in Loop

John Doe 25

UNIT II

Regular expression

- Regular expression is a sequence of pattern that defines a string. It is used to denote regular languages.
- It is also used to match character combinations in strings. String searching algorithm used this pattern to find the operations on string.
- In regular expression, x* means zero or more occurrence of x. It can generate {e, x, xx, xxx, xxxx,.....}
- In regular expression, x+ means one or more occurrence of x. It can generate {x, xx, xxx, xxxx,.....}

Operations on Regular Language

The various operations on regular language are:

Union: If L and M are two regular languages then their union L U M is also a union.

1. L U M = {s | s is in L or s is in M}

Intersection: If L and M are two regular languages then their intersection is also an intersection.

1. L ⋂ M = {st | s is in L and t is in M}

Kleene closure: If L is a regular language then its kleene closure L1* will also be a regular language.

L* = Zero or more occurrence of language L.

Example

Write the regular expression for the language:

$L = \{ab^n w : n \geq 3, w \in (a,b)^+\}$

Solution:

$L = \{ab^3 a, ab^4 a, ab^5 b\}$

The string of language L starts with "a" followed by atleast three b's. It contains atleast one "a" or one "b" that is string are like abbba, abbbbbba, abbbbbbbb, abbbb.....a

So regular expression is:

$r = ab^3 b^* (a+b)^+$

Regex Tutorial

The term Regex stands for Regular expression. The regex or regexp or regular expression is a sequence of different characters which describe the particular search pattern. It is also referred/called as a Rational expression.

It is mainly used for searching and manipulating text strings. In simple words, you can easily search the pattern and replace them with the matching pattern with the help of regular expression.

Regular Expression Characters

There are following different type of characters of a regular expression:

1. Metacharacters
2. Quantifier

3. Groups and Ranges
4. Escape Characters or character classes

Metacharacters

Meta characters	Description	Example
^	This character is used to match an expression to its right at the start of a string.	**^a** is an expression match to the string which starts with '**a**' such as "**aab", "a9c", "apr", "aaaaab**", etc.
$	The $sign is used to match an expression to its left at the end of a string.	**r$** is an expression match to a string which ends with r such as "**aaabr", "ar", "r", "aannn9r**", etc.
.	This character is used to match any single character in a string except the line terminator, i.e. /n.	**b.x** is an expression that match strings such as **"bax", "b9x", "bar"**.
|	It is used to match a particular character or a group of characters on either side. If the character on the left side is matched, then the right side's character is ignored.	**A|b** is an expression which gives various strings, but each string contains either **a** or **b**.
****	It is used to escape a special character after this sign in a string.	
A	It is used to match the character 'A' in the string.	This expression matches those strings in which at least one-time **A** is present. Such strings are "**Amcx", "mnAr", "mnopAx4"**.
Ab	It is used to match the substring 'ab' in the string.	This expression matches those strings in which **'Ab'** is present at least one time. Such strings are "**Abcx", "mnAb", "mnopAbx4"**.

Quantifiers

The quantifiers are used in the regular expression for specifying the number of occurrences of a character

Characters	Description	Example

+	This character specifies an expression to its left for one or more times.	**s+** is an expression which gives **"s", "ss",** "**sss**", and so on.
?	This character specifies an expression to its left for 0 (Zero) or 1 (one)times.	**aS?** is an expression which gives either "**a**" or "**as", but not "ass".**
*****	This character specifies an expression to its left for 0 or more times	**Br*** is an expression which gives **"B", "Br", "Brr", "Brrr"**, and so on...
{x}	It specifies an expression to its left for only x times.	**Mab{5}** is an expression which gives the following string which contains 5 b's: **"Mabbbbb"**
{x, }	It specifies an expression to its left for x or more times.	**Xb{3, }** is an expression which gives various strings containing at least 3 b's. Such strings are "**Xbbb", "Xbbbb"**, and so on.
{x,y}	It specifies an expression to its left, at least x times but less than y times.	**Pr{3,6}a** is an expression which provides two strings. Both strings are as follows: **"Prrrr" and "Prrrrr"**

Groups and Ranges

The groups and ranges in the regular expression define the collection of characters enclosed in the brackets.

Characters	**Description**	**Example**
( )	It is used to match everything which is in the simple bracket.	**A(xy)** is an expression which matches with the following string: **"Axy"**
{ }	It is used to match a particular number of occurrences defined in the curly bracket for its left string.	**xz{4,6}** is an expression which matches with the following string: **"xzzzzz"**
[ ]	It is used to match any character from a range of	**xz[atp]r** is an expression which matches with the following strings: **"xzar", "xztr", and "xzpr"**

	characters defined in the square bracket.	
[pqr]	It matches p, q, or r individually.	Following strings are matched with this expression: **"p", "q", and "r".**
[pqr][xy]	It matches p, q, or r, followed by either x or y.	Following strings are matched with this expression: **"px", "qx", and "rx", "py", "qy", and "ry".**
(?: ...)	It is used for matching a non-capturing group.	A(?:nt|pple) is an expression which matches to the following string: **"Apple"**
[^.....]	It matches a character which is not defined in the square bracket.	Suppose, **Ab[^pqr]** is an expression which matches only the following string: **"Ab"**
[a-z]	It matches letters of a small case from a to z.	This expression matches the strings such as: **"a", "python", "good".**
[A-Z]	It matches letters of an upper case from A to Z.	This expression matches the strings such as: **"EXCELLENT", "NATURE".**
^[a-zA-Z]	It is used to match the string, which is either starts with a small case or upper-case letter.	This expression matches the strings such as: **"A854xb", "pv4fv", "cdux".**
[0-9]	It matches a digit from 0 to 9.	This expression matches the strings such as: **"9845", "54455"**
[aeiou]	This square bracket only matches the small case vowels.	-
[AEIOU]	This square bracket only matches the upper-case vowels.	-
ab[^4-9]	It matches those digits or characters which are not defined in the square bracket.	This expression matches those strings which do not contain 5, 6, 7, and 8.

Escape Characters or Character Classes

Characters	Description
\s	It is used to match a one white space character.
\S	It is used to match one non-white space character.
\0	It is used to match a NULL character.
\a	It is used to match a bell or alarm.
\d	It is used to match one decimal digit, which means from 0 to 9.
\D	It is used to match any non-decimal digit.
\n It helps a user to match a new line.	
\w	It is used to match the alphanumeric [0-9a-zA-Z] characters.
\W	It is used to match one non-word character
\b	It is used to match a word boundary.

Use of Regular Expression in JavaScript

You can easily use the regular expression in the JavaScript code by the help of following two string methods:

1. **search():** This method searches the regular expression in the string and also returns the position where the match found.
2. **Replace():** This method is used to return the string after the replacement of a matched character in a string.

Example 1: This example uses the **search()** method in the JavaScript script for understanding the regular expression.

```
1. <!DOCTYPE html>
```

```
<html>
<head>
<script>
var string = "Our Site is helpfull for studying about technical courses.!";
pattern="technical";
var res = string.search(pattern); /* This statement stores the position of the patte
rn in a string, if it is found in a string. */
document.write("Position of the pattern in a string:");
document.write(res);
</script>
</head>
<body>
</body>
</html>
```

```
Position of the pattern in a string:40
```

Example 2: This example uses the **replace()** method in the JavaScript script for understanding the regular expression.

```
<html>
<head>
<script>
var string = "You are a Bad Student";
var pattern=/Bad/;
var replace1="Good";
var res = string.replace(/Bad/,replace1);
/* The above statement replaces the Bad word from the string by the Good wor
d using the replace method. */
document.write("After replacing the substring, the modified string is:"+ '<br>');
document.write(res);
</script>
</head>
<body>
</body>
</html>
```

```
After replacing the substring, the modified
string is:

You are a Good Student
```