



Unit 1 Basics of JavaScript Programming

Marks: 12 (R-4, U-4, A-4)

Course Outcome: Create interactive web pages using program flow control structure.

Unit Outcome:

- 1) Create object to solve the given problem.
- 2) Develop javascript to implement the switch-case statement for the given problem.
- 3) Develop javascript to implement loop for solving the given iterative problem.
- 4) Display properties of the given object using getters and setters.
- 5) Develop program using basic features of javascript to solve the given problem.

Topics and Sub-topics:

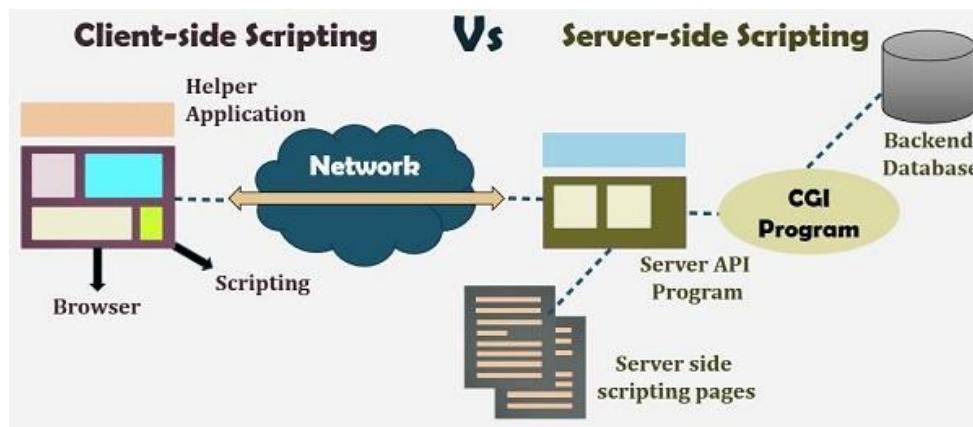
- 1.1 Features of JavaScript
- 1.2 Object Name, Property, Method, Dot Syntax, Main Event
- 1.3 Values and Variables
- 1.4 Operators and Expressions
- 1.5 if statement, if...else. If...elseif, Nested if
- 1.6 switch... case statement
- 1.7 Loop statement
- 1.8 Querying and setting properties and Deleting properties,
Property Getters and Setters



Introduction:

The scripts can be written in two forms, at the server end (back end) or at the client end (server end). The main difference between server-side scripting and client-side scripting is that the server-side scripting involves server for its processing. On the other hand, client-side scripting requires browsers to run the scripts on the client machine but does not interact with the server while processing the client-side scripts.

A script is generally a series of program or instruction, which has to be executed on other program or application. As we know that the web works in a client-server environment. The client-side script executes the code to the client side which is visible to the users while a server-side script is executed in the server end which users cannot see.



Source: <https://techdifferences.com/difference-between-server-side-scripting-and-client-side-scripting.html>

Server-side scripting is a technique of programming for producing the code which can run software on the server side, in simple words any scripting or programming that can run on the web server is known as server-side scripting.

The operations like customization of a website, dynamic change in the website content, response generation to the user's queries, accessing the database, and so on are performed at the server end.

The server-side scripting constructs a communication link between a server and a client (user). Earlier the server-side scripting was implemented by the **CGI (Common Gateway Interface)** scripts. CGI was devised to execute the scripts from programming languages such as C++ or Perl on websites.

The server-side involves four parts: server, database, API's and back-end web software developed by the server-side scripting language. When a browser sends a request to the server for a webpage consisting of server-side scripting, the web server processes the script prior to serving the page to the browser. Here the processing of a script could include extracting information from a database, making simple calculations, or choosing the appropriate content that is to be displayed in the client end. The script is being processed



and the output is sent to the browser. The web server abstracts the scripts from the end user until serving the content, which makes the data and source code more secure. The Programming languages for server-side programming are:

- 1) PHP
- 2) C++
- 3) Java and JSP
- 4) Python
- 5) Ruby

Client-side scripting is performed to generate a code that can run on the client end (browser) without needing the server-side processing.

Basically, these types of scripts are placed inside an HTML document. The client-side scripting can be used to examine the user's form for the errors before submitting it and for changing the content according to the user input.

In Client-side scripting, the web requires three elements for its functioning which are, client, database and server.

The effective client-side scripting can significantly reduce the **server load**. It is designed to run as a scripting language utilizing a web browser as a host program. For example, when a user makes a request via browser for a webpage to the server, it just sent the HTML and CSS as plain text, and the browser interprets and renders the web content in the client end.

The Programming languages for client-side programming are:

- 1) Javascript
- 2) VBScript
- 3) HTML
- 4) CSS (Cascading Style Sheet)
- 5) AJAX

Comparison: Server-side vs. Client-side scripting

| BASIS FOR COMPARISON | SERVER-SIDE SCRIPTING | CLIENT-SIDE SCRIPTING |
|----------------------|---|--|
| Basic | Works in the back end which could not be visible at the client end. | Works at the front end and script are visible among the users. |



| BASIS FOR COMPARISON | SERVER-SIDE SCRIPTING | CLIENT-SIDE SCRIPTING |
|----------------------|---|--|
| Processing | Requires server interaction. | Does not need interaction with the server. |
| Languages involved | PHP, ASP.net, Ruby on Rails, Python, etc. | HTML, CSS, JavaScript, etc. |
| Affect | Could effectively customize the web pages and provide dynamic websites. | Can reduce the load to the server. |
| Security | Relatively secure. | Insecure. |

Why we use Javascript?

Using HTML, we can only design a web page but you 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 for perform all these tasks at client side you need to use JavaScript.

Where JavaScript is Used?

There are too many web applications running on the web that are using JavaScript technology like Gmail, Facebook, twitter, google map, YouTube etc.

Following are some uses of JavaScript:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time
- Validate user input in an HTML form before sending the data to a server.
- Build forms that respond to user input without accessing a server.
- Change the appearance of HTML documents and dynamically write HTML into separate Windows.
- Open and close new windows or frames.



- Manipulate HTML "layers" including hiding, moving, and allowing the user to drag them around a browser window.
- Build small but complete client-side programs.
- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)
- Displaying clocks etc.

1.1 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;

- JavaScript is an object-based scripting language.
- Giving the user more control over the browser.
- It Handling dates and time.
- It Detecting the user's browser and OS,
- It is light weighted.
- JavaScript is a scripting language and it is not java.
- JavaScript is interpreter-based scripting language.
- JavaScript is case sensitive.
- JavaScript is object-based language as it provides predefined objects.
- Every statement in JavaScript must be terminated with semicolon (;).
- Most of the JavaScript control statements syntax is same as syntax of control statements in C language.
- An important part of JavaScript is the ability to create new functions within scripts. Declare a function in JavaScript using **function** keyword.
- The concept of class and OOPs has been more refined. Also, in JavaScript, two important principles with OOP in JavaScript are Object Creation patterns (**Encapsulation**) and Code Reuse patterns (**Inheritance**). Although JavaScript developers rarely use this feature but its there for everyone to explore.
- JavaScript is platform-independent or we can say it is portable; which simply means that you can simply write the script once and run it anywhere and anytime. In general, you can write your JavaScript applications and run them on any platform or any browser without affecting the output of the Script.



Advantages of JavaScript:

- **Speed:** Client-side JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server.
- **Simplicity:** JavaScript is relatively simple to learn and implement.
- **Popularity:** JavaScript is used everywhere on the web.
- **Interoperability:** JavaScript plays nicely with other languages and can be used in a huge variety of applications.
- **Server Load:** Being client-side reduces the demand on the website server.
- **Light-weight and interpreted:** JavaScript is a lightweight scripting language because it is made for data handling at the browser only. Since it is not a general-purpose language so it has a limited set of libraries. Also, as it is only meant for client-side execution and that too for web applications, hence the lightweight nature of JavaScript is a great feature. JavaScript is an interpreted language which means the script written inside JavaScript is processed line by line. These Scripts are interpreted by JavaScript interpreter which is a built-in component of the Web browser. But these days many JavaScript engines in browsers like the V8 engine in chrome uses just in time compilation for JavaScript code.
- Gives the ability to create rich interfaces.
- **Client-side Validations:** This is a feature which is available in JavaScript since forever and is still widely used because every website has a form in which users enter values, and to make sure that users enter the correct value, we must put proper validations in place, both on the client-side and on the server-side. JavaScript is used for implementing client-side validations.

Disadvantages of JavaScript:

- **Security:** As the code executes the user's computer, in some cases it can be exploited for malicious purpose.
- Javascript does not read and write the files.
- Javascript can not be used for networking applications.
- Javascript does not have multi-threading and multi-processing capabilities.



- Javascript does not support overloading and overriding.

1.2 Object Name, Property, Method, Dot Syntax, Main Event:

JavaScript is an Object based scripting language.

A JavaScript object is a collection of named values.

These named values are usually referred to as properties of the object.

A JavaScript objects are collection of properties and methods.

- ✓ A Methods is a function that is a member of an object.
 - ✓ A Property is a value or set of values that is the member of an object.
- In JavaScript, almost "everything" is an object.
- ✓ Booleans can be objects (if defined with the new keyword)
 - ✓ Numbers can be objects (if defined with the new keyword)
 - ✓ Strings can be objects (if defined with the new keyword)
 - ✓ Dates are always objects.
 - ✓ Maths are always objects
 - ✓ Regular expressions are always objects.
 - ✓ Arrays are always objects.
 - ✓ Functions are always objects.
 - ✓ Objects are always objects.

Object Name:

Each object is uniquely identified by a name or ID.

With JavaScript, you can define and create your own objects.

There are different ways to create new objects:

A. Define and create a single object, using an object literal.

- Using an object literal, you both define and create an object in one statement.
- An object literal is a list of names: value pairs (like age:10) inside curly braces {}.
- The following example creates a new JavaScript object with 3 properties:

```
var person = {  
    firstName: "Hhh",  
    lastName: "Bbb",  
    age: 10  
};
```



In above example, person is an object and firstName , lastName and age are three properties.

“Hhh” , “Bbb” and 10 these are values associated with properties.

Code:

```
<html>
<body>
<script>
emp={id:"VP-179",name:"Aaa Bbb",salary:50000}
document.write(emp.id+" "+emp.name+" "+emp.salary);
</script>
</body>
</html>
```

Output:

VP-179 Aaa Bbb 50000

- B. Define and create a single object, with the keyword “new” OR by creating instance of Object

new keyword is used to create object.

Syntax: var objectname=new Object();

Example:

```
var person = new Object();
person.firstName = "Hhh";
person.age = 10;
```

Code:

```
<html>
<body>
<script>
var emp=new Object();
emp.id="VP-179";
emp.name="Aaa ";
```



```
emp.salary=50000;  
document.write(emp.id+" "+emp.name+" "+emp.salary);  
</script>  
</body>  
</html>
```

Output:

VP-179 Aaa 50000

C. Define an object constructor, and then create objects of the constructed type.

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.

Example:

```
function person(firstName, lastName, age)  
{  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.age = age;  
}  
p=new person("aaa","vvv",10);  
document.write(p.firstName+" "+p.lastName+" "+p.age);
```

Code:

```
<html>  
<body>  
<script>  
function emp(id,name,salary)  
{  
this.id=id;  
this.name=name;  
  
this.salary=salary;  
}  
e=new emp("VP-179","Aaa ",50000);  
document.write(e.id+" "+e.name+" "+e.salary);  
</script>
```



```
</body>
</html>
```

Output:

VP-179 Aaa 50000

Types of Objects:

- **Native Objects/ Built-in Objects:** are those objects supplied by JavaScript. Examples of these are Math, Date, String, Number, Array, Image, etc.

1) Math:

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. |

Code:

```
<html>
<head>
<title>JavaScript Math Object Properties</title>
</head>
<body>
<script type="text/javascript">
var value1 = Math.E;
document.write("E Value is :" + value1 + "<br>");
var value3 = Math.LN10;
document.write("LN10 Value is :" + value3 + "<br>");
var value4 = Math.PI;
document.write("PI Value is :" + value4 + "<br>");
</script>
</body>
```



```
</html>
```

Output:

E Value is :2.718281828459045

LN10 Value is :2.302585092994046

PI Value is :3.141592653589793

Math Met.

| Math 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. |

Code:

```
<html>
<body>
<script type="text/javascript">
var value = Math.abs(-20);
document.write("ABS Value : " + value +<br>);
var value = Math.tan(5);
document.write("TAN Value : " + value +<br>);
</script>
</body>
</html>
```

Output:

ABS Value: 20

TAN Value : -3.380515006246586



2) Date

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 = new Date();
```

Date Methods:

| Date 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. |

Code:

```
<html>
<body>
<h2>Date Methods</h2>
<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>
</body>
</html>
```

Output:

Date Methods

Locale String: 7/3/2020, 5:23:19

PM

Hours: 17

Day: 5

Month: 6

FullYear: 2020

Minutes: 23

In above code, getMonth() will returns 6 since months starts from 0 that is

0->January , 1->February

2-> March , 3 ->April

And so on.



3) String

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:

| String properties | Description |
|-------------------|--|
| length | It returns the length of the string. |
| constructor | It returns the reference to the String function that created the object. |

String Methods:

| String 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. |

Code:

```
<html>
<body>
<script type="text/javascript">
var str = "A JavaScript";
document.write("<b>Char At:</b> " + str.charAt(4)+"<br>");
document.write("<b>Char Code At:</b> " + str.charCodeAt(0)+"<br>");
document.write("<b>Index of:</b> " + str.indexOf("p")+"<br>");
document.write("<b>Lower Case:</b> " + str.toLowerCase()+"<br>");
document.write("<b>Upper Case:</b> " + str.toUpperCase()+"<br>");
</script>
</body>
</html>
```

Output:

Char At: v
CharCode At: 65
Index of: 10
Lower Case: a javascript
Upper Case: A JAVASCRIPT

- **Host Objects:** are objects that are supplied to JavaScript by the browser environment. Examples of these are window, document, forms, etc.

Window:

The window object represents a window in browser.

An object of window is created automatically by the browser.

Window is the object of browser; it is not the object of javascript.

Window Methods:

| Window methods | Description |
|----------------|-------------|
| | |

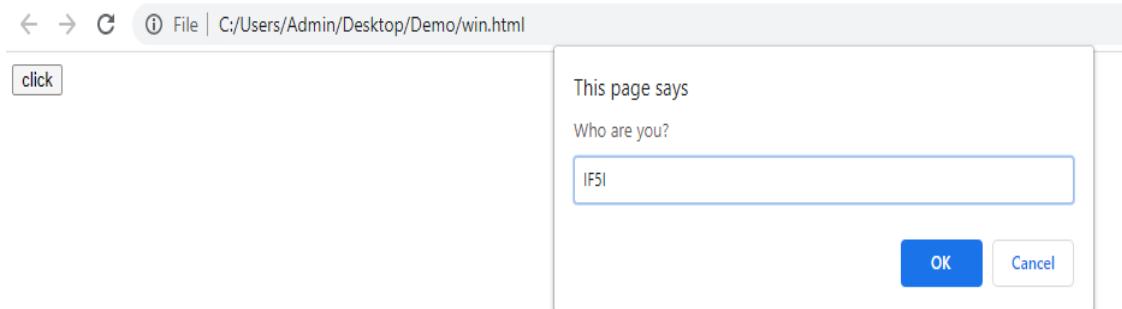
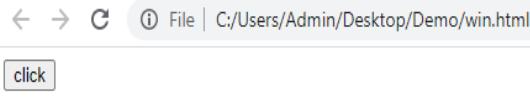


| | |
|-----------|---|
| alert() | displays the alert box containing message with ok button. |
| confirm() | displays the confirm dialog box containing message with ok and cancel button. |
| prompt() | displays a dialog box to get input from the user along with ok and cancel button. |
| open() | opens the new window. |
| close() | closes the current window. |

Code:

```
<script type="text/javascript">
function msg()
{
var a= window.prompt("Who are you?");
window.alert("I am "+a);
}
</script>
<input type="button" value="click" onclick="msg()">
```

Output:





click

This page says

I am IF5I

OK

- **User-Defined Objects:** are those that are defined by you, the programmer.

Property:

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only.
- The syntax for accessing the property of an object is:
objectName.property // person.age
objectName["property"] // person["age"]
objectName[expression] // x = "age"; person[x]

Dot Operator:

The properties and methods associated with any object can be accessed by using dot(.) Operator.

Example, emp.id or op.add();

Dot operator is used to how to interact with objects, methods, events and properties.

Dot operator is also used to add new property.

Example, emp.designation=“Lecturer”;

Code:

```
<html>
<body>
<script>
var person =
{
    firstname:"Hhh",
    age:10
};
person.std = "Fifth"; //adding new property as "std"
document.write(person.firstname+ " is in "+person.std+" standard"); //Accessing
properties with dot
</script>
</body> </html>
```



Output:

Hhh is in Fifth standard

Methods:

JavaScript methods are actions that can be performed on objects.

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is defined with the **function keyword**, followed by a **name**, followed by parentheses () .

The parentheses may include parameter names separated by commas:

(**parameter1**, **parameter2**, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

Syntax:

```
function name(parameter1, parameter2, parameter3) Op7
{
    // code to be executed
}
```

Code: simple method to define a function

```
<html>
<body>
<script>
function op_add(p1, p2)
{
    return p1 + p2;
}
document.write("Addition is="+op_add(4, 5));
</script>
</body>
</html>
```

Output:

Addition is=9

Code: define a function as a property

```
<script>
var person =
```



```
{  
    firstname:"Hhh",  
    lastname:"Bbb",  
    Fullscreen:function() // define a function as a property  
    {  
        return this.firstname+" "+this.lastname;  
    }  
};  
document.write("Person Detail is="+person.Fullname());  
</script>
```

Output:

Person Detail is=Hhh Bbb

Code: define a function as an expression

```
<script>  
var x = function (a, b) // function as an expression  
{  
    return a * b ;  
}  
document.write("function returns= " +x(4, 5));  
</script>
```

Output:

function returns= 20

Main Event:

- An event is an action performed by user or web browser.
- In order to make a web pages more interactive, the script needs to be accessed the contents of the document and know when the user is interacting with it.
- Events may occur due to:
 - 1) a document loading
 - 2) user clicking on mouse button
 - 3) browser screen changing size

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed



- An HTML button was clicked

Event handlers can be used to handle, and verify, user input, user actions, and browser actions:

| Event | Description |
|-------------|--|
| onchange | An HTML element has been changed |
| Onclick | The user clicks an HTML element |
| onmouseover | The user moves the mouse over an HTML element |
| onmouseout | The user moves the mouse away from an HTML element |
| onkeydown | The user pushes a keyboard key |
| onload | The browser has finished loading the page |

Code:

```
<html>
<head>
<script type="text/javascript">
function msg()
{
    alert("Hello IF5I students");
}
</script>
</head>
<body>
<center>
<p><h1>Welcome to Client-side scripting</h1></p>
<form>
<input type="button" value="click" onclick ="msg()"/>    // onclick event
</form>
</body>
</html>
```

Output:



Welcome to Client-side scripting

click

This page says

Hello IF5I students

OK

Welcome to Client-side scripting

click

DOM getElementById() Method

The getElementById() method returns the elements that has given ID which is passed to the function.

This function is widely used in web designing to change the value of any particular element or get a particular element.

Syntax: `document.getElementById(element_id);`

Parameter: This function accepts single parameter `element_id` which is used to hold the ID of element.

Return Value: It returns the object of given ID. If no element exists with given ID then it returns null.

Code:

```
<html>
<body>
<p id="demo">Click the button to change the color of this paragraph.</p>
<button onclick="myFunction()">change color</button>
<script>
function myFunction()
{
    var x = document.getElementById("demo");
    x.style.color = "red";
}
```



```
</script>
</body>
</html>
```

Output:

Click the button to change the color of this paragraph.

Click the button to change the color of this paragraph.

1.3 Values and Variables

- In JavaScript there are two types of scope:
 - Local scope
 - Global scope
- JavaScript has function scope: Each function creates a new scope.
- Scope determines the accessibility (visibility) of these variables.
- Variables defined inside a function are not accessible (visible) from outside the function.

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

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

We can say that variable is a container that can be used to store value and you need to declare a variable before using it.

In JavaScript, the var keyword is used to declare a variable. Also, starting from ES6 we can also declare variables using the let keyword.

JavaScript Syntax for Declaring a Variable

Following is the syntax for declaring a variable and assigning values to it.

```
var variable-name;  
or  
let variable-name;
```

We can also define a variable without using the semicolon too. Also, when we have to define multiple variables, we can do it like this:

```
var x,y,z;  
or  
var x,y,z
```



JavaScript Variable Example:

Now let's take a simple example where we will declare a variable and then assign it a value.

```
var emp_name;  
var emp_name="dhavan";
```

JavaScript: Types of Variables

JavaScript supports two types of variables, they are:

- Local Variable
- Global Variable

You can use them according to the requirement in the application. Let's learn about both JavaScript Local variables and JavaScript Global variables with examples.

1. JavaScript Local Variable

JavaScript Local variable is a variable that is **declared inside a code block or a function body or inside a loop body** and it has scope within the code block or the function. In simple words, the scope of a local variable is between the opening and closing curly braces { }, when declared and defined inside a code block or a function body.

Starting from ES6 it is recommended to use the let keyword while declaring local variables.

A JavaScript local variable is declared inside block or function.

It is accessible within the function or block only.

For example:

```
<script>  
function abc()  
{  
var x=10; //x is a local variable  
}  
</script>
```

JavaScript Global Variable

JavaScript Global Variable is a variable that is **declared anywhere inside the script and has scope for the complete script execution**. Global variables are not declared inside any block or function but can be used in any function, or block of code.

Its recommended that we use the var keyword to declare the global variables, starting from ES6.



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.

Code:

```
<html>
<body>
<script>
var data=200; //global variable
function a()
{
document.write(data);
}
function b()
{
document.write(data);
}
a(); //calling JavaScript function
b();
</script>
</body>
</html>
```

Output:

200 200



Javascript let Keyword

In JavaScript, let is a keyword which is used to declare a local variable with block scope. Unlike var keyword which declares global scope variables, with **ECMAScript2016**(ES6) the let keyword was introduced to define local scope variables as defining all the variables in global scope leads to a lot of problems while writing large JavaScript applications.

It allows you to declare **limited scope variables** that cannot be accessible outside of the scope.

Let's take an example to understand the need of let keyword when we already had var keyword to create variables in JavaScript. Suppose you are writing a big JavaScript code which involves multiple loops, functions, and variables, as usual in all the loops you used the same variable name for the counter which is i, and you used the var keyword to define the variable i, now what will happen is, the variable i will carry on its changed value throughout the whole script as it is a global variable and if you forget to re-initialize it to zero anywhere, it will cause an error and your code will break. And you will have to put in extra efforts to look for the error.

Whereas, if you define the counter variable i using the let keyword, its scope will be limited to the loop only and when the first loop will end so will the counter variable. This way, using let keyword makes more sense because we use very few global variables and many local variables in general programming practice.

let does not create properties of the window object when declared globally.

The syntax for using the let keyword for defining a variable is the same as that of the var keyword.

```
let var1 [= value1] [, var2 [= value2]] [, ..., varN [= valueN];
```

As shown in the syntax above, yes, we can use a single let keyword **to define multiple variables**, but this is not new, we can do so using the var keyword too.

Let's see a few examples to see how this let keyword is used and how it works.

Use let inside a code block:

JavaScript variable declared inside a **block { }** cannot be accessed from outside the block, if we have defined the variable using the let keyword. See the below example:

```
{  
  let x = 2;  
}  
alert(x) // not accessible
```

Output:

```
uncaught ReferenceError: x is not defined
```

In the above example, the variable is accessible only inside the block. See the below example, to see that:



```
{  
  let x = 2;  
  alert(x) // accessible  
}
```

Output:

```
2
```

Use let inside a Loop:

It is more suitable for a loop, where we declare local variables to be used as counters. So, the variable does not conflict with the code written outside of the loop. See the below example:

```
let i = 5;  
for(let i = 0; i < 10; i++) {  
  // code  
}  
alert(i); // print 5
```

Output:

```
5
```

As you can see in the output it shows **5**, even though the loop incremented the value of **i** variable up to **10**, that is because of the scope of the local variable **i** in the for loop ending with the loop itself, hence it is not accessible outside the loop.

Use let inside a Function:

As we know, let keyword declares the local scope variable. So variable declared inside the function will retain within the function scope. If we will try accessing such variables from outside the function, we will get an error. See the below example:

```
function show()  
{  
  let amount = 2500; // Function Scope  
}  
alert(amount) // not accessible
```

Output:

```
Uncaught ReferenceError: amount is not defined
```

JavaScript let vs var Keyword

The let and var, both keywords are used to declare variables, but the main difference is the scope of the declared variables.



A variable declared inside a block using var is accessible outside of the block as it has a global scope but a variable declared using the let keyword has a local scope. Let's see an example:

```
{  
    let amount = 2500; // block Scope  
    var withdraw = 2000; // global scope  
}  
document.write(withdraw) // accessible  
document.write(amount) // not accessible
```

Output:

```
2000  
Uncaught ReferenceError: amount is not defined
```

JavaScript const keyword is used to define constant values that cannot be changed once a value is set. The value of a constant can't be changed through reassignment, and it can't be redeclared.

The scope of **const** is block-scoped it means it cannot be accessed from outside of block. In case of scope, it is much like variables defined using the **let** statement.

Constants can be either global or local to the block in which it is declared. Global constants do not become properties of the window object, unlike var variables.

JavaScript const Keyword:

Syntax

Below we have the syntax to define a constant value in JavaScript.

```
const name1 = value1 [, name2 = value2 [, ... [, nameN = valueN]]]
```

We don't have to use var or let keyword while using the const keyword. Also, we can **define a list or a simple value or a string etc as a constant**.

Lets understand, how to create constant in JavaScript program. See the below example:

```
{  
    const Pi = 3.14;  
    alert(Pi);  
}
```

Output:

```
3.14
```



Let's try another example where we will try changing the value of the constant and see if we allowed to reassign value or not.

```
{  
    const Pi = 3.14;  
    alert(Pi);  
    // Reassign value  
    Pi = 3.143;  
    alert(Pi);  
}
```

Output:

3.14

Uncaught TypeError: Assignment to constant variable.

The scope of the variable defined using `const` keyword is same as that of a variable declared using the `let` keyword. So, **constant declared inside a block will not accessible outside of that block**. Let's take an example and see:

```
{  
    const Pi = 3.14;  
    alert(Pi);  
}  
  
// outside block  
  
alert(Pi); // error
```

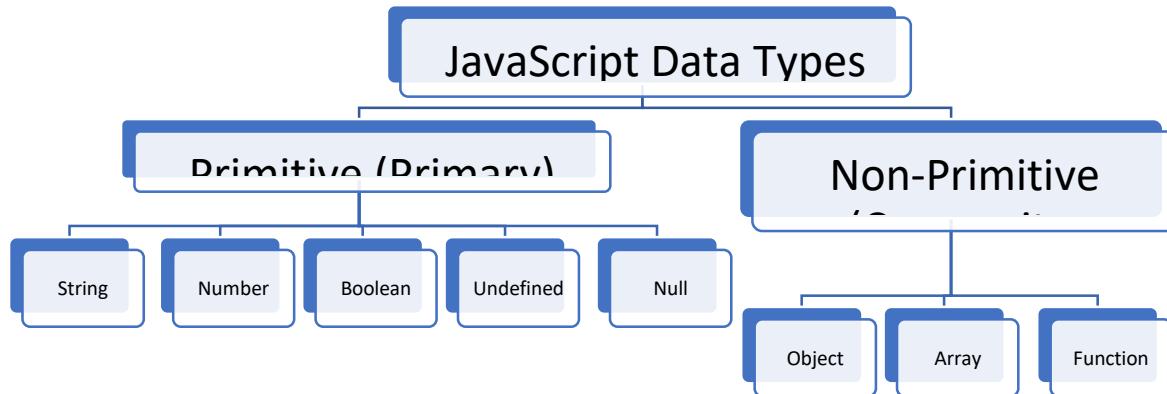
Output:

3.14

Uncaught ReferenceError: Pi is not defined

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/ Composite Data Types
- JavaScript is a dynamic type language; means you don't need to specify type of the variable.
- You need to use `var` here to specify the data type.
- It can hold any type of values such as numbers, strings etc.
- For example: `var a=40;//holding number`
 - `var b="Info Technology";//holding string`



Data Types: Primitive

Primitive data types can hold only one value at a time.

1) The String Data Type

The *string* data type is used to represent textual data (i.e. sequences of characters).

Strings are created using single or double quotes surrounding one or more characters, as shown below:

```
var a = 'Welcome'; // using single quotes  
Var+ b = "Welcome"; // using double quotes
```

2) The Number Data Type

- ✓ The *number* data type is used to represent positive or negative numbers with or without decimal place.
- ✓ The Number data type also includes some special values which are: Infinity, -Infinity, NaN(Not a Number)
- ✓ Example,

```
var a = 25; // integer  
var b = 80.5; // floating-point number  
var c = 4.25e+6; // exponential notation, same as 4.25e6 or 4250000  
var d = 4.25e-6; // exponential notation, same as 0.00000425
```

3) The Boolean Data Type

- ✓ The Boolean data type can hold only two values: True/False
- ✓ Example,

```
var a = 2, b = 5, c = 10;  
alert(b > a) // Output: true  
alert(b > c) // Output: false
```

4) The Undefined Data Type

- ✓ The undefined data type can only have one value—the special value “undefined”.



- ✓ If a variable has been declared, but has not been assigned a value, has the value "undefined".

- ✓ Example,

```
var a;  
var b = "Welcome";  
alert(a) // Output: undefined  
alert(b) // Output: Welcome
```

5) The Null Data Type

- ✓ A Null value means that there is no value.

- ✓ It is not equivalent to an empty string (" ") or zero or it is simply nothing.

- ✓ Example,

```
var a = null;  
alert(a); // Output: null  
var b = "Hello World!"  
alert(b); // Output: Hello World!  
b = null;  
alert(b) // Output: null
```

Data Types: Non-primitive

1) The Object Data Type

- ✓ a complex data type that allows you to store collections of data.

- ✓ An object contains properties, defined as a key-value pair.

- ✓ A property key (name) is always a string, but the value can be any data type, like strings, numbers, Boolean, or complex data types like arrays, function and other objects.

- ✓ Example,

```
var car =  
{model: "SUZUKI", color: "WHITE", model_year: 2019}
```

2) The Array Data Type

- ✓ An array is a type of object used for storing multiple values in single variable.

- ✓ Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, Booleans, functions, objects, and even other arrays.



- ✓ The array index starts from 0, so that the first array element is arr [0].
- ✓ The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:

```
✓ var cities = ["London", "Paris", "New York"];
✓ alert(cities[2]); // Output: New York
✓ var a = ["London", 500, "aa56", 5.6];
```

3) The Function Data Type

- ✓ The function is callable object that executes a block of code.
- ✓ Since functions are objects, so it is possible to assign them to variables, as shown in the example below:

```
var ab = function()
{
    return "Welcome";
}
alert(typeof ab); //output: function
alert(ab()); //output:Welcome
```

Code:

```
<html>
<body>
<h1>JavaScript Array</h1>
<script>
var stringArray = ["one", "two", "three"];
var mixedArray = [1, "two", "three", 4];
document.write(stringArray+<br>);
document.write( mixedArray);
</script>
</body>
</html>
```

JavaScript Array

one,two,three

1,two,three,4

Output:

Values/Literals

They are **types** that can be assigned a single **literal** value such as the number 5.7, or a string of characters such as "hello".



Types of Literals:

- Array Literal
- Integer Literal
- Floating number Literal
- Boolean Literal (include True and False)
- Object Literal
- String Literal

Array Literal:

- 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.
- Creating an empty array :
`var tv = [];`
Creating an array with four elements.
`var tv = ["LG", "Samsung", "Sony", "Panasonic"]`

✓ Comma in array literals:

- In the following example, the length of the array is four, and tv[0] and tv[2] are undefined.
- `var tv = [, "Samsung", , "Panasonic"]`
- This array has one empty element in the middle and two elements with values. (tv[0] is "LG", tv[1] is set to undefined, and tv[2] is "Sony")
`Var tv = ["LG", , "Sony",]`

Integer Literal:

An **integer** must have at least one digit (0-9).

- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)

Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

Example: 123, -20, 12345



2. Hexadecimal (base 16)

Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f.

A leading 0x or 0X indicates the number is hexadecimal.

Example: 7b, -14, 3039

3. Octal (base 8)

Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Example: 0173, -24, 30071

Floating number Literal:

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.') .
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Example of some floating numbers:

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4×10^3]
- 4E-3 [Equivalent to $4 \times 10^{-3} \Rightarrow .004$]

Object Literal:

An object literal is zero or more pairs of comma-separated list of property names and associated values, enclosed by a pair of curly braces.

In JavaScript an object literal is declared as follows:

1. An object literal without properties:

```
var userObject = {}
```

2. An object literal with a few properties :

```
var student = {  
    First-name : "Suresy",  
    Last-name : "Rayy",  
    Roll-No : 12  
};
```

Syntax Rules

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.



- There will be no comma after the last property name/value pair.

String Literal:

- JavaScript has its own way to deal with string literals.
- A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings.
- The following are the examples of string literals:
 - `string1 = "w3resource.com"`
`string1 = 'w3resource.com'`
 - `string1 = "1000"`
- In addition to ordinary characters, you can include special characters in strings, as shown in the following.
 - `string1 = "First line. \n Second line."`

Special characters in JavaScript:

| character | Description |
|-----------|-----------------|
| \' | Single quote |
| \" | Double quote |
| \\" | Backslash |
| \b | Backspace |
| \f | Form Feed |
| \n | New line |
| \r | Carriage return |
| \t | Horizontal tab |
| \v | Vertical tab |

Comments in JavaScript:

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.

There are two types of comments in JavaScript.

1. Single-line Comment

It is represented by double forward slashes (//).

It can be used before and after the statement.

Example,

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

2. Multi-line Comment

It can be used to add single as well as multi line comments.

It is represented by forward slash with asterisk then asterisk with forward slash.

Example,

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

1.4 Operators and Expression

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

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

1) Arithmetic Operators: used to perform arithmetic operations on the operands.

| Operator | Description | Example |
|----------|-------------|--------------|
| + | Addition | $10+20 = 30$ |



| | | |
|----|----------------|---------------------------|
| - | Subtraction | $20-10 = 10$ |
| * | Multiplication | $10*20 = 200$ |
| / | Division | $20/10 = 2$ |
| % | Modulus | $20\%10 = 0$ |
| ++ | Increment | var a=10; a++; Now a = 11 |
| -- | Decrement | var a=10; a--; Now a = 9 |

Code:

```
<html>
<body>
<script type = "text/javascript">
    var a = 33;
    var b = 10;
    var c = "Test";
    document.write("a + b = ");
    result = a + b;
    document.write(result+"<br>");

    document.write("a - b = ");
    result = a - b;
    document.write(result+"<br>");

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



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

document.write("a + b + c = ");
result = a + b + c;
document.write(result+<br>);

a = ++a;
document.write("++a = ");
result = ++a;
document.write(result+<br>);

b = --b;
document.write("--b = ");
result = --b;
document.write(result+<br>);

</script>
</body>
</html>
```

Output:

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8

2) Comparison (Relational Operator): Compares the two operands

| Operator | Example |
|----------|---------|
| | |



| | | |
|--------------------|------------------------------------|--------------------------------|
| <code>==</code> | Is equal to | <code>10==20 = false</code> |
| <code>===</code> | Identical (equal and of same type) | <code>10===20 = false</code> |
| <code>!=</code> | Not equal to | <code>10!=20 = true</code> |
| <code>!==</code> | Not Identical | <code>20!==20 = false</code> |
| <code>></code> | Greater than | <code>20>10 = true</code> |
| <code>>=</code> | Greater than or equal to | <code>20>=10 = true</code> |
| <code><</code> | Less than | <code>20<10 = false</code> |
| <code><=</code> | Less than or equal to | <code>20<=10 = false</code> |

Code:

```
<html>
<body>
<script type = "text/javascript">
    var a = 10;
    var b = 20;

    document.write("(a == b) => ");
    result = (a == b);
    document.write(result+ "<br>");

    document.write("(a < b) => ");
    result = (a < b);
    document.write(result+ "<br>");

    document.write("(a > b) => ");
    result = (a > b);
    document.write(result+ "<br>");

    document.write("(a != b) => ");
    result = (a != b);
    document.write(result+ "<br>");
```



```
document.write("(a >= b) => ");
result = (a >= b);
document.write(result+ "<br>");

document.write("(a <= b) => ");
result = (a <= b);
document.write(result+ "<br>");
</script>
</body>
</html>
```

Output:

(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true

3) **Bitwise Operator:** perform bitwise operations on operands

| Operator | Description | Example |
|----------|---------------------|---------------------------|
| & | Bitwise AND | (10==20 & 20==33) = false |
| | Bitwise OR | (10==20 20==33) = false |
| ^ | Bitwise XOR | (10==20 ^ 20==33) = false |
| - | Bitwise NOT | (-10) = -10 |
| << | Bitwise Left Shift | (10<<2) = 40 |
| >> | Bitwise Right Shift | (10>>2) = 2 |



| | | |
|-----|----------------------------------|---------------|
| >>> | Bitwise Right Shift with Zero | $(10>>2) = 2$ |
|-----|----------------------------------|---------------|

Code:

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

    var a = 2; // Bit presentation 10
    var b = 3; // Bit presentation 11
    document.write("(a & b) => ");
    result = (a & b);
    document.write(result+<br>);

    document.write("(a | b) => ");
    result = (a | b);
    document.write(result+<br>);

    document.write("(a ^ b) => ");
    result = (a ^ b);
    document.write(result+<br>);

    document.write("(-b) => ");
    result = (-b);
    document.write(result+<br>);

    document.write("(a << b) => ");
    result = (a << b);
    document.write(result+<br>);

    document.write("(a >> b) => ");
    result = (a >> b);
    document.write(result+<br>);

</script>
</body>
</html>
```



```
</script>
</body>
</html>
```

Output:

(a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
(a << b) => 16
(a >> b) => 0

4) Logical Operator:

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

Code:

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

    var a = true;
    var b = false;

    document.write("(a && b) => ");
    result = (a && b);
    document.write(result+ "<br>");


```



```
document.write("(a || b) => ");
result = (a || b);
document.write(result+ "<br>");

document.write("!(a && b) => ");
result = (!(a && b));
document.write(result+ "<br>");
</script>
</body>
</html>
```

Output:

(a && b) => false
(a || b) => true
!(a && b) => true

```
<!DOCTYPE html>
<html>
<body>
    <h1>Demo: JavaScript Logical Operators</h1>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>
    <script>
        var a = 5, b = 10;
        document.getElementById("p1").innerHTML = (a != b) && (a < b);
        document.getElementById("p2").innerHTML = (a > b) || (a == b);
        document.getElementById("p3").innerHTML = (a < b) || (a == b);
        document.getElementById("p4").innerHTML = !(a < b);
        document.getElementById("p5").innerHTML = !(a > b);
    </script>
</body>
</html>
```



Demo: JavaScript Logical Operators

true

false

true

false

true

5) Assignment Operator:

| 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 |

Code:

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

    var a = 33;
    var b = 10;

    document.write("Value of a => (a = b) => ");
    result = (a = b);
    document.write(result+<br>);

    document.write("Value of a => (a += b) => ");


</script>
</body>
</html>
```



```
result = (a += b);
document.write(result+"<br>");

document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result+"<br>");

document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result+"<br>");

document.write("Value of a => (a /= b) => ");
result = (a /= b);
document.write(result+"<br>");

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result+"<br>");
</script>
</body>
</html>
```

Output:

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0



6) Special Operator:

| Operator | Description |
|------------|---|
| (?:) | Conditional Operator/ternary 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 |

Code: *typeof* operator

```
<html>
<body>
<script type = "text/javascript">
    var a = 10;
    var b = "Information";
    var c= function(x)
    {
        return x*x;
    }

    document.write(typeof a+"<br>"); // a=10 datatype is number
</script>
</body>
</html>
```



```
document.write(typeof b+"<br>"); // b="Information" is a string  
document.write(typeof c+"<br>"); // c= function  
document.write(c(4));  
</script>  
</body>  
</html>
```

number
string
function
16

Output:

Code: *Ternary (?:)* operator

```
<script type = "text/javascript">  
    var a = 10;  
    var b = 20;  
    r=(a>=b)? "a is large":"b is large";  
    document.write(r);  
</script>
```

```
<script type = "text/javascript">  
    var a = 10;  
    var b = 20;  
    if(a>=b)  
        document.write("a is large");  
    else  
        document.write("b is large");  
</script>
```

Output:

b is large

In above example, if else can be replaced with ternary operator.

JavaScript Operator Precedence and Associativity

Operator precedence determines the order in which operators are evaluated. Operators with higher precedence are evaluated first. For example, the expression $(3+4*5)$, returns **23**, because of multiplication operator(*) having **higher precedence** than addition(+). Thus * must be evaluated first.

Operator associativity determines the order in which operators of the same precedence are processed. For example, **assignment operators are right-associative**, so you can write **a=b=5**, and with this statement, **a** and **b** are assigned the value **5**.



The below table shows the precedence and associativity of operators. In this table, precedence is from bottom to top i.e items at the bottom having low precedence and precedence increases as we move to the top of the table.

| Operator type | Operator (Symbol) | Associativity |
|----------------|-------------------|---------------|
| member | . | left-to-right |
| | [] | |
| new | new | right-to-left |
| function call | () | left-to-right |
| increment | ++ | |
| decrement | -- | |
| logical-not | ! | right-to-left |
| bitwise not | - | right-to-left |
| unary + | + | right-to-left |
| unary negation | - | right-to-left |
| typeof | typeof | right-to-left |
| void | void | right-to-left |
| delete | delete | right-to-left |
| multiplication | * | left to right |



| Operator type | Operator (Symbol) | Associativity |
|---------------|-------------------------|---------------|
| division | / | left to right |
| modulus | % | left to right |
| addition | + | left to right |
| subtraction | - | left to right |
| bitwise-shift | << >> >>> | left to right |
| relational | < <= > >= | left to right |
| in | in | left to right |
| instanceof | instanceof | left to right |
| equality | == != ==== !== | left to right |
| bitwise-and | & | left to right |
| bitwise-xor | ^ | left to right |



| Operator type | Operator (Symbol) | Associativity |
|---------------|--|---------------|
| bitwise-or | | left to right |
| logical-and | && | left to right |
| logical-or | | left to right |
| conditional | :? | right to left |
| assignment | = += -= *= /= %= <<= >>= >>>= &= ^= = , | right to left |
| comma | , | left to right |

Expression:

Any unit of code that can be evaluated to a value is an expression.

Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation.

Types of Expression:



1. Primary Expression:

Primary expressions refer to stand alone expressions such as literal values, certain keywords and variable values.

```
'hello world'; // A string literal  
23;           // A numeric literal  
true;          // Boolean value true  
sum;           // Value of variable sum  
this;          // A keyword that evaluates to the current object.
```

2. Object and Array Initializers

Object and array initializers are expressions whose value is a newly created object or array.

Object initializer expressions uses curly brackets, and each subexpression is prefixed with a property name and a colon.

Example, var emp={ name:"Aaa", branch:"IF"};

OR

```
var person={ };  
person.name="Aaa";  
person.branch="IF";
```

An array initializer is a comma-separated list of expressions surrounded with a square bracket.

Example, var tv=["LG", "Samsung"];

3. Property Access Expressions

A property access expression evaluates to the value of an object property or an array element.

JavaScript defines two syntaxes for property access:

```
expression.identifier;  
expression[identifier];
```

Example:

```
emp.firstName;  
emp.lastName;
```

4. Function Definition Expression

A function expression is part of a variable assignment expression and may or may not contain a name.

Since this type of function appears after the assignment operator =, it is evaluated as an expression.

Function expressions are typically used to assign a function to a variable.



Function expressions are evaluated only when the interpreter reaches the line of code where function expressions are located.

Example:

```
var sq=function (x)
{ return x*x;
}
```

5. Invocation Expression

An *invocation expression* is JavaScript's syntax for calling (or executing) a function or method.

It starts with a function expression that identifies the function to be called.

The function expression is followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis.

When an invocation expression is evaluated, the function expression is evaluated first, and then the argument expressions are evaluated to produce a list of argument values.

Example,

f(0) // f is the function expression; 0 is the argument expression

Math.max(x,y,z) // Math.max is the function; x, y and z are the arguments.

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

```
    
```

```
        var obj = { add: function(a, b)
                    {
                        return a + b;
                    }
                };alert(obj.add(4,5));delete obj.add;alert(obj.add(9,5));
            </script>
```

1.5 if statement, if...else. If...elseif, Nested if

Conditional statements are used to perform different actions based on different conditions. In JavaScript we have the following conditional statements:



- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

1) if statement:

Use `if` statement to specify a block of JavaScript code to be executed if a condition is true.

\Syntax:

Example:

```
<html>
  <body>
    <script>
      if (new Date().getHours() < 18)
      {
        document.write("Good day!");
      }
    </script>
  </body>
</html>
```



2) The else Statement

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

Syntax:

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

Example:

```
<html>
<body>
<script>
if (new Date().getHours() < 18)
{
    document.write("Good day!");
}
else
{
    document.write("Good Evening!");
}
</script>
</body>
</html>
```

3) The else if Statement

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

Syntax:



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

```
<html>
<body>
<script>
var greeting;
var time = new Date().getHours();
if (time < 10)
{
    greeting = "Good morning";
}
else if (time < 20)
{
    greeting = "Good day";
}
else
{
    greeting = "Good evening";
}
document.write(greeting);
</script>
</body>
</html>
```

4) The switch case Statement

The switch statement is used to perform different actions based on different conditions. It is used to select one of many code blocks to be executed.

Syntax:

```
switch(expression)
```

```
{
```

```
case x:
```

```
    // code block
```

```
break;
```



This is how it works:

The switch expression is evaluated once.

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

If there is a match, the associated block of code is executed.

Example:

```
<html>
<body>
<script>
var day;
switch (new Date().getDay())
{
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
}
```



```
case 6:  
    day = "Saturday";  
}  
document.write("Today is " + day);  
</script>  
</body>  
</html>
```

default keyword:

default keyword specifies the code to run if there is no case match.

The `getDay()` method returns the weekday as a number between 0 and 6.

If today is neither Saturday (6) nor Sunday (0), write a default message.

Example:

```
switch (new Date().getDay())  
{  
    case 6:  
        text = "Today is Saturday";  
        break;  
    case 0:  
        text = "Today is Sunday";  
        break;  
    default:  
        text = "Looking forward to the Weekend";  
}
```

Example:

```
<html>  
<body>  
<script>  
// program for a simple calculator  
var result;  
  
// take the operator input  
var operator = prompt('Enter operator ( either +, -, * or / ):');
```



```
// take the operand input
var number1 = parseFloat(prompt('Enter first number: '));
var number2 = parseFloat(prompt('Enter second number: '));

switch(operator)
{
    case '+':
        result = number1 + number2;
        document.write(`${number1} + ${number2} = ${result}`);
        break;

    case '-':
        result = number1 - number2;
        document.write(`${number1} - ${number2} = ${result}`);
        break;

    case '*':
        result = number1 * number2;
        document.write(`${number1} * ${number2} = ${result}`);
        break;

    case '/':
        result = number1 / number2;
        document.write(`${number1} / ${number2} = ${result}`);
        break;

    default:
        document.write('Invalid operator');
        break;
}
</script>
</body>
</html>
```

Output:



An embedded page on this page says

Enter operator (either +, -, * or /):

OK

Cancel

An embedded page on this page says

Enter first number:

OK

Cancel

An embedded page on this page says

Enter second number:

OK

Cancel

9 / 3 = 3

1.7 JavaScript Loop Statement

The JavaScript loops are used to *iterate the piece of code* using for, while, do while or for-in loops.

There are four types of loops in JavaScript.

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



1) 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.

Syntax:

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

Example:

```
<script>
for (i=0; i<=10; i=i+2)
{
    document.write(i + "<br/>")
}
</script>
```

2) do while Loop

loop is a variant of the while loop.

This loop will execute the code block once.

before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

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

Example:



```
<script>
var i=21;
do{
document.write(i +<br/>);
i++;
}while (i<=25);
</script>
```

3) while loop

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

Syntax:

```
while (condition)
{
Code to be executed
}
```

Example:

```
<script>
var i=11;
while (i<=20)
{
document.write(i + "<br/>");
i++;
}
</script>
```

4) for-in loop

The `for/in` statement loops through the properties of an object.

The block of code inside the loop will be executed once for each property.

Syntax:

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

Example:



```
<script type = "text/javaScript">
var lang = { first : "C", second : "Java",third : "Python", fourth : "PHP"};
for (prog in lang)
{
    document.write(lang[prog] + "<br >");
}
</script>
```

Output:

C
Java
Python
PHP



Difference between While Loop and Do – While Loop

| While Loop | Do – While Loop |
|---|--|
| In while loop, first it checks the condition and then executes the program. | In Do – While loop, first it executes the program and then checks the condition. |
| It is an entry – controlled loop. | It is an exit – controlled loop. |
| The condition will come before the body. | The condition will come after the body. |
| If the condition is false, then it terminates the loop. | It runs at least once, even though the conditional is false. |
| It is a counter-controlled loop. | It is an iterative control loop. |

break statement

break statement breaks the loop and continues executing the code after the loop.

The break statement can also be used to jump out of a loop.

Example:

```
<script type = "text/javascript">
var text = "";
var i;
for (i = 0; i < 10; i++)
{
    if (i === 4)
    {
        break;
    }
    text = text + "The number is " + i + "<br>";
}
document.write(text);
</script>
```

Output:

The number is 0
The number is 1
The number is 2
The number is 3



continue statement

Continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

Example:

```
<script type = "text/javaScript">
var text = "";
var i;
for (i = 0; i <= 6; i++)
{
    if (i === 4)
        {continue;
    }
    text =text + "The number is " + i + "<br>";
}
document.write(text);
</script>
```

Output:

The number is 0
The number is 1
The number is 2
The number is 3
The number is 5
The number is 6

1.8 Querying and Setting Properties

To obtain the value of a property, use . (dot) operator or square[] bracket.

The left-hand side should be an expression whose value is an object.

If using dot (.) operator, the right-hand must be a simple identifier that names the property.

If using square brackets, the value within the brackets must be an expression that evaluates to a string that contains the desired property name.

Example,

```
var name=author.lastname; //get the “lastname ” property of the book
var title=book[“main title”]; //get the “main title” property of the book
```

To create or set a property, use a dot or square brackets as you would to query the property, but put them on the left-hand side of an assignment expression:

Example,



```
book.price=250; //create or set a property of price  
book["main title"]="JavaScript" //set the "main title" property
```

Deleting properties:

The delete operator deletes a property from an object.

The delete operator deletes both the value of the property and the property itself.

Syntax:

```
delete var_name.property;
```

Example, `delete person.name;` or
`delete person["name"];`

Code:

```
<html>  
<body>  
<script>  
var a={name:"Priti",age:35};  
document.write(a.name+" "+a.age+"<br>");  
delete a.age; //delete property  
document.write(a.name+" "+a.age);  
</script>  
</body>  
</html>
```

Output:

```
Priti 35  
Priti undefined
```

Property getter and setter

Also known as Javascript assessors.

Getters and setters allow you to control how important variables are accessed and updated in your code.

JavaScript can secure better data quality when using getters and setters.

Following example access fullName as a function: person.fullName().

```
<script>  
// Create an object:  
var person = { firstName: "Chirag",  
              lastName : "Shetty",  
              fullName : function()  
              {  
                return this.firstName + " " + this.lastName;
```



```
    }  
};  
document.write(person.fullName());  
</script>
```

Following example `fullName` as a property: `person.fullName`.

```
<script>  
// Create an object:  
var person = { firstName: "Yash ", lastName : "Desai",  
    get fullName()  
    {  
        return this.firstName + " " + this.lastName;  
    }  
};  
// Display data from the object using a getter  
document.write(person.fullName());  
</script>
```



Code: **Getters and setters allow you to get and set properties via methods.**

```
<script>
var person = {
    firstName: 'Chirag',
    lastName: 'Shetty',
    get fullName()
    {
        return this.firstName + ' ' + this.lastName;
    },
    set fullName (name)
    {
        var words = name.split(' ');
        this.firstName = words[0];
        this.firstName = words[0].toUpperCase();
        this.lastName = words[1];
    }
}
document.write(person.fullName); //Getters and setters allow you to get and set
properties via methods.
document.write("<br>"+"before using set fullname()"+"<br>");
person.fullName = 'Yash Desai'; //Set a property using set
document.writeln(person.firstName); // Yash
document.write(person.lastName); // Desai
</script>
```

Output:

Chirag Shetty
before using set fullname()
YASH Desai



Unit 2

Array, Function and String

Marks: 14 (R-2, U-4, A-8)

Course Outcome: Implement arrays and functions in javascript.

Unit Outcome:

1. Create array to solve the given problem.
2. Perform the specified string manipulation operation on the given string.
3. Develop javascript to implement the given function.
4. Develop javascript to convert the given Unicode to character form.
5. Develop javascript to convert the given character to Unicode and vice-versa.

Topics and Sub-topics:

2.1 Array-Declaring an array, initializing an array, defining an array element, looping an array, adding an array element, sorting an array element, combining an array element into a string, changing elements of an array, Objects as Associative array.

2.2 Function-Defining a function, writing a function, adding an argument, scope of variable and arguments

2.3 Calling a function- calling a function with or without an argument, calling function from HTML, function calling another function, returning the value from a function.

2.4 String- manipulate a string, joining a string, retrieving a character from given position, retrieving a position of character in a string, dividing a text, copying a sub-string, converting string to number and numbers to string, changing the case of string, finding Unicode of a character.



2. Introduction

- Array, Function and String are the basic concepts in JavaScript.
- Array in JavaScript is used to store multiple values in a single variable.
- JavaScript function is the block of code designed to perform a particular task.
- A function is a group of reusable code which can be called anywhere in the JavaScript program. This eliminates the need of writing the same code again and again.
- The string object in JavaScript let you work with the series of characters. JavaScript strings are used for storing and manipulating text.

2.1 Array

- Array is used to store a set of values (different types) in a single variable name.
- Arrays are the fundamental part of most programming languages and scripting languages.
- Using array, we can store multiple values under a single name.

2.1.1. Declaring an Array

In JavaScript, objects and array are handled almost identically, because arrays are merely a special kind of object.

There are two ways to declare the arrays:

1) By using new Array() constructor:

The Array() constructor creates the Array objects..

You can declare an array with the “new” keyword to instantiate the array in memory.

You can invoke this constructor in three distinct ways:

A. Call it with no arguments:

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

This method is creating an empty array with no elements and is equivalent to the array literal [].

B. Call it with a single numeric argument, which specifies a length:

```
Var a = new Array(10);
```



This technique creates an array with specified length.

C. Explicitly specify two or more array elements or a single non-numeric element for the element for the array:

```
Var a = new Array(5,4,3,2,1,"testing", "testing");
```

In this form, the constructor arguments become the elements of the new array.

2) By using Literal Notation:

Instead of new Array(), you can use square brackets [].

When we declare array using square brackets is called the “array literal notation”:

```
Var x= [];
```

```
Var x=[5];
```

2.1.2 Initializing an Array:

Initialization is the process of assigning a value when either a variable or an array is declared.

When initializing an array, you place the value within the parentheses of the Array() constructor.

The following example initializes the products array with the value 'BMW', which is assigned to the first element of this array:

```
var products = new Array('BMW')
```

In the real world, an array usually has more than one array element, with each element having its own value. Therefore, you'll find yourself having to initialize the array with more than one value.

Here's how this is done:

```
var products = new Array('BMW', 'Maruti', 'Mahindra')
```

while Initializing array with declaration then all elements must specify in parenthesis and elements should be separated by a comma.

Code:

```
<html>
<head>
<title>Array</title>
</head>
```



```
<body>
<script>
var products = new Array('BMW', 'Maruti', 'Mahindra');
document.write("Length of array is :" + products.length);
</script>
</body>
</html>
```

Output:

Length of array is : 3

2.1.3 Defining Array Elements

Array is used to store a set of values in a single variable name. In order to differentiate between these set of values. Array make use of index.

Once array is declared we can define its elements as follows:

```
var cars = new Array(3);

cars[0] = "BMW";

cars[1] = "Maruti";

cars[2] = 2546 ;
```

As JavaScript automatically changes the value of length when you add elements to an array that you created with the Array keyword. Array indicates in JavaScript always start at 0 , not 1, so the length property is always one greater than the largest index in the array.

Accessing an array value is quite easy. We use array index to access a value. If we want to access val 1 from the above syntax , we use Array[0], So,

Array[0] holds “BMW”

Array[1] holds “Maruti”

Array[2] holds “Honda”

Code:

```
<html>
<head>
<title> Array</title>
</head>
```



```
<body>
<script>
var cars = new Array(3);
cars[0] = "BMW";
cars[1] = "Maruti";
cars[2] = "Honda";
document.write(cars[0]);
document.write("<br>" + cars[1]);
document.write("<br>" + cars[2]);
</script>
</body></html>
```

Output:

BMW

Maruti

Honda

2.1.4 Looping an Array

Loops are handy, if you want to run the same code over and over again, each time with a different value. We can use arrays within loops and access array elements using loops in JavaScript.

To identify how many loops, we must know the total number of elements present inside the array which can find out using array_name.length.

Example:

```
<html>
<body>
<h2>JavaScript For Loop</h2>
<script>
var cars = ["BMW", "Volvo", "Ford", "Fiat"];
var text = "";
var i;
for (i = 0; i < cars.length; i++)
{
document.write(cars[i]+"<br>");
}
</script>
</body>
</html>
```

Output:

JavaScript For Loop

BMW

Volvo

Ford

Fiat



2.1.4 Adding an Array Element

On some occasions your JavaScript will need to increase the size of the array while your JavaScript is running.

There are two methods to add the elements into the array:

Method1:

The easiest way to add a new element to an array is using the `push()` method.

The `push()` method adds new items to the end of an array, and returns the new length.

Syntax:

```
array.push(item1, item2, ..., itemX);
```

Example:

```
var fruits = [ "Banana", "Orange", "Apple", "Mango" ];
fruits.push( "Lemon" ); // adds a new element (Lemon) to fruits
```

Method 2:

The `unshift()` method adds one or more elements to the beginning of an array and returns the new length of the array.

Syntax:

```
array.unshift(item1, item2, ..., itemX);
```

Example:

```
var fruits = [ "Banana", "Orange", "Apple", "Mango" ];
fruits.unshift( "Lemon", "Pineapple" );
```

Example:

```
<html>
<head>
<title> Array</title>
<body>
<script>
    var fruits = new Array(3);
    fruits[0] = "Banana";
    fruits[1] = "Orange";
    fruits[2] = "Apple";
    fruits[3] = "Mango";
    for(i=0;i<fruits.length;i++)
    {
        document.write(fruits[i] + " +"<br> );
    }
</script>

```



```
fruits.push("Lemon");
fruits.unshift("Pineapple");

for(i=0;i<fruits.length;i++)
{
    document.write(fruits[i] + " ");
}
</script>
</body>
</html>
```

Output

Banana
Orange
Apple
Mango
Pineapple Banana Orange Apple Mango Lemon

2.1.5 Sorting Array Elements:

The index of the array elements determines the order in which values appear in an array when a for loop is used to display the array. Sometimes you want values to appear in sorted order, which means that strings will be presented alphabetically and numbers will be displayed in ascending order. You can place an array in sorted order by calling the sort() method of the array object. the index of the element to which the value is assigned.

Here's what you need to do to sort an array:

1. Declare the array.
2. Assign values to elements of the array.
3. Call the sort() method.

Example:

```
<html>
<body>
<script>
```



```
var products = new Array();
products[0] = 'Soap ';
products[1] = 'Water';
products[2] = 'Pizza';
products[3] = 'Fan';
products.sort();
for (var i = 0; i < products.length; i++)
{
document.write(products[i] + '<br>');
}
</script>
</body>
</html>
```

Output:

Fan
Pizza
Soap
Water

2.1.6 Reversing an Array Element:

The reverse() method reverses the elements in an array.

- You can use it to sort an array in descending order.
- Syntax: array.reverse();

Example:

```
<html>
<body>
<script>
var fruits = ["Banana", "Watermelon", "Chikoo", "Mango", "Orange", "Apple"];
fruits.sort();
document.write(fruits+ "<br>");
fruits.reverse();
```



```
document.write(fruits+"<br>");  
</script>  
</body>  
</html>
```

Output:

Apple,Banana,Chikoo,Mango,Orange,Watermelon

Watermelon,Orange,Mango,Chikoo,Banana,Apple

2.1.7 Combining an Array Element into String

Sometimes there is need to combine all elements of an array into a single string.

Let car is an array consist of three elements as follows:

```
var car = new Array(3);  
car[0] = "BMW";  
car[1] = "Maruti";  
car[2] = "Honda";
```

Array elements can be combined in two ways:

1)Using the concat() method:

The concat() method separates each value with a comma. It is used to join one or more arrays.

Syntax:

```
array.concat()
```

Example:

```
var CO_Subject = ["PHP", "CSS", "Java"];  
var Math_Subject= ["Applied Math", "Elements of Maths"];  
var subjects = CO_Subject.concat(Math_Subject);  
document.write(subjects);
```

Output:

PHP,CSS,Java,Applied Math,Elements of Maths

2. Using the join() method:



- The `array.join()` method is an inbuilt function in JavaScript which is used to join the elements of an array into a string.
- The elements of the string will be separated by a specified separator and its default value is comma(,).

Syntax: `array.join(separator);`

Parameters: *

Separator: It is Optional.

it can be either used as parameter or not. Its default value is comma(,).

Example:

```
<html>
<body>
<script>

var products = new Array();
products[0] = 'Car ';
products[1] = 'Water';
products[2] = 'Soap';
products[3] = 'Pizza';
var str = products.concat();
document.write(str);
document.write('<br>');
var str = products.join(' ');
document.write(str);
</script>
</body>
</html>
```

Output:

Car,Water,Soap,Pizza

Car Water Soap Pizza



2.1.8 Changing Elements of an Array:

JavaScript provides **shift()** method to remove the first element of an array.

The shift() method removes the first element of the array then moves the other tasks up on the list and returns the removed item.

Example:

```
<html>
<head>
<title> Array</title>
</head>
<body>
<script>
Var car = new Array(3);
car[0] = "BMW";
car[1] = "Honda";
car[2] = "Maruti";
car.shift();
for (i=0;i<car.length;i++)
{
    document.write(items[i]+");
}
</script>
</body>
</html>
```

Output:

Honda Maruti



The **pop()** method remove an item from the end of an array

Example:

```
<html>
<head>
<title> Array</title>
</head>
<body>
<script>
    Var car = new Array(3);
    car[0] = "BMW";
    car[1] = "Honda";
    car[2] = "Maruti";
    car.pop();
    for (i=0;i<car.length;i++)
    {
        document.write(items[i]+");
    }
</script>
</body>
</html>
```

Output:

BMW Honda

2.1.9 Changing Element of an Array

The **splice()** method can be used to add new items to an array, and removes elements from an array.

Syntax: arr.splice(start_index, removed_elements, list_of_elements_to_be_added);

Parameter:

- The first parameter defines the position where new elements should be added (spliced in).
- The second parameter defines how many elements should be removed.
- The `list_of_elements_to_be_added` parameter define the new elements to be added(optional).

Example:

```
<html>
```



```
<body>
<script>
var fruits = ["Banana", "Watermelon", "Chikoo", "Mango", "Orange", "Apple"];
document.write(fruits+<br>);
fruits.splice(2,2, "Lemon", "Kiwi");
document.write(fruits+<br>);
fruits.splice(0,2); //removes first 2 elements from array document.write(fruits+<br>);
</script>
</body>
</html>
```

Output:

Banana,Watermelon,Chikoo,Mango,Orange,Apple
Banana,Watermelon,Lemon,Kiwi,Orange,Apple
Lemon,Kiwi,Orange,Apple

The **slice()** method slices out a piece of an array into a new array.

Syntax: arr.slice(array starting from array element 1);

Parameter:

·slices out a part of an array starting from array element 1.

Example:

```
<html>
<body>

<script>
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
document.write(fruits);
var citrus = fruits.slice(2);
document.write(" <br> "+citrus);
</script>

</body>
</html>
```

Output:



Banana,Orange,Lemon,Apple,Mango
Lemon,Apple,Mango

2.1.10 Object as Associative Array

- Arrays are JavaScript objects.
- The dot (.) operator can be used to access object property.
- The [] operator can also be used to access array property.
- Thus, the following two JavaScript expressions have the same value:

```
object.property ;  
object["property"] ;
```

- To refer to an object property using array notation, simply pass the property name as a String to the array square brackets applied to the object, as follows:

```
objectName["propertyName"] ;
```

Example:

```
<html>  
<body>  
<script>  
var object1 = new Object;  
object1.name = "abc";  
object1.nationality = "Indian";  
document.write(" property name: " + object1["name"] );  
document.write("<br>");  
document.write(" property nationality: " + object1["nationality"] );  
</script>  
</body>  
</html>
```

OUTPUT :

property name: Girija
property nationality: Indian

Methods of Array Object:



| Sr. No. | Methods | Description |
|---------|-------------|---|
| 1 | concat | Returns a new array consisting of a combination of two array |
| 2 | indexOf | Returns the index of the first occurrence of a value in an array. |
| 3 | Join | Joins all elements of an array into a string |
| 4 | lastIndexOf | Returns the index of the last occurrence of a specified value in an array |
| 5 | Pop | Removes the last element of an array |
| 6 | Push | Add new elements to the end of an array and returns the new length |
| 7 | Reverse | Reverses the order of the elements in an array |
| 8 | Shift | Removes the first element of an array and return that element |
| 9 | Slice | Selects a part of an array, and returns the new array |
| 10 | Sort | Sorts the elements of an array |
| 11 | Splice | Adds/Remove elements from array |
| 12 | toString | Converts an array to a string, and returns the result |
| 13 | unshift | Adds new element at the beginning of an array |
| 14 | valueOf | Returns the primitive value of an array |

Example: Using array methods:

```
<html>
<body>
<script>
    var arr = new Array();
    arr.push(1,2,3);
    arr.push(5,6);
    document.write("After the Join method" +arr.join(","));
    arr.pop();
    document.write("<br>After the pop method" +arr.join(","));
    arr.shift();
    document.write("<br>After the shift method" +arr.join(","));

    arr.reverse();
    document.write(" <br>After the reverse method"+arr.join(", "));

    arr.unshift(1);
    document.write("<br>After the unshift method" +arr.join(","));

```



```
</script>
</body>
</html>
```

Output:

```
After the Join method1,2,3,5,6
After the pop method1,2,3,5
After the shift method2,3,5
After the reverse method5,3,2
After the reverse method1,5,3,2
```

2-Dimensional Array:

The two-dimensional array is a *collection of items which share a common name and they are organized as a matrix in the form of rows and columns.*

The two-dimensional array is an array of arrays, so we create an array of one-dimensional array objects.

Example:

```
var branch = [
    ['Computer Engg', "CO"],
    ['Information Technology', "IF"],
    ['Electronics and Telecommunication', "EJ"]];
```

Example: 2-D array

```
<script>
var branch = [
    ['Computer Engg', "CO"],
    ['Information Technology', "IF"],
    ['Electronics and Telecommunication', "EJ"],
    ['Civil Engineering', "CV"],
    ['Chemical Engg', "CE"],
    ['Instrument Engg', "IE"]
];
// display now
```



```
for(i = 0; i < branch.length; i++)
document.write(branch[i][0] + ',' + branch[i][1] + '<br>');
</script>
```

Output:

Computer Engg,CO
Information Technology,IF
Electronics and Telecommunication,EJ
Civil Engineering,CV
Chemical Engg,CE
Instrumnet Engg,IE

Multi-Dimensional Array:

Multidimensional arrays are not directly provided in JavaScript. If we want to use anything which acts as a multidimensional array then we need to create a multidimensional array by using another one-dimensional array. So multidimensional arrays in JavaScript is known as arrays inside another array. We need to put some arrays inside an array, then the total thing is working like a multidimensional array. The array, in which the other arrays are going to insert, that array is use as the multidimensional array in our code. To define a multidimensional array, its exactly the same as defining a normal one-dimensional array.

Example:

```
<script>
var my_ans = new Array(); // declaring array
my_ans.push({0:45,1:55,2:65,3:45});
my_ans.push({0:145,1:155,2:165,3:"VP"});
my_ans.push({0:245,1:255,2:265});
my_ans.push({0:"aaa",1:"bbb",2:"ccc",3:"ddd"});

// displaying the array data //
```



```
for(i=0;i<4;i++)
{
    document.write("key : " + i + " =>value: " + my_ans[i][0] +
    ',' +my_ans[i][1] + ',' +my_ans[i][2]+ ',' +my_ans[i][3] + "<br>");
}
</script>
```

Output:

```
key: 0 =>value: 45,55,65,45
key: 1 =>value: 145,155,165, VP
key : 2 =>value: 245,255,265, undefined
key: 3 =>value: aaa,bbb,ccc,ddd
```

2.2 Function

Functions are building blocks of any programming language. Function is a block of statements that perform certain tasks.

Functions are of the two types:

1. Built-in functions are the functions that are already defined in JavaScript. Examples are written (), prompt () etc.

2. User-Defined functions are defined by the user.

2.2.1 Defining a function

- A function must be defined before it can be called in a JavaScript statement. If you think about it, this makes sense, because the browser must learn the definition of the word (the function name) before the browser sees the word (the function call) in a statement.
- The best place to define a function is at the beginning of a JavaScript that is inserted in the <head> tag, because then all subsequent JavaScript on the web page will know the definition of that function. The browser always loads everything in the <head> tag before it starts executing any JavaScript.
- A function definition consists of four parts: the name, parentheses, a code block, and an optional return keyword.

Function Name



The function name is the name that you've assigned the function. It is placed at the top of the function definition and to the left of the parentheses. Any name will do, as long as it follows certain naming rules. The name must be

- Letter(s), digit(s), or underscore character
- Unique to JavaScript on your web page, as no two functions can have the same name

The name cannot

- Begin with a digit
- Be a keyword
- Be a reserved word

Parentheses

- Parentheses are placed to the right of the function name at the top of the function definition.
- Parentheses are used to pass values to the function; these values are called arguments.

Code Block

- The code block is the part of the function definition where you insert JavaScript statements that are executed when the function is called by another part of your JavaScript application.
- Open and close French braces define the boundaries of the code block. Statements that you want executed must appear between the open and close French braces.

Return (Optional)

- The return keyword tells the browser to return a value from the function definition to the statement that called the function.

2.2.2 Writing function definition

We can write function with argument and without argument in JavaScript.

Syntax for function without argument:

```
function function_name(parameters...)  
{  
    Statements ....
```

{

Example:

```
function hellojavascript()
{
    alert("hello");
}
```

2.2.3 Adding an Argument

- A function typically needs data to perform its task. Sometimes you provide the data when you define the function, such as the salary and percentage increase in salary instead of writing this data into the function definition.
- Other times, the data is known only when you run your JavaScript.
- For example, we could ask the user to enter the salary and percentage increase in salary instead of writing this data into the function definition.
- Data that is needed by a function to perform its task that is not written into the function definition must be passed to the function as an argument.
- An argument is one or more variables that are declared within the parentheses of a function definition.

Syntax:

```
function function_name(arg1, arg2)
{
    lines of code to be executed
}
```

Example:

```
<html>
<body>
    <h1>Demo: JavaScript function parameters</h1>
    <script>
        function ShowMessage(firstName, lastName)
        {
            alert("Hello " + firstName + " " + lastName)
        }
        ShowMessage("Steve", "Jobs");
    </script>
</body>
</html>
```

```
</script>
</body>
</html>
```

2.2.4 Scope of Variable and Arguments

- The scope of a variable means how the different parts of the program can access that variable. In JavaScript there are two types of Scope namely: Local Scope and Global Scope.
- A variable can be declared within a function this is called a local variable, because the variable is local to the function. Other parts of your JavaScript don't know that the local variable exists because it's not available outside the function.
- But a variable can be declared outside a function. Such a variable is called a global variable because it is available to all parts of your JavaScript—that is, statements within any function and statements outside the function can use a global variable.
- JavaScript developers use the term scope to describe whether a statement of a JavaScript can use a variable. A variable is considered in scope if the statement can access the variable. A variable is out of scope if a statement cannot access the variable.

Example of Local Variable:

```
function myFunction()
{
    var carName = "Volvo";
    // code here CAN use carName
}
```

Example of Global Variable:

```
var carName = "Volvo";

// code here can use carName

function myFunction()
{
    // code here can also use carName
}
```

2.3 Calling a Function

- You call a function any time that you want the browser to execute statements contained in the code block of the function.
- A function is called by using the function name followed by parentheses. If the function has arguments, values for each argument are placed within the parentheses.
- You must place these values in the same order that the arguments are listed in the function definition. A comma must separate each value.

2.3.1 Calling function without argument:

- Here is an example of how to define and call a function that does not have any arguments.
- The function definition is placed within the `<head>` tag and the function call is placed within the `<body>` tag.
- When the function is called, the browser goes to the function definition and executes statements within the code block of the function.

Example:

```
<html>
<body>
<script>
function IncreaseSalary()
{
var salary = 500000 * 1.25;
alert('Your new salary is ' + salary);
}
IncreaseSalary();
</script>
</body>
</html>
```

Output:

This page says
Your new salary is 625000

OK

2.3.2 Calling function with arguments:

- The Salary and Increase variables are then used within the parentheses of the function call, which tells the browser to assign these values to the corresponding arguments in the function definition. The function calculates and displays the new salary.

Example:

```
<html>
<body>
<script>
function IncreaseSalary(OldSalary, PerIncrease)
{
var NewSalary =
OldSalary * (1 + (PerIncrease / 100))
alert("Your new salary is " + NewSalary)
}
var Salary = prompt('Enter old salary.', '')
var Increase =
prompt('Enter salary increase as percent.', '')
IncreaseSalary(Salary, Increase)
</script>
</body>
</html>
```



Output:

This page says

Enter old salary.

OK

Cancel

This page says

Enter salary increase as percent.

OK

Cancel

This page says

Your new salary is 2400

OK

2.3.3. Calling Function from HTML

- A function can be called from HTML code on your web page. Typically, a function will be called in response to an event, such as when the web page is loaded or unloaded by the browser.
- You call the function from HTML code nearly the same way as the function is called from within a JavaScript, except in HTML code you assign the function call as a value of an HTML tag attribute.

```
<html>
```

```
<script >
```

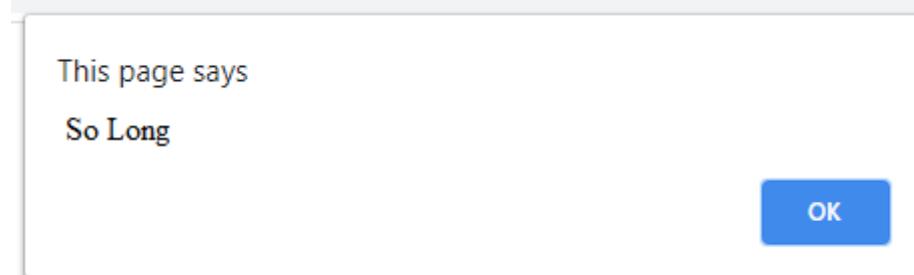
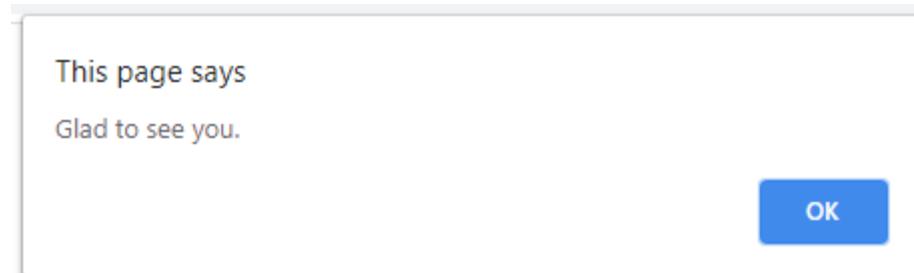


```
function WelcomeMessage()
{
    alert('Glad to see you.')
}

function GoodbyeMessage()
{
    alert('So long.')
}

</script>
<body onload="WelcomeMessage()"
onunload="GoodbyeMessage()">
</body>
</html>
```

Output:



2.3.4 Function Calling another Function



JavaScript developers typically divide an application into many functions, each of which handles a portion of the application.

```
<html>
<head>
<title> function calling another function </title>
<script>
function Logon()
{
var userID;
var password;
var valid;
userID=prompt('Enter user ID', ' ');
password=prompt('Enter Password', ' ');
valid=validateLogon(userID, password);
if(valid === true)
{
    alert("Valid Logon");
}
else
{
    alert("InValid Logon");
}
}

function validateLogon(id, pwd)
{
var ret_val;
if(id === '111' && pwd === 'aaa')
{
    ret_val=true;
}
else
{
    ret_val=false;
}
return ret_val;
```

```
}
```

```
</script>
```

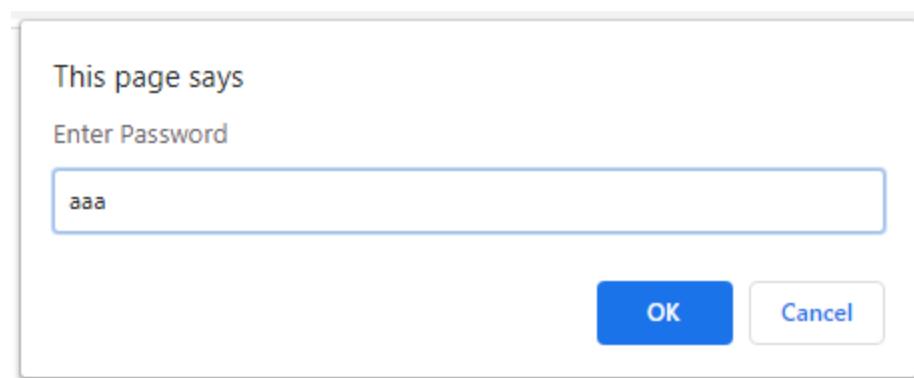
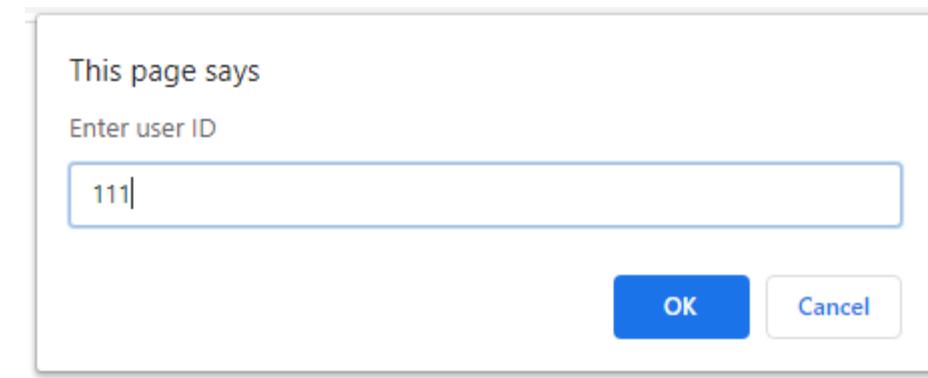
```
</head>
```

```
<body onload="Logon()">
```

```
</body>
```

```
</html>
```

Output:



2.3.5 Returning a value from functions:



- A function can be designed to do something and then report back to the statement that calls after it's finished
- A function reports back to the statement that calls the function by returning a value to that statement using the return keyword, followed by a return value in a statement.

```
<html>
<body>
<script>
var x = myFunction(4, 3);
function myFunction(a, b)
{
    return a * b;
}
document.write(x);
</script>
</body>
</html>
```

Output:

12

2.4 String

- A string value is chain of zero or more Unicode characters.
- You use string data type to represent text in JavaScript.
- The String object is used to storing and manipulating text. A string is simply storing the series of characters.
- There are 2 ways to create string in JavaScript

A) By string literal:

The string literal is created using double quotes.

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

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

B) By string object (using new keyword)

Syntax:



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

2.4.1 Manipulating string:

In JavaScript there are various properties and methods associated with string objects.

We can make use of these to perform some manipulating on string.

String Properties:

| Sr. No. | Property | Description |
|---------|-------------|--|
| 1 | constructor | This property returns a reference to the string function that created the object |
| 2 | length | Returns the length of a string. |
| 3 | prototype | Allows you to add new properties and methods to a String object. |

Code: Length Property

```
<script type = "text/javascript">
    var str = new String( "Vidyalankar Polytechnic" );
    document.write("str.length is:" + str.length);
</script>
```

Output: str.length is:23

Code: constructor Property

```
<html>
    <head>
        <title>JavaScript String constructor Method</title>
    </head>

    <body>
        <script type = "text/javascript">
            var str = new String();
            document.write("str.constructor is:" + str.constructor);
        </script>
    </body>
</html>
```



Output:

```
str.constructor is: function String() { [native code] }
```

Code: prototype property

```
// without using prototype property
```

```
<html>
<body>
    <h1>without using prototype property</h1>
    <script>
        function Student()
        {
            this.name = 'Pallavi';
            this.gender = 'F';
        }

        var studObj1 = new Student();
        studObj1.age = 20;
        document.write(studObj1.age+<br>);

        var studObj2 = new Student();
        document.write(studObj2.age);
    </script>
</body>
</html>
```

Output:

without using prototype property

```
20
undefined
```

With Prototype property:

```
<html>
```



```
<body>
  <h1>Prototype</h1>
  <script>
    function Student()
    {
      this.name = 'Pallavi';
      this.gender = 'M';
    }
    Student.prototype.age = 20;
    var studObj1 = new Student();
    document.write(studObj1.age+<br>);
    var studObj2 = new Student();
    document.write(studObj2.age);
  </script>
</body>
</html>
```

Output:

Prototype

20

20

String Methods:

| Methods | Description |
|---------------|---|
| charAt() | It provides the char value present at the specified index |
| charCodeAt() | It provides the Unicode value of a character present at the specified index. |
| concat() | It provides a combination of two or more strings. |
| indexOf() | It provides the position of a char value present in the given string. |
| lastIndexOf() | It provides the position of a char value present in the given string by searching a character from the last position. |



| | |
|----------------|---|
| search() | It searches a specified regular expression in a given string and returns its position if a match occurs. |
| match() | It searches a specified regular expression in a given string and returns that regular expression if a match occurs. |
| replace() | It replaces a given string with the specified replacement. |
| substr() | It is used to fetch the part of the given string on the basis of the specified starting position and length. |
| substring() | It is used to fetch the part of the given string on the basis of the specified index. |
| slice() | It is used to fetch the part of the given string. It allows us to assign positive as well negative index. |
| toLowerCase() | It converts the given string into lowercase letter. |
| toUowerCase() | It converts the given string into uppercase letter. |
| toString() | It provides a string representing the particular object. |
| valueOf() | It provides the primitive value of string object. |
| split() | It splits a string into substring array, then returns that newly created array. |
| trim() | It trims the white space from the left and right side of the string. |
| fromCharCode() | The fromCharCode() method converts Unicode values into characters. |

2.4.2 Joining a String:

- When you concatenate a string, you form a new string from two strings by placing a copy of the second string behind a copy of the first string.
- The new string contains all the characters from both the first and second strings.
- You use the concatenation operator (+) to concatenate two strings, as shown here

NewString = FirstString + SecondString

```
<html>
<bod>
<script>
var s1="Rahul";
var s2="Patil";
```



```
var s3=s1.concat(s2);
document.write(s3);
var s4=s1+s2;
document.write("<br>"+s4);
</script>
</body>
</html>
```

Output:

RahulPatil
RahulPatil

2.4.3 Retrieving a character from given position

- The charAt() method requires one argument, which is the index of the character that you want to copy.

```
<html>
<body>
<script>
var str="javascript";
document.write(str.charAt(2));
</script>
</body>
</html>
```

Output:

v

2.4.4 Retrieving a position of characters in a string

- To find a character or characters in string we have two methods, indexOf() and search()
- You can determine the index of a character by calling the indexOf() method of the string object.
- The indexOf() method returns the index of the character passed to it as an argument.
- The search () method searches a string for a specified value and returns the position of the match.



```
<html>
<body>
<script>
var s1="JavaScript is a scripting language";
var n=s1.indexOf("a");
document.writeln(n+<br>);
document.writeln(s1.search("scripting"));
var m=s1.lastIndexOf("a");
document.writeln("<br>"+m);
</script>
</body>
</html>
```

Output:

1
16
31

2.4.5 Dividing Text

- The `split()` method creates a new array and then copies portions of the string, called a *substring*, into its array elements.
- You must tell the `split()` method what string (*delimiter*) is used to separate substrings in the string.
- You do this by passing the string as an argument to the `split()` method.

```
<html>
<body>
<script>
var str="CO IF EJ";
document.write(str.split(" "));
</script>
</body>
</html>
```

Output:

CO,IF,EJ



2.4.6 Copying a Substring:

- Now you've learned how to divide a string into many substrings by using the `split()` method and a string called a *delimiter*.
- This is useful when you are separating a string containing data elements into individual data elements.
- However, the `split()` method isn't of much use to you if you need to copy one substring.
- For this, you'll need to use one of two other methods: `substring()` and `substr()`.

1. The `substring()` is a method of a `String` object that copies a substring from a `String` based on a beginning and an end position that is passed as an argument to the `substring()` method.

Syntax:

```
String.substring(startindex, endindex);
```

2. In the real world, you probably won't know the starting position and end position of characters for your substring, because a user can enter any length string into your application. You can overcome this problem by using the `substr()` method. The `substr()` method returns a substring. You must tell it the starting position of the first character that you want to include in the substring and how many characters you want copied into the substring from the starting position. Both positions are passed as arguments to the `substr()` method.

Syntax:

```
String.substr(startindex,length);
```

```
<html>
<body>
<script>
var str="JavaScript";
document.write(str.substr(0,6));
document.write("<br>");
document.writeln(str.substring(4,9));
</script>
</body>
</html>
```

Output:



2.4.7 Converting String to number:

- If you need to convert string values to number values, you can do so by converting a number within a string into a numeric value that can be used in a calculation.

- You do this by using the **parseInt()** method and **parseFloat()** method of the string object. The parseInt() method converts a number in a string to an integer numeric value, which is a whole number. You write the parseInt() method this way:

```
var num = parseInt(StringName)
```

- The parseFloat() method is used similarly to the parseInt() method, except the parseFloat() method is used with any number that has a decimal value.

- **Number()**: converts a string into number.

```
var StrPrice = '10.95'  
var NumPrice = parseFloat(StrCount)
```

```
<html>  
<body>  
<script>  
var a=50;  
var b="67";  
var c="45.75";  
var ans=a + parseInt(b)+parseFloat(c);  
document.write("Addition="+ans);  
var sum=a+ Number(b)+parseFloat(c);  
document.write("<br>"+ "SUM=" +sum);  
</script>  
</body>  
</html>
```

Output:

Addition=162.75
SUM=162.75

2.4.8 Converting number to string



- As you can probably guess, you need to convert a numeric value to a string before the number can be used in the string.
- You do this by calling the `toString()` method of the number object.
- The `toString()` method can be used to convert both integers and decimal values (floats).
- Here's how to convert a number value to a string:

```
Var NumCount = 100  
var StrCount = NumCount.toString()
```

```
<html>  
<body>  
<script>  
var a=50;  
var b=80;  
var ans=a + b.toString();  
document.write("Addition="+ans);  
</script>  
</body>  
</html>
```

Output:

Addition=5080

2.4.9 Changing the Case of the string

- JavaScript provides two methods to change the case of string by using `toLowerCase()` and `toUpperCase()` method.

1. `toLowerCase()`: this method converts all the string character to lowercase. It does not take any argument.

2. `toUpperCase()`: this method converts all the string character to uppercase. It does not take any argument.

```
<html>  
<body>  
<script>  
var str = "JavaScript";
```



```
document.writeln(str.toLowerCase());
document.writeln("<br>" + str.toUpperCase());
</script>
</body>
</html>
```

Output:

```
javascript
JAVASCRIPT
```

2.4.9 Finding a Unicode of a character

- Unicode is a standard that assigns a number to every character, number, and symbol that can be displayed on a computer screen, including characters and symbols that are used in all languages.
- You can determine the Unicode number or the character that is associated with a Unicode number by using the **charCodeAt()** method and **fromCharCode()** method. Both are string object methods.
- The **charCodeAt()** method takes an integer as an argument that represents the index of the character in which you're interested. If you don't pass an argument, it defaults to index 0.
- The **charCodeAt()** method returns the Unicode number of the string:
var UnicodeNum = StringName.charCodeAt()
- If you need to know the character, number, or symbol that is assigned to a Unicode number, use the **fromCharCode()** method. The **fromCharCode()** method requires one argument, which is the Unicode number.

```
<html>
<body>
<script>
var x = "Javatpoint";
document.writeln(x.charCodeAt(3));
document.write("<br>");
var res = String.fromCharCode(72, 69, 76, 76, 79);
var res1 = String.fromCharCode(73, 70, 77, 77, 80);
document.write(res);
document.write("<br>" + res1);
```



```
</script>
</body>
</html>
```

Output:

97
HELLO
IFMMP

Code:charCodeAt()

```
<script>
var x="Javatpoint";
document.writeln(x.charCodeAt(3));
</script>
```

Output:

97

2.4.10 JavaScript String replace() Method

```
<script>
var str="JavaProgramming";
document.writeln(str.replace("Programming", "Script"));
</script>
```

Output:

JavaScript

2.4.11 JavaScript String search() Method

The search() method searches a string for a specified value, and returns the position of the match.

The search value can be string or a regular expression.

This method returns -1 if no match is found.



Syntax: string.search(searchvalue);

```
<script>
var str="JavaScript is a scripting language.";
document.writeln(str.search("scripting"));
</script>
```

Output:

16

2.4.12 JavaScript String match() Method

The **string.match()** is an inbuilt function in JavaScript which is used to search a string for a match against an any regular expression and if the match will found then this will return the match as an array otherwise it returns null.

Syntax:

string.match(regExp);

```
<script>
var str="JavaProgramming";
document.writeln(str.match("Java"));
</script>
```

Output:

Java

22.4.13 JavaScript String slice() Method

The slice() method returns the selected elements in an array, as a new array object.

The slice() method selects the elements starting at the given *start* argument, and ends at, *but does not include*, the given *end* argument

Syntax: array.slice(*start, end*);

Where,



Start: Optional. An integer that specifies where to start the selection (The first element has an index of 0). Use negative numbers to select from the end of an array. If omitted, it acts like "0"

end: Optional. An integer that specifies where to end the selection. If omitted, all elements from the start position and to the end of the array will be selected. Use negative numbers to select from the end of an array

Code:

```
<script>
var str = "JavaScript";
document.writeln(str.slice(0));
document.writeln("<br>" + str.slice(4));
</script>
```

Output:

JavaScript
Script

Code: Write a javascript function to insert a string within a string at a particular position.

```
<html>
<body>
<button onclick = "myfunction()">click</button>
<script language="javascript" type="text/javascript">
function myfunction()
{
    var s1 = "client scripting";
    var s2 = " side";
    var output = [s1.slice(0, 6), s2, s1.slice(6)].join("");
    document.write(output);
}
</script>
</body>
</html>
```

Output:



V2V EDTECH LLP

DIPLOMA | DEGREE | BSCIT



unit 3

Form and Event Handling

Marks: 10 (R-2, U-4, A-4)

Course Outcome: Create event-based web forms using javascript.

Unit Outcome:

1. Write JavaScript to design a form to accept input values for the given problem.
2. Use JavaScript to implement form events to solve the given problems.
3. Develop JavaScript to dynamically assign specified attribute value to the given form control.
4. Use the given intrinsic functions with specified parameters.

Topics and Sub-topics:

- 3.1: Building blocks of a Form, properties and methods of form, button, text, text area, checkbox, radio button, select element.
- 3.2: Form events – Mouse Events, Key Events
- 3.3: Form objects and elements
- 3.4: Changing attribute value dynamically
- 3.5: Changing option list dynamically
- 3.6: Evaluating checkbox selection
- 3.7: Changing a label dynamically
- 3.8: Manipulating form elements
- 3.9 Intrinsic JavaScript functions, disabling elements, read only elements

Forms are one of the most common web page elements used with JavaScript.

JavaScript is commonly used with following two reasons:

- To add functionality that makes forms easier for users to fill out



- To validate or process the data that a user enters before that data is submitted to a server-side script.

JavaScript form object represents HTML form. HTML forms are a very powerful tool for interacting with users.

An HTML form is used to collect user input. The user input can then be sent to a server for processing. JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

3.1: Building blocks of a Form, properties and methods of form, button, text, text area, checkbox, radio button, select element:

A form is a section of an HTML document that contains elements such as radio buttons, text boxes and option lists. HTML form elements are also known as **controls**.

Elements are used as an efficient way for a user to enter information into a form. Typical form control objects also called "**widgets**" includes the following:

- ✓ Text box for entering a line of text.
- ✓ Push button for selecting an action.
- ✓ Radio buttons for making one selection among a group of options.
- ✓ Check boxes for selecting or deselecting a single, independent option.

The <form> element can contain one or more of the following form elements:

- <input>
- <textarea>
- <button>
- <select>
- <option>
- <fieldset>
- <label>
- <legend>

The attributes of the form are:

| Attribute | Value | Description |
|-----------|---------------|---|
| action | URL | Specifies where to send the form-data when a form is submitted |
| method | get post | Specifies the HTTP method to use when sending form-data |
| name | text | Specifies the name of a form |
| id | Any id | Unique identifier |
| onSubmit | Function name | Event fired up when the form is submitted and before the execution of the action. |



Syntax:

```
<form name = "myform" id = "myform" action = "page.html" onSubmit = "test()">  
-----objects-----  
</form>
```

HTML Form Elements:

| Sr. No | HTML Element | Type Property | Event Handler | Description and Events |
|--------|---|---------------|---------------|---|
| 1 | <input type = "button"> or <button type = "button"> | "button" | onclick | A push buttons. |
| 2 | <input type = "checkbox"> | "checkbox" | onchange | A toggle button without radio button behavior. |
| 3 | <input type = "file"> | "file" | onchange | An input held for entering the name of a file to upload to the web server, value property is read only. |
| 4 | <input type = "hidden"> | "hidden" | none | Data submitted with the form but not visible to the user. |
| 5 | <option> | none | none | A single item within a select object, event handlers are on the select object and not on individual option objects. |
| 6 | <input type = "password"> | "password" | onchange | An input field for password entry where typed characters are not visible. |
| 7 | <input type = "radio"> | "radio" | onchange | A toggle button with radio button behavior where only one item is selected at a time. |
| 8 | <input type = "reset"> or <button type = "reset"> | "reset" | onclick | A push button that resets a form. |



| | | | | |
|----|--|--------------------|----------|--|
| 9 | <select> | “select-on e” | onchange | A list or drop-down menu from which one item may be selected. |
| 10 | <select multiple> | “select-mu ltiple” | onchange | A list from which multiple items are selected. |
| 11 | <input type = “submit”> or <button type = “submit”> | “submit” | onclick | A push button that submits a form. |
| 12 | <input type = “text”> | “text” | onchange | A single line text entry field. |
| 13 | <textarea> | “textarea” | onchange | A multi-line text entry field. |
| 14 | <label> | | | defines a label |
| 15 | <fieldset> | | | tag is used to group related elements in a form. tag draws a box around the related elements. |

<input> tag with its parameters:

1. name: Can be used so that the value of the element can be processed.
2. type: Can be used to specify the type of input.
3. id: Identification name of element.
4. value: Can be used to specify the initial value. It is required when type is set to checkbox or radio. It should not be used when type is set to file.
5. checked: Can be used when type is set to checkbox or radio to set the initial state of a checkbox or radio button to be selected.
6. maxlength: Can be used to specify the maximum number of characters allowed in a textbox.
7. src: Can be used when type is set to image to specify the location of an image file.
8. alt: Can be used when type is set to image to specify the alternative text of the image, which should be a short description.

Code: To accept first name, last name, email and birthdate. After clicking on button, details will be displayed as an output.

```
<html>
<head>
<style>
fieldset
```



```
{  
background-color: pink;  
}  
  
legend  
{  
background-color: gray;  
color: white;  
padding: 5px 10px;  
}  
  
input  
{  
margin: 5px;  
}  
</style>  
</head>  
<body>  
<form action=" ">  
<fieldset>  
<legend>Personalia:</legend>  
<label for="fname">First name:</label>  
<input type="text" id="fname" name="fname"><br><br>  
<label for="lname">Last name:</label>  
<input type="text" id="lname" name="lname"><br><br>  
<label for="email">Email:</label>  
<input type="email" id="email" name="email"><br><br>  
<label for="birthday">Birthday:</label>  
<input type="date" id="birthday" name="birthday"><br><br>  
</fieldset>  
</form>  
<p>Click the button to get the details:</p>  
<button onclick="myFunction()">Details</button>  
<BR>  
<p id="demo"></p>  
<p id="demo1"></p>  
<p id="demo2"></p>  
<p id="demo3"></p>
```



```
<script>
function myFunction()
{
    var y = document.getElementById("fname").value;
    document.getElementById("demo").innerHTML = y;
    var x = document.getElementById("lname").value;
    document.getElementById("demo1").innerHTML = x;
    var z = document.getElementById("email").value;
    document.getElementById("demo2").innerHTML = z;
    var w = document.getElementById("birthday").value;
    document.getElementById("demo3").innerHTML = w;
}
</script>
</body>
</html>
```

Output:

Personalia:

First name:

Last name:

Email:

Birthday:

Click the button to get the details:

Manisha

Padwal

manisha.padwal@vpt.edu.in

2020-07-30



Properties and Methods:

- The Form object represents a <form> element in an HTML document. The elements property is an HTML collection that provides convenient access to all elements of the form. The submit () and reset () methods allow a form to be submitted or reset under program control.
- Each form in a document is represented as an element of the documents.forms[] array. The elements of a form are collected in the array like object Form.elements.

Properties of Form:

| Sr. No. | Properties | Description |
|---------|-------------|--|
| 1 | action | Read/Write string that reflects the action attribute of the form. |
| 2 | elements[] | An array containing all of the elements of the form. Use it to loop through form easily. |
| 3 | encoding | Read/Write string that specifies how the form data is encoded. |
| 4 | length | The number of elements in the form. |
| 5 | method | Read/Write string that specifies how the method the form is submitted. |
| 6 | name | The name of the form. |
| 7 | target | The name of the target frame or window form is to be submitted to. |

Methods of Form:

| Sr. No. | Methods | Description |
|---------|----------|-----------------|
| 1 | reset() | Resets the form |
| 2 | submit() | Submits a form |

Methods of Events:

| Sr. No. | Methods | Description |
|---------|----------|--|
| 1 | onReset | Code is executed when the form is reset. |
| 2 | onSubmit | Code is executed when form is submitted. |

Code: Assign values to the text boxes after clicking on a button.

```
<html> <head>
<script type="text/javascript">
function assign()
{

```



```
document.forms.book.title.value="CSS_book";
document.forms.book.author.value="Manisha Padwal";
}
</script>
</head>
<body>
<form id="book">
Title of Book:<input id="title"> <br> <br>
Author of Book:<input id="author"> <br> <br>
<input type="button" id="btn" value="Assign Values" onclick="assign()">
</form>
</body>
</html>
```

Output:

Title of Book:

Author of Book:

Forms and the elements they contain can be selected from a document using

- getElementById() method
- getElementsByName() method
- getElementsByTagName() method
- getElementsByClassName() method
- innerHTML property
- innerText property

~~getElementById() method~~.

- The getElementById() method returns the element that has the ID attribute with the specified value.
- This method is one of the most common methods in the HTML **DOM**, and is used almost every time you want to manipulate, or get info from, an element on your document.



Syntax:

```
document.getElementById(elementID);
```

Code: Following code displays after clicking on button, text color is changed as red.

```
<html>
<body>

<p id="demo">Click the button to change the color of this paragraph.</p>

<button onclick="myFunction()">change color</button>

<script>
function myFunction()
{
    var x = document.getElementById("demo");
    x.style.color = "red";
}
</script>
</body>
</html>
```

Output:

Click the button to change the color of this paragraph.

Click the button to change the color of this paragraph.

2. getElementsByName() method



The `getElementsByName()` method returns a collection of all elements in the document with the specified name (the value of the `name` attribute).

Syntax:

```
document.getElementsByName(name);
```

Code: Check all `<input>` elements with `type="checkbox"` in the document that have a `name` attribute with the value "animal":

```
<html>
<body>
<input name="program" type="checkbox" value="IF">
Information Technology <br>
<input name="program" type="checkbox" value="CO">
Computer Engineering
<br>
<p>Click the button to check all checkboxes that have a name attribute with the value "program".</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
    var x = document.getElementsByName("program");
    var i;
    for (i = 0; i < x.length; i++)
    {
        if (x[i].type == "checkbox")
        {
            x[i].checked = true;
        }
    }
}
</script>
</body>
</html>
```

Output:



- Information Technology
- Computer Engineering

Click the button to check all checkboxes that have a name attribute with the value "program".

Try it

- Information Technology
- Computer Engineering

Click the button to check all checkboxes that have a name attribute with the value "program".

Try it

3. `getElementsByName()` method

The `getElementsByName()` method returns a collection of all elements in the document with the specified tag name.

Syntax:

```
document.getElementsByName(tagname);
```

Code: Following code illustrates the use of `getElementsByName()` to count how many LI elements are present in unordered list.

Find out how many `` elements there are in the document.

```
<html>
<body>
<p>An unordered list:</p>
<ul>
  <li>Information Technology</li>
  <li>Computer Engineering</li>
  <li>Chemical Engineering</li>
</ul>
<p>Click the button to find out how many li elements there are in this document. </p>
<button onclick="myFunction()">Click</button>
<p id="demo"></p>
<script>
function myFunction()
```



```
{  
    var x = document.getElementsByTagName("LI");  
    document.getElementById("demo").innerHTML = x.length;  
}  
</script>  
</body>  
</html>
```

Output:

An unordered list:

- Information Technology
- Computer Engineering
- Chemical Engineering

Click the button to find out how many li elements there are in this document.

Click

3

Code: Change the background color of all `<p>` elements in the document as pink and text color as blue.

```
<html>  
<body>  
<p>Computer Engineering</p>  
<p>Information Technology</p>  
  
<p>Electronics and Telecommunication</p>  
<button onclick="myFunction()">Try it</button>  
<script>  
function myFunction()  
{  
    var x = document.getElementsByTagName("P");  
    var i;  
    for (i = 0; i < x.length; i++) {  
        x[i].style.backgroundColor = "pink";  
        x[i].style.color = "blue";  
    }  
}
```



```
}
```

```
</script>  </body>  </html>
```

Output:

Computer Engineering

Computer Engineering

Information Technology

Information Technology

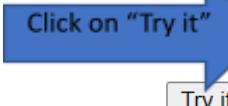
Electronics and Telecommunication

Electronics and Telecommunication

[Try it](#)

[Try it](#)

Click on "Try it"



4) getElementsByClassName() Method

The **getElementsByClassName()** method returns an object containing all the elements with the specified class names in the document as objects. Each element in the returned object can be accessed by its index. This method can be called upon any individual element to search for its descendant elements with the specified class names.

Syntax:

```
document.getElementsByClassName(classnames);
```

Parameters: This method takes only one parameter, which is a string containing space-separated class names of the elements to search for.

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>DOM getElementByClassName() Method</title>
<style>
h1 {
    color: green;
}
body {
    text-align: center;
}
.example {
    padding: 10px;
    margin: auto;
    margin-top: 10px;
    border: 1px solid black;
}
```



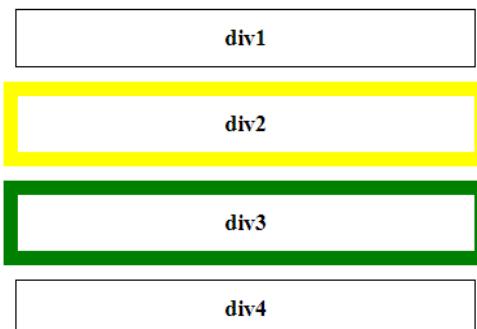
```
width: 300px;
}
</style>
</head>

<body>
<h1>Client_side Scripting</h1>
<h2>DOM getElementByClassName() Method</h2>
<div>
<h4 class="example">div1</h4>
<h4 class="yellowBorder example">div2</h4>
<h4 class="greenBorder example">div3</h4>
<h4 class="example">div4</h4>
</div>
<script>
document.getElementsByName('greenBorder example')[0]
    .style.border="10px solid green";
document.getElementsByName('yellowBorder example')[0]
    .style.border="10px solid yellow";
</script>
</body>
</html>
```

Output:

Client_side Scripting

DOM getElementByClassName() Method



5) innerHTML property



The easiest way to modify the content of an HTML element is by using the innerHTML property.

To change the content of an HTML element, use this syntax:

```
document.getElementById(id).innerHTML = new HTML;
```

Code: to change the text by using innerHTML property.

```
<html>
<body>
<script type="text/javascript">
function changeText()
{
document.getElementById('js').innerHTML = 'Fifth Semester Javascript!!!!';
}
</script>
<p>Welcome to <b id='js'>JAVASCRIPT</b> </p>
<input type='button' onclick='changeText()' value='Change Text' />
</body>
</html>
```

Output:

Welcome to **JAVASCRIPT** Welcome to **Fifth Semester Javascript!!!!**

Change Text

Change Text

Code: Script for count the number of <p> tag and <H2> tag.

```
<html>
<head>
<style>
div
{
border: 1px solid black;
margin: 5px;
}
</style>
```



```
</head>
<body>
<div id="myDIV">
<p>Information Technology</p>
<p>Computer Engg.</p>
<p>Electronics and Telecommunication</p>
<p>Chemical Engg.</p>
</div>
<div id="myh">
<H2>Vidyalankar Polytechnic</H2>
<H2>Vidyalankar Institute of Technology </H2>
</div>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<p id="demo1"></p>
<script>
function myFunction()
{
    var x = document.getElementById("myDIV").getElementsByTagName("P");
    document.getElementById("demo").innerHTML = x.length;
    var y = document.getElementById("myh").getElementsByTagName("H2");
    document.getElementById("demo1").innerHTML = y.length;
}
</script>
</body>
</html>
```

Output:



Information Technology
Computer Engg.
Electronics and Telecommunication
Chemical Engg.

Vidyalankar Polytechnic

Vidyalankar Institute of Technology

Try it

4

2

Code:

```
<script type="text/javascript">
function changeText()
{
    var userInput = document.getElementById('userInput').value;
    document.getElementById('vp').innerHTML = userInput;
}
</script>
<p>Welcome <b id='vp'>JavaScript</b> </p>
<input type='text' id='userInput' value='Enter Text Here' />
<input type='button' onclick='changeText()' value='Change Text' />
```

Output:

Welcome JavaScript

Enter Text Here

Change Text

Welcome Java Programming

Java Programming

Change Text

Code:

```
<html>
<body>
Name: <input type="text" id="userInputName" value=""> <br> <br>
Password: <input type="password" id="userInputPwd" value=""> <br> <br>
<input type="button" onclick="changeText()" value="Change Text">
<p>Name is <b id="vp">JavaScript</b> </p>
```



```
<p>Password is <b id="vp1">JavaScript</b> </p>
<script>
function changeText()
{
var userInputName = document.getElementById("userInputName").value;
document.getElementById("vp").innerHTML = userInputName;
var userInputPwd = document.getElementById("userInputPwd").value;
document.getElementById("vp1").innerHTML = userInputPwd;
}
</script>
</body>
</html>
```

Output:

Name:

Name: VP

Password:

Password:

Name is **JavaScript**

Name is **VP**

Password is **JavaScript**

Password is **information Tech**

Code: Execute a JavaScript when a form is submitted.

```
<html>
<body>
<p>When you submit the form, a function is triggered which alerts some text.</p>
<form action="" onsubmit="myFunction()">
    Enter name: <input type="text" name="fname">
    <input type="submit" value="Submit">
</form>
<script>
function myFunction()
{
    alert("The form was submitted");
}
</script>
```



```
</body>
</html>
```

Output:

When you submit the form, a function is triggered which alerts some text.

Enter name:

This page says

The form was submitted

OK

<input> Element of form:

<input> tag defines the start of an input field where the user can enter data.

Syntax:

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

6) innerText

property

The innerText property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

In this example, we are going to display the password strength when releases the key after press.

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

</script>
```



```
<form name="myForm">  
<input type="password" value="" name="userPass" onkeyup="validate()">  
Strength:<span id="mylocation">no strength</span>  
</form>
```

Attributes of <input> Tag:

Name: Name assigns a name to the input field. The name of the input field is used to send the information to the server.

Value: This attribute sets the value for the input field.

Type: type attributes indicates the type of input element that has to be given in following table.

| Value | Description |
|----------|---|
| button | Defines a clickable button (mostly used with a JavaScript to activate a script) |
| checkbox | Defines a checkbox |
| color | Defines a color picker |
| date | Defines a date control (year, month, day (no time)) |
| email | Defines a field for an e-mail address |
| hidden | Defines a hidden input field |
| image | Defines an image as the submit button |
| password | Defines a password field |
| radio | Defines a radio button |
| reset | Defines a reset button |
| submit | Defines a submit button |
| text | Default. Defines a single-line text field |

Button:

Button is created by using following code:

```
<form method = "GET" action = "">  
<input type = "button" name = "MyButton" value = "Click" onclick = "msg()">  
</form>
```

There are several types of button, which are specified by the type attribute:

1. Button which corresponds to the graphic component.
2. Submit, which is associated to the form and which starts the loading of the file assigned to the action attribute.
3. Image button in which an image loaded from a file.

A Button object also represents an HTML <button> element which is specified as follows:



```
<button name = "btn" value = "MyButton" onclick = "msg()">
```

Inside a <button> element you can put content, like text or images. But this is not the case with the buttons created with <input> tag.

| Attribute | Value | Description |
|-----------|---------------------------|---|
| name | <i>name</i> | Specifies a name for the button |
| type | button reset submit | Specifies the type of button |
| value | <i>text</i> | Specifies an initial value for the button |

Event handling with Button:

A push button that activates a JavaScript when it is clicked:

Syntax:

```
<input type="button" value="aaa" onclick="function_name()">
```

OR

```
<button onclick="Function_name()"> Click me </button>
```

Code:

```
<html>
<body>
<h2>Show a Push Button</h2>
<p>The button below activates a JavaScript when it is clicked. </p>
<form>
  <input type="button" value="Click me" onclick="msg()">
</form>
<script>
function msg()
{
  alert("Hello world!");
}
</script>
</body>
</html>
```

Output:



← → C ⌂ File | C:/Users/Admin/Desktop/Demo/form/button_event_demo.html

Show a Push Button

The button below activates a JavaScript when it is clicked.

This page says

Hello world!

Code:

```
<html>
<body>
<h3>The onclick Event using Button tag. </h3>
<button onclick="myFunction()">Click me</button>
<p id="demo"></p>
<script>
function myFunction()
{
    document.getElementById("demo").innerHTML = "Welcome to JavaScript";
}
</script>
</body>
</html>
```

Output:

The onclick Event using Button tag.

Welcome to JavaScript

Code:

```
<html>
<body>
<p id="demo">Click me.</p>
<script>
document.getElementById("demo").onclick = function()
{
    myFunction()
};

function myFunction()
```



```
{  
    document.getElementById("demo").innerHTML = "YOU CLICKED ME!";  
}  
</script>  
</body>  
</html>
```

Output:

Click me. **YOU CLICKED ME!**

Text:

Input “text” is an object to enter a single line of text whose content will be part of form data.

In html a text is created by following code:

```
<input type="text" name="textname" id="textid" value=" assign_value" />
```

Code:

```
<script type="text/javascript">  
function changeText()  
{  
    var userInput = document.getElementById('userInput').value;  
    document.getElementById('vp').innerHTML = userInput;  
}  
</script>  
  
<input type='text' id='userInput' value='Enter Text Here' />  
    <p>Welcome <b id='vp'>JavaScript</b> </p>  
<input type='button' onclick='changeText()' value='Change Text'/'>  
</script>
```

Output:



Enter Text Here

CSS

Welcome **JavaScript**

Change Text

Change Text

TextArea

The Textarea object represents an HTML <textarea> element.

The <textarea> tag indicates a form field where the user can enter a large amount of text.

You can access a <textarea> element by using getElementById():

Attributes of TextArea tag:

| Property | Description |
|----------|--|
| cols | Sets or returns the value of the cols attribute of a text area |
| name | Sets or returns the value of the name attribute of a text area |
| rows | Sets or returns the value of the rows attribute of a text area |
| value | Sets or returns the contents of a text area |
| wrap | Sets or returns the value of the wrap attribute of a text area. <ul style="list-style-type: none">• Soft: "Soft" forces the words to wrap once inside the textarea but once the form is submitted, the words will no longer appear as such, and line breaks and spacing are not maintained.• Hard : "Hard" wraps the words inside the text box and places line breaks at the end of each line so that when the form is submitted the text will transfer as it appears in the field, including line breaks and spacing.• "Off": sets a textarea to ignore all wrapping and places the text into one ongoing line. <p style="text-align: center;"><i>textareaObject.wrap = soft/hard/off</i></p> |
| readonly | Setting a "yes" or "no" value for the readonly attribute determines whether or not a viewer has permission to manipulate the text inside the text field. |
| disabled | Disabling the textarea altogether prevents the surfer from highlighting, copying, or modifying the field in any way. To accomplish this, set the <i>disabled</i> property to "yes". |

Code: to demonstrate the <textarea> and its attributes.

```
<html>
<body>

<textarea cols="30" rows="5" wrap="hard" readonly="yes" disabled="yes">
As you can see many times word wrapping is often the desired look for your textareas.
Since it makes everything nice and easy to read and preserves line breaks.
</textarea>
```



```
</body>
```

```
</html>
```

Output:

As you can see many times word wrapping is often the desired look for your textareas. Since it makes everything nice and easy to

A screenshot of a web browser showing a disabled text area. The text "As you can see many times word wrapping is often the desired look for your textareas. Since it makes everything nice and easy to" is displayed, with each line starting at the same vertical position and ending with a line break. The text area has scroll bars on the right side.

In above code, disabled="yes" that is textarea is disabled (cannot perform any highlighting, copying, or modifying) .

Without using “disabled” attribute of <textarea>

```
<textarea cols="30" rows="5" wrap="hard" readonly="yes">
```

Output will be like:

As you can see many times word wrapping is often the desired look for your textareas. Since it makes everything nice and easy to

A screenshot of a web browser showing an enabled text area. The text "As you can see many times word wrapping is often the desired look for your textareas. Since it makes everything nice and easy to" is displayed, with each line starting at a different vertical position and ending with a line break. The text area has scroll bars on the right side.

Code: to display the content from textarea.

```
<html>
<body>
Address:<br>
<textarea id="myTextarea" cols=32 Rows=5>
</textarea>
<p>Click the button to get the content of the text area.</p>
<button type="button" onclick="myFunction()">Display Address</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var x = document.getElementById("myTextarea").value;
    document.getElementById("demo").innerHTML = x;
}
</script>
</body>
```



```
</html>
```

Output:

Address:

```
Vidyalankar Polytechnic,  
Antop Hill,  
Wadala,  
Mumbai-400037
```

Click the button to get the content of the text area.

Vidyalankar Polytechnic, Antop Hill, Wadala, Mumbai-400037

Methods of TextArea:

| Method | Description |
|----------|--|
| select() | Selects the entire contents of a text area |

Code: Select the contents of a text area.

```
<html>  
<body>  
Address:<br>  
<textarea id="myTextarea" cols=32 Rows=5>  
</textarea>  
<p>Click the button to get the content of the text area.</p>  
<button type="button" onclick="myFunction()">Display Address</button>  
<p id="demo"></p>  
<script>  
function myFunction()  
{  
    var x = document.getElementById("myTextarea").value;  
    document.getElementById("demo").innerHTML = x;  
}  
</script>  
</body>  
</html>
```

Output:

**Address:**

Vidyalanakr
Polytechnic
Wadala
Mumbai

Click the button to select the contents of the text area.

Try it

Checkbox:

<input> elements of type checkbox are rendered by default as boxes that are checked (ticked) when activated. A checkbox allows you to select single values for submission in a form (or not).

Syntax for creating checkbox is:

```
<input type="checkbox" id="myCheck" onclick="myFunction()">
```

A checkbox can have only two states:

1. Checked
2. Unchecked

Code:

```
<html>
<body>
<div>
Program:
<br>
<input type="checkbox" name="program" id="it" value="IT">
<label for="it">Information Tech</label> <br>

<input type="checkbox" name="program" id="co" value="CO" checked>
<label for="co">Computer Engg</label> <br>

<input type="checkbox" name="program" id="ej" value="EJ">
<label for="ej">Electronics</label> <br>
<button onclick="validate();">Validate</button>
</div>
```



```
<div id="status">
</div>
<script>
function validate()
{
    var elements = document.getElementsByName("program");
    var statusText = " ";
    for (var index=0; index <elements. Length; index++)
    {
        statusText = statusText + elements[index].value+"="+elements[index].checked+"<br>";
    }
    document.getElementById("status").innerHTML = statusText;
}
</script>
</body>
</html>
```

Output:

Program:

- Information Tech
- Computer Engg
- Electronics

IT=true

CO=true

EJ=false

Radio Button:

The radio button allows the user to choose one of a predefined set of options. You can define groups with the name property of the radio buttons.

Radio buttons with the same name belong to the same group. Radio buttons with different names belongs to the different groups. At most one radio button can be checked in a group.

Syntax:



```
<input type="radio" id="male" name="gender" value="male">
```

Code:

```
<html>
<body>
<form method="post" action=" " onsubmit="return ValidateForm();">
    <fieldset>
        <legend>Select Course:</legend>
        <input type="radio" name="br" value="IT" checked>IT<br>
        <input type="radio" name="br" value="CO">CO<br>
        <input type="radio" name="br" value="EJ">EJ<br>
        <br>
        <input type="submit" value="Submit now">
    </fieldset>
</form>
<script type="text/javascript">
function ValidateForm()
{
    var obj = document.getElementsByName("br");
    for(var i = 0; i < obj.length; i++)
    {
        if(obj[i].checked == true)
        {
            if(confirm("You have selected " + obj[i].value))
                return true;
            else
                return false;
        }
    }
}
</script>
</body>
</html>
```

Output:



Select Course:

IT
 CO
 EJ

This page says
You have selected IT

Select Course:

IT
 CO
 EJ

This page says
You have selected CO

Select:

Form SELECT elements (<select>) within your form can be accessed and manipulated in JavaScript via the corresponding Select object.

To access a SELECT element in JavaScript, use the syntax:

```
document.myform.selectname //where myform and selectname are names of your form/element.
```

```
document.myform.elements[i] //where i is the position of the select element within form
```

```
document.getElementById("selectid") //where "selectid" is the ID of the SELECT element on the page.
```

Option Object

The Option object represents an HTML <option> element.

Access an Option Object

You can access an <option> element by using getElementById().

Option Object Properties

| Property | Description |
|-----------------|---|
| defaultSelected | Returns the default value of the selected attribute |



| | |
|----------|---|
| disabled | Sets or returns whether an option is disabled, or not |
| form | Returns a reference to the form that contains the option |
| index | Sets or returns the index position of an option in a drop-down list |
| label | Sets or returns the value of the label attribute of an option in a drop-down list |
| selected | Sets or returns the selected state of an option |
| text | Sets or returns the text of an option |
| value | Sets or returns the value of an option to be sent to the server |

Code: Disable the third option (index 2) in a drop-down list and apply color as red to disabled index.

```
<html>
<body>
<select id="programs" size="5">
    <option>Computer Engineering</option>
    <option>Information Technology</option>
    <option>Chemical Engineering</option>
    <option>Electronics & TeleComm.</option>
</select>

<p>Click the button to disable the third option (index 2) in the dropdown list.</p>

<button onclick="myFunction()">Disable Option</button>

<script>
function myFunction()
{
    var x = document.getElementById("programs").options[2].disabled = true;
```



```
document.getElementById("programs").options[2].style.color = "red";
}
</script>
</body>
</html>
```

Output:



Click the button to disable the third option (index 2) in the dropdown list.

Disable Option



Click the button to disable the third option (index 2) in the dropdown list.

Disable Option

Code: Display index and text associated with that index.

```
<html>
<body>
Select a Program and click the button:
<select id="mySelect">
<option>Information Technology</option>
<option>Computer Engg</option>
<option>Civil Engg</option>
<option>Electronics and Telecommunication</option>
</select>
<button type="button" onclick="myFunction()">
Display index</button>
<script>
function myFunction()
{
    var x = document.getElementById("mySelect").selectedIndex;
    var y = document.getElementById("mySelect").options;
```

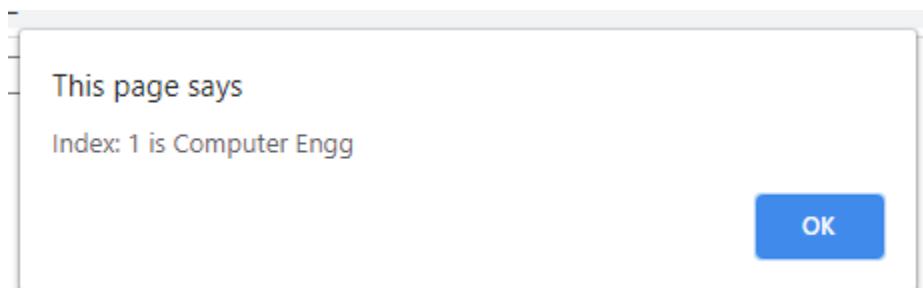


```
        alert("Index: " + y[x].index + " is " + y[x].text);
    }
</script>
</body>
</html>
```

Output:

Select a Program and click the button:

The screenshot shows a dropdown menu with the following options: Computer Engg, Information Technology, Computer Engg (highlighted in blue), Civil Engg, and Electronics and Telecommunication. To the right of the dropdown is a button labeled "Display index".



3.2 Form Events:

The form property within the document object contains an array of all forms defined within the document.

Each element within the array is a form object, the index number associated with the form object defines the order in which the form appears on the webpage.

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. When javascript code is included in HTML, javascript react over these events and allow the execution. This process of reacting over the events is called **Event Handling**. Thus, javascript handles the HTML events via Event Handlers.

For example, when a user clicks over the browser, add javascript code, which will execute the task to be performed on the event.

Table: Event handlers for Form Elements.



| Object | Event Handler |
|------------|--|
| button | <i>onClick, onBlur, onFocus</i> |
| checkbox | <i>onClick, onBlur, onFocus.</i> |
| FileUpLoad | <i>onClick, onBlur, onFocus</i> |
| hidden | <i>none</i> |
| password | <i>onBlur, onFocus, onSelect.</i> |
| radio | <i>onClick, onBlur, onFocus</i> |
| reset | <i>onReset</i> |
| select | <i>onFocus, onBlur, onChange.</i> |
| submit | <i>onSubmit</i> |
| text | <i>onClick, onBlur, onFocus , onChange</i> |
| textarea | <i>onClick, onBlur, onFocus , onChange</i> |

The main utility of a button object is to trigger an event, say an *onClick()* event, but a button object has no default action.

There are several types of buttons associated with a form:

- submit
- reset
- button

These events are fired when some click related activity is registered.

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 (The onblur event occurs when an object loses focus.) |
| change | onchange | When the user modifies or changes the value of a form element |

Code: onfocus event

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

Output:

Javascript Events

Enter something here



Code: onsubmit event

```
<html>
<body>

<p>When you submit the form, a function is triggered which alerts some text.</p>

<form action="" onsubmit="myFunction()">
Enter name: <input type="text" name="fname">
```



```
<input type="submit" value="Submit">  
</form>  
  
<script>  
function myFunction()  
{  
    alert("The form was submitted");  
}  
</script>  
</body>  
</html>
```

Output:

When you submit the form, a function is triggered which alerts some text.

Enter name:

This page says

The form was submitted

OK

Code: onblur event

Execute a JavaScript when a user leaves an input field, as soon as user leaves the input field, content in text box is appeared as in uppercase and color is blue.

```
<html>  
<body>
```

Enter your name: <input type="text" id="fname" onblur="myFunction()">

<p>When you leave the input field, a function is triggered which transforms the input text to upper case.</p>

```
<script>  
function myFunction()  
{  
    var x = document.getElementById("fname");  
    x.value = x.value.toUpperCase();  
    document.getElementById("fname").style.color="blue";  
}  
</script>  
</body>  
</html>
```

Output:



Enter your name: **YOGITA KHANDAGALE**

When you leave the input field, a function is triggered which transforms the input text to upper case.

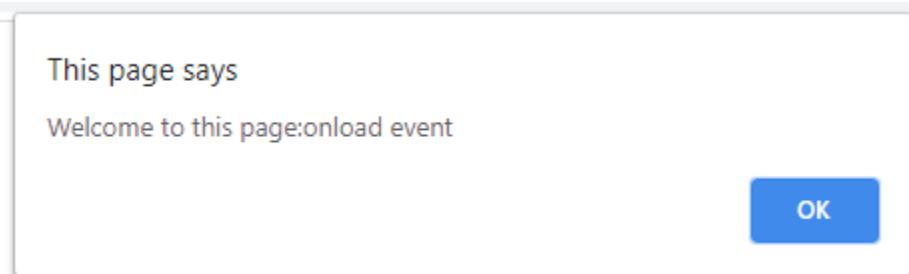
Window/Document events:

| 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 |

Code: onload event

```
<html>
<head>
    <script type="text/javascript">
        function message() {
            alert("Welcome to this page:onload event");
        }
    </script>
</head>
<body onload="message()">
When page loaded alert is displayed.
</body>
</html>
```

Output:



This page says
Welcome to this page:onload event

OK



Code: onresize event

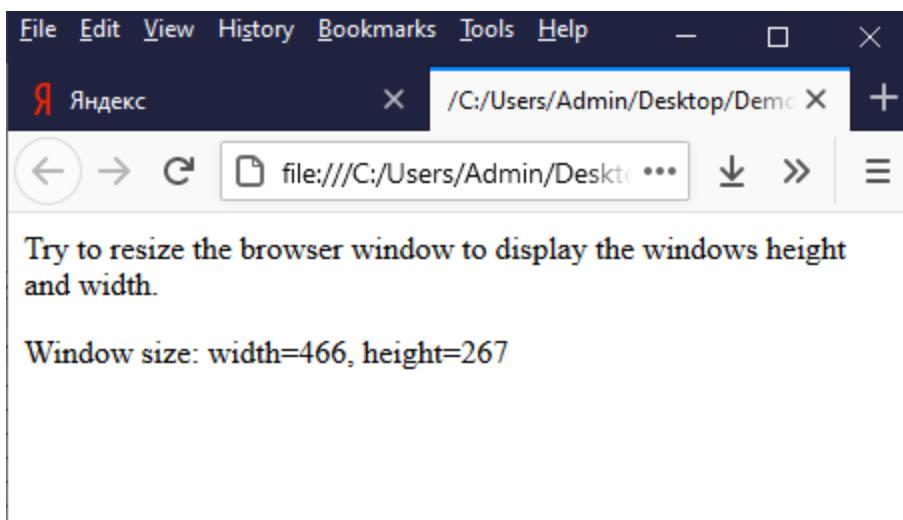
```
<html>
<body onresize="myFunction()>

<p>Try to resize the browser window to display the windows height and width.</p>

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

<script>
function myFunction()
{
    var w = window.outerWidth;
    var h = window.outerHeight;
    var txt = "Window size: width=" + w + ", height=" + h;
    document.getElementById("demo").innerHTML = txt;
}
</script>
</body>
</html>
```

Output:



Code: onclick event

```
d
```

Output:

Hello in Different Countries

[Japan](#) [India](#) [Germany](#)

This page says

namaste

OK

Mouse Events:

| Attribute | Value | Description |
|---------------|--------|---|
| onclick | script | Fires on a mouse click on the element |
| ondblclick | script | Fires on a mouse double-click on the element |
| onmousedown | script | Fires when a mouse button is pressed down on an element |
| onmousemove | script | Fires when the mouse pointer is moving while it is over an element |
| onmouseout | script | Fires when the mouse pointer moves out of an element |
| onmouseover | script | Fires when the mouse pointer moves over an element |
| onmouseup | script | Fires when a mouse button is released over an element |
| onwheel | script | Fires when the mouse wheel rolls up or down over an element |
| oncontextmenu | script | oncontextmenu event occurs when the user right-clicks on an element to open the context menu. |

Code: onmouseout and onmouseover event

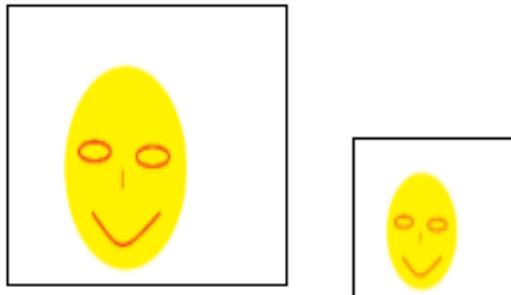
```
<html>
```



```
<html>
<body>

<script>
function bigImg(x)
{
    x.style.height = "120px";
    x.style.width = "120px";
}
function normalImg(x)
{
    x.style.height = "64px";
    x.style.width = "64px";
}
</script>
</body>
</html>
```

Output:



Code: onwheel event

When you roll the mouse over the paragraph either up or down, paragraph text will be increased to 35 pixels.

```
<html>
<body>

<div id="aa" onwheel="myFunction()">
```

This example demonstrates how to assign an "onwheel" event event to a DIV element.
Roll the mouse wheel over me - either up or down!</div>



```
<script>
function myFunction()
{
  document.getElementById("aa").style.fontSize = "35px";
}
</script>
</body>
</html>
```

Output:

This example demonstrates how to assign an "onwheel" event event to a DIV element. Roll the mouse wheel over me - either up or down!

This example demonstrates how to assign an "onwheel" event event to a DIV element. Roll the mouse wheel over me - either up or down!

Code: ondblclick event

```
<html>
<body>
<h2>
<p id="demo" ondblclick="color_change()">Double-click me to change my text color.</p> </h2>

<script>
function color_change()
{
  document.getElementById("demo").style.color = "red";
  document.getElementById("demo").style.backgroundColor = "yellow";
}
</script>

</body>
</html>
```

Output:

Double-click me to change my text color.



Double-click me to change my text color.

Code: onmousedown and onmouseup event

```
<html>
<body>

<p id="p1" onmousedown="mouseDown()" onmouseup="mouseUp()">
Click the text! The mouseDown() function is triggered when the mouse button is
pressed down over this paragraph. The function sets the color of the text to red. The
mouseUp() function is triggered when the mouse button is released. The mouseUp()
function sets the color of the text to blue.
</p>
<script>
function mouseDown()
{
    document.getElementById("p1").style.color = "red";
}
function mouseUp()
{
    document.getElementById("p1").style.color = "blue";
}
</script>
</body>
</html>
```

Output:

Click the text! The mouseDown() function is triggered when the mouse button is pressed down over this paragraph. The function sets the color of the text to red. The mouseUp() function is triggered when the mouse button is released. The mouseUp() function sets the color of the text to blue.

Click the text! The mouseDown() function is triggered when the mouse button is pressed down over this paragraph. The function sets the color of the text to red. The mouseUp() function is triggered when the mouse button is released. The mouseUp() function sets the color of the text to blue.

Code: oncontextmenu event

Execute a JavaScript when the user right-clicks on a <div> element with a context menu:



```
<html>
<head>
<style>
div {
    background: yellow;
    border: 1px solid black;
    padding: 10px;
}
</style>
</head>
<body>
<div oncontextmenu="myFunction()" contextmenu="mymenu">
<p>Right-click inside this box to see the context menu!
</div>
<script>
function myFunction()
{
    alert("You right-clicked inside the div!");
}
</script>
</body>
</html>
```

Output:

The screenshot shows a web page with a yellow rectangular div element. Inside the div, the text "Right-click inside this box to see the context menu!" is displayed. A context menu, titled "mymenu", is open over the div. The menu contains a single item: "You right-clicked inside the div!". At the bottom of the menu is an "OK" button.

Code: onmousemove event



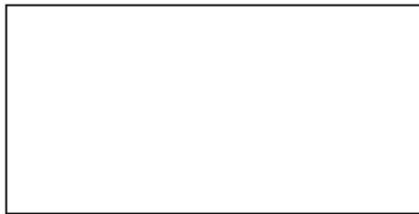
Execute a JavaScript when moving the mouse pointer over a <div> element and display the x and y coordinates.

```
<html>
<head>
<style>
div {
    width: 200px;
    height: 100px;
    border: 1px solid black;
}
</style>
</head>
<body>
<p>This example demonstrates how to assign an "onmousemove" event to a div element.</p>
<div onmousemove="myFunction(event)"></div>
<p>Mouse over the rectangle above, and get the coordinates of your mouse pointer.</p>
<p id="demo"></p>
<script>
function myFunction(e)
{
    var x = e.clientX;
    var y = e.clientY;
    var coor = "Coordinates: (" + x + "," + y + ")";
    document.getElementById("demo").innerHTML = coor;
}
</script>
</body>
</html>
```

Output:



This example demonstrates how to assign an "onmousemove" event to a div element.



Mouse over the rectangle above, and get the coordinates of your mouse pointer.

Coordinates: (64,91)

```
<!DOCTYPE HTML>
<html>
<head>
<title>Example: Working with Mouse events</title>
<style>
body{font-family:Verdana;
background:#44c767; color:#fff;}
</style>
<script>
var count = 0;
function tracker(){
count++;
alert(count + " Mouse moves have been registered");
}
function popup1() {
alert("Event Registered : onMouseOver");
}
function popup2() {
alert("Event Registered : onMouseOut");
}
```



```
</script>
</head>
<body>
<p>Move the mouse over the link to trigger the event
<a href="#" onmouseover="popup1()"> onMouseOver</a></p>

<p>Move the mouse over the link to trigger the event
<a href="#" onmouseout="popup2()"> onMouseOver</a></p>

<p>Move the mouse over the button, it keeps a track of number
of times the mouse moves over the button</p>
<button onmouseover="tracker()">Move over this button </button>
</body>
</html>
```

```
<!DOCTYPE HTML>
<html>
<head><title>Javascript Events: Attach an Event Listener</title>
</head>
<body>
<p id="content">We must let go of the life we have planned,
so as to accept the one that is waiting for us </p>
</body>
<script>
var p = document.getElementById("content");
p.addEventListener('click',change); //event Listener attached
```



```
// false denotes that bubbling method of event propagation

function change(){
    this.style.background = "#FA8B7C";
    this.style.fontFamily = "Verdana";
    this.style.padding = "10px";
    this.style.color = "#fff";
    this.style.border = "4px dotted green";
}

</script>

</html>
```

http://www.tutorialspark.com/javascript/JavaScript_Mouse_Event.php

KeyEvent:

These event types belong to the KeyboardEvent Object:

| Event | Description |
|-----------------|--|
| <u>keydown</u> | The event occurs when the user is pressing a key |
| <u>keypress</u> | The event occurs when the user presses a key |
| <u>keyup</u> | The event occurs when the user releases a key |

- 1) The **onkeydown** event occurs when the user is pressing a key (on the keyboard).

Syntax: <element onkeydown="myScript">

The keydown event occurs when the user presses down a key on the keyboard. You can handle the keydown event with the onkeydown event handler. The following example will show you an alert message when the keydown event occurs.

[Code: onkeydown event](#)



k

Output:

Note: Try to enter some text inside input box and textarea.

This page says

You have pressed a key inside text input!

OK

a

aa

Note: Try to enter some text inside input box and textarea.

2) The **onkeyup** event occurs when the user releases a key (on the keyboard).

Syntax: <element onkeyup="myScript">

You can handle the keyup event with the onkeyup event handler.

The following example will show you an alert message when the keyup event occurs.

Code: onkeyup event

```
<html>
<body>
    <input type="text" onkeyup="alert('You have released a key inside text input!')">
    <hr>
        <textarea cols="30" onkeyup="alert('You have released a key inside textarea!')"></textarea>
    <p><strong>Note:</strong> Try to enter some text inside input box and textarea.</p>
</body>
</html>
```

Output:

Note: Try to enter some text inside input box and textarea.

This page says

You have released a key inside textarea!

OK

3) The **onkeypress** event occurs when the user presses a key (on the keyboard).



Syntax: <element onkeypress="myScript">

The keypress event occurs when a user presses down a key on the keyboard that has a character value associated with it. For example, keys like Ctrl, Shift, Alt, Esc, Arrow keys, etc. will not generate a keypress event, but will generate a keydown and keyup event.

You can handle the keypress event with the `onkeypress` event handler.

The following example will show you an alert message when the keypress event occurs.

[Code: onkeypress event](#)

```
<html>
<body>
<p>Press a key inside the text field to set a red background color.</p>

<input type="text" id="demo" onkeypress="myFunction()">

<script>
function myFunction()
{ document.getElementById("demo").style.backgroundColor = "red";
}
</script>
</body>
</html>
```

[Output:](#)

Press a key inside the text field to set a red background color.

Press a key inside the text field to set a red background color.

 V

In JavaScript, using the `addEventListener()` method, you can handle an event:

Syntax: `object.addEventListener("name_of_event", myScript);`

[Code: addEventListener\(\)](#)

```
<html>
<body>
```



```
<p>Press a key inside the text field to set a red background color.</p>
```

```
<input type="text" id="demo">
<script>
document.getElementById("demo").addEventListener("keypress", myFunction);

function myFunction()
{
    document.getElementById("demo").style.backgroundColor = "red";
}
</script>
</body>
</html>
```

Output:

Press a key inside the text field to set a red background color.

A

```
<!DOCTYPE html>
<html>
<head>
<title>Javascript Mouse Events</title>
<style>
#target-div {
    width: 320px;
    height: 150px;
    background: blue;
    margin-bottom: 10px;
}
</style>
<script>
```



```
function clickHandler(evt) {  
    var html = "evt.altKey=" + evt.altKey;  
  
    document.getElementById("log-div").innerHTML = html;  
}  
</script>  
</head>  
<body>  
    <h3>Press Ctrl key and Click</h3>  
    <div id="target-div" onclick="clickHandler(event)">  
    </div>  
    <div style="color:red;" id="log-div">  
    </div>  
</body>  
</html>
```

3.3 Changing an attribute value dynamically:

The change in any attribute value can be reflected to the user by highlighting the value or text by some color.

The onchange event is associated with many elements `<input>`, `<select>` of a form object and helpful to make call to a function where the change of attribute value code is written.

In following example onchange event is used with two textboxes and whenever user will make change in value of these textboxes, text color and background of text boxes will change.

Code: onchange event to change text color as blue and background color is pink.

```
<html>  
    <head>  
        <script type="text/javascript">  
            function highlight(x)  
            {  
                x.style.color="blue";  
                x.style.backgroundColor="pink";  
            }  
        </script>
```



```
</head>
<body>
<form name="myform" action=" " method="post">
Institute Name:
<input type="text" name="iname" onchange="highlight(this)"/>
<br>
Program:
<input type="text" name="infotech" onchange="highlight(this)"/>
<br>
<input type="submit" value="submit" name="submit">
</form>
</body>
</html>
```

Output:

Institute Name: **Vidyalankar**

Program: **Information Technology**

submit

Code: onchange event to change the text in text box.

```
<html>
<body>
Enter some text:
<input type="text" name="txt" value="Hello" onchange="myFunction(this.value)">

<script>
function myFunction(val)
{
    alert("The input value has changed. The new value is: " + val);
}
</script>
</body>
</html>
```

Output:

Enter some text:

This page says

The input value has changed. The new value is: Hello VP

OK

“with” keyword

The with keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to with becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax:

```
with (object)
{
}
```

Properties used without the object name and dot

```
}
```

Example:

| Without using “with” keyword | Use of “with” keyword |
|---|---|
| <pre><html> <body> <h2>JavaScript Math functions without using "with"</h2> <script> var r = 10; { a = (Math.PI) * r * r; x = r * Math.cos(Math.PI); y = r * Math.sin(Math.PI / 2); z=Math.sqrt(16); } document.write("a="+a+" "); document.write("x="+x+" "); document.write("y="+y+" "); document.write("z="+z+" "); </script> </body> </html></pre> | <pre><html> <body> <h2>JavaScript Math functions without using "with"</h2> <script> var r = 10; with (Math) { a = PI * r * r; x = r * cos(PI); y = r * sin(PI / 2); z=sqrt(16); } document.write("a="+a+" "); document.write("x="+x+" "); document.write("y="+y+" "); document.write("z="+z+" "); </script> </body> </html></pre> |



| | |
|---|---|
| Output: JavaScript Math functions without using "with" a=314.1592653589793 x=-10 y=10 z=4 | Output: JavaScript Math functions using "with" a=314.1592653589793 x=-10 y=10 z=4 |
|---|---|

3.4 Changing option list dynamically:

Code: Following example provides two radio buttons to the user one is for fruits and another is for vegetables.

When user will select the fruits radio button, the option list should present only the fruits names to user and when user will select the vegetable radio button, the option list should present only the vegetable names to user so that user can select an appropriate element of interest.

```
<html>
<body>
<html>
<script type="text/javascript">
function modifyList(x)
{
with(document.forms.myform)
{
if(x ==1)
{
optionList[0].text="Kiwi";
optionList[0].value=1;
optionList[1].text="Pine-Apple ";
optionList[1].value=2;
optionList[2].text="Apple";
optionList[2].value=3;
}

if(x ==2)
{
```



```
optionList[0].text="Tomato";
optionList[0].value=1;
optionList[1].text="Onion ";
optionList[1].value=2;
optionList[2].text="Cabbage ";
optionList[2].value=3;
}

}

}

</script>
</head>
<body>
<form name="myform" action=" " method="post">
<select name="optionList" size="3">
<option value=1>Kiwi
<option value=1>Pine-Apple
<option value=1>Apple
</select>
<br>
<input type="radio" name="grp1" value=1 checked="true" onclick="modifyList(this.value)">
Fruits

<input type="radio" name="grp1" value=2 onclick="modifyList(this.value)"> Vegetables
</form>
</body>
</html>
```

Output:

The screenshot shows a user interface with two dropdown menus and two radio buttons. On the left, a dropdown menu displays three items: 'Kiwi', 'Pine-Apple', and 'Apple'. The 'Apple' option is highlighted with a blue selection bar. Below this dropdown are two radio buttons: one labeled 'Fruits' and another labeled 'Vegetables'. The 'Fruits' button is selected, indicated by a blue circle with a dot. On the right, another dropdown menu displays three items: 'Tomato', 'Onion', and 'Cabbage'. The 'Cabbage' option is highlighted with a blue selection bar. Below this dropdown are two radio buttons: one labeled 'Fruits' and another labeled 'Vegetables'. The 'Vegetables' button is selected, indicated by a blue circle with a dot.

Code: Following example provides four list elements as name of branches. When you select a branch from list, selected branch will be displayed as output.

```
<html>
<body>
```



```
<p>Select Program from list:</p>
<select id="mySelect" onchange="myFunction()">
    <option value="CO">Computer Engg</option>
    <option value="IF">Information Technology</option>
    <option value="EJ">Electronics and Tele</option>
    <option value="CE">Chemical Engg</option>
</select>
<p id="demo"></p>
<script>
function myFunction()
{
    var x = document.getElementById("mySelect").value;
    document.getElementById("demo").innerHTML = "You selected: " + x;
}
</script>
</body>
</html>
```

Output:

Select Program from list:

Information Technology ▾

You selected: IF

3.5 Evaluating check box selections

A checkbox is created by using the input element with the type="checkbox" attribute-value pair.

A checkbox in a form has only two states (checked or un-checked). Checkboxes can be grouped together under a common name.

Code: Following example make use of five checkboxes to provide five options to the user regarding favorite color. After the selection of favorite colors, all selected color names are displayed as output.

```
<html>
<head>
    <title>Print value of all checked CheckBoxes on Button click.</title>
    <script type="text/javascript">
        function printChecked()
        {
```



```
var items=document.getElementsByName('check_print');
var selectedItems="";
for(var i=0; i<items.length; i++)
{
    if(items[i].type=='checkbox' && items[i].checked==true)
        selectedItems+=items[i].value+"<br>";
}
document.getElementById("y").innerHTML =selectedItems;
}

</script>
</head>
<body>
<big>Select your favourite accessories: </big><br>
<input type="checkbox" name="check_print" value="red">red<br>
<input type="checkbox" name="check_print" value="Blue">Blue<br>
<input type="checkbox" name="check_print" value="Green">Green<br>
<input type="checkbox" name="check_print" value="Yellow">Yellow<br>
<input type="checkbox" name="check_print" value="Orange">Orange<br>
<p><input type="button" onclick='printChecked()' value="Click me"/></p>
    You Selected:
    <p id="y"></p>
</body>
</html>
```

Output:

Select your favourite accessories:

- red
- Blue
- Green
- Yellow
- Orange

Click me

You Selected:

Blue
Green
Orange

3.6 Changing labels dynamically

What is a label?

The `<label>` tag is used to provide a usability improvement for mouse users i.e, if a user clicks on the text within the `<label>` element, it toggles the control.



Approach:

- Create a label element and assign an id to that element.
- Define a button that is used to call a function. It acts as a switch to change the text in the label element.
- Define a JavaScript function, that will update the label text.
- Use the innerHTML property to change the text inside the label.
The innerHTML property sets or returns the HTML content of an element.

Code: Given an HTML document and the task is to change the text and color of a label using JavaScript.

```
<html>
<head>
</head>
<body style="text-align:center;">
<h1 style="color:green;">
Client-SideScripting
</h1>
<h4>
Click on the button to change the text of a label
</h4>
<label id = "aaa">
Welcome to Client-Side Scripting Course.
</label>
<br>
<button onclick="change_L()">
    Click Here!
</button>

<script>
    function change_L()
    {
        document.getElementById('aaa').innerHTML
            = "CSS is a client-side scripting language.";
        document.getElementById('aaa').style.color
            = "red";
    }
</script>
</body>
</html>
```



Output:

Client-SideScripting

Click on the button to change the text of a label

Welcome to Client-Side Scripting Course.

[Click Here!](#)

Client-SideScripting

Click on the button to change the text of a label

CSS is a client-side scripting language.

[Click Here!](#)

3.7 Manipulating form elements

Javascript make it possible with help of hidden element which is similar to any html element except it does not appear on screen.

Code: Following example is displaying the text of hidden text box after clicking on submit button.

```
<html>
<head>
<title>
    HTML Input Hidden value Property
</title>
</head>
<body style="text-align:center;">
    <h1 style="color:green;">
        Vidyalankar Polytechnic
    </h1>
    <h2>Input Hidden value Property</h2>
    <input type="hidden" id="it"
        value="Information Technology">
    <button onclick="disp_hidden_Text()">
        Submit
    </button>
    <p id="demo" style="color:green;font-size:35px;"></p>
    <script>
        function disp_hidden_Text()
```



```
{  
    var x = document.getElementById("it").value;  
    document.getElementById("demo").innerHTML = x;  
}  
</script>  
</body>  
</html>
```

Output:

Vidyalankar Polytechnic

Input Hidden value Property

Information Technology

3.8 Intrinsic javascript functions

The HTML <input> src Attribute is used to *specify the URL of the image to be used as a submit Button*. This attribute is not used with <input type="image">

Syntax:

```
<input src="URL">
```

Attribute Values: It contains a single value URL which specifies the link of source image. There are two types of URL link which are listed below:

- Absolute URL: It points to another webpage.
- Relative URL: It points to other files of the same web page.

Code: Following example we have used one tag to simulate the functionality of submit button. Before writing the code make sure one “submit.jpg” picture should save in your folder.

```
<html>  
<body>  
<h1>The input src attribute</h1>  
  
<form action=" ">  
  <label for="fname">Institute Name:</label>  
  <input type="text" id="name" name="name"><br><br>
```



```
<input type="image" src="submit.jpg" alt="Submit" width="130" height="48"
onclick="myFunction()">
</form>
<p id="demo"></p>
<script>
function myFunction()
{
    var x = document.getElementById("name").value;
    document.write("You Submitted:<h2>" +x + "</h2>");
}
</script>
</body>
</html>
```

Output:

The input src attribute

Institute Name:

You Submitted:



Vidyalanakr Polytechnic

Disabling Elements:

It is common to display a form with some elements disabled, which prevents the user from entering information into the element.

Code: Following example shows to enable and disable text field.

```
<html>
<body>
Name: <input type="text" id="myText">
<p>Click the button to enable/disable the text field.</p>
<button onclick="myFunction()">
change status
</button>
<script>
function myFunction()
{
var txt=document.getElementById("myText")
if ('disabled' in txt)
```



```
{  
txt.disabled=!txt.disabled;  
}  
}  
</script>  
</body>  
</html>
```

Output:

Name:

Click the button to enable/disable the text field.

Name:

Click the button to enable/disable the text field.

OR

```
<html>  
<body>  
  
First Name: <input type="text" id="myText"><br>  
<br>  
<button onclick="disableTxt()">Disable Text field</button>  
<button onclick="undisableTxt()">Undisable Text field</button>  
  
<script>  
function disableTxt()  
{  
    document.getElementById("myText").disabled = true;  
}  
  
function undisableTxt()  
{  
    document.getElementById("myText").disabled = false;  
}  
</script>  
</body>  
</html>
```

Output:

First Name: First Name:

Read only elements:

The `readOnly` property sets or returns whether a text field is read-only, or not. A read-only field cannot be modified. However, a user can tab to it, highlight it, and copy the text from it.

Set a text field to read-only:

```
document.getElementById("myText").readOnly = true;
```

Syntax:

To return the `readOnly` property: `textObject.readOnly`

To Set the `readOnly` property: `textObject.readOnly = true|false`

Code: Following example illustrate the use of read only property.

When user clicks on “Click here” button, text box is disabled.

```
<html>
<body>

Name: <input type="text" id="myText" value="VP">

<p>Click the button to set the text field to read-only.</p>

<p><strong>Tip:</strong> To see the effect, try to type something in the text field before and after clicking the button.</p>

<button onclick="myFunction()">Click here </button>

<script>
function myFunction()
{
    document.getElementById("myText").readOnly = true;
}
</script>

</body>
</html>
```



V2V EDTECH LLP

DIPLOMA | DEGREE | BSCIT

Output:

Name:

Click the button to set the text field to read-only.

Tip: To see the effect, try to type something in the text field before and after clicking the button.



Unit 4

Cookies and Browser Data

4.1 Cookies- Basics of Cookies, reading a cookie value, writing a cookie value, creating Cookies, deleting a cookie, Setting the expiration Date of Cookies

4.2 Browser-Opening a window, scrolling new window focus, window position, changing the content of window, closing a window, scrolling a web page, multiple windows at once, creating a web page in a new window, javascript in URLs, javascript security, Timers, Browser location and history.



4.1 Cookies:

4.1.1 Basics of Cookies

A cookie is an amount of information that persists between a server-side and a client-side.

Cookies originally designed for server-side programming.

A web browser stores this information at the time of browsing.

A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the user's information among all the web pages.

Why do you need a Cookie?

The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

Types of cookies

All cookies are not created equal. There are 3 types of them:

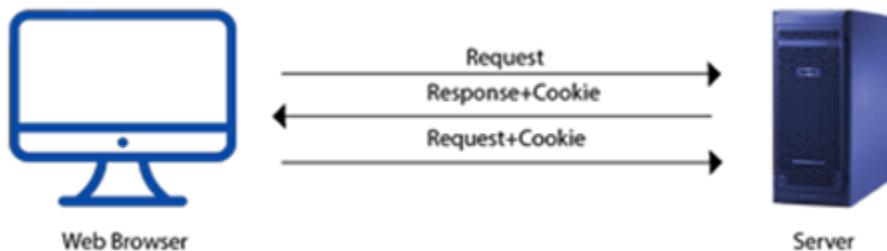
- Session: They expire when you close your browser (or if you stay inactive for a certain time). They're used for example on e-commerce websites so you can continue browsing without losing what you put in your cart.
- Permanent: They persist even when the browser is closed. They have an expiration date though and by law, you can't make them last more than 6 months. They're used to remember your passwords and login info so you don't have to re-enter them every time.
- Third-party: Cookies attributes usually corresponds to the website domain they are on. Not for third-party cookies—as you probably gathered from the name, they are installed by ... third-party websites (no wayy), such as advertisers. They gather data about your browsing habits, and allow them to track you across multiple websites. Other websites using third-party cookies: Facebook, Flickr, Google Analytics, Google Maps, Google Plus, SoundCloud, Tumblr, Twitter and YouTube.

How Cookies Works?

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
- So, to recognize the old user, we need to add the cookie with the response from the server browser at the client-side.



- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.



4.1.2 Read a Cookie with JavaScript

With JavaScript, cookies can be read like this:

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

4.1.3 Write a Cookie with JavaScript

You can access the cookie like this which will return all the cookies saved for the current domain.

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

Change a Cookie with JavaScript

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

```
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

Note: All cookies example should execute on Mozilla browser.

Code: to set and get cookies on onclick event.

```
<html>
<head>
<script>
function writeCookie()
{
with(document.myform)
{
document.cookie="Name=" + person.value + ";"
alert("Cookie written");
}
}
```



```
}
```

```
function readCookie()
{
var x;
if(document.cookie=="")
x="";
else
x=document.cookie;
document.write(x);
}
</script>
</head>
<body>
<form name="myform" action="">
Enter your name:  

<input type="text" name="person"><br>
<input type="Reset" value="Set Cookie" type="button" onclick="writeCookie()">
<input type="Reset" value="Get Cookie" type="button" onclick="readCookie()">
</form>
</body>
</html>
```

Output:

Enter your name: Information Technology

Cookie written

OK

After Clicking on Get Cookie,



Name=Information Technology

4.1.4 Creating a cookie

In JavaScript, we can create, read, update and delete a cookie by using **document.cookie** property.

The following syntax is used to create a cookie:

```
document.cookie = "name=value";
```

You can also add an expiry date (in GMT time).

By default, the cookie is deleted when the browser is closed:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 GMT";
```

With a path parameter, you can tell the browser what path the cookie belongs to.

By default, the cookie belongs to the current page.

```
document.cookie = "username=Chirag Shetty; expires=Thu, 18 Dec 2013 12:00:00 GMT; path=/";
```

4.1.5 Delete a Cookie with JavaScript

Deleting a cookie is very simple.

You don't have to specify a cookie value when you delete a cookie.

Just set the expires parameter to a passed date:

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

Code: To delete cookies

```
<html>
<head>
</head>
<body>
<input type="button" value="setCookie" onclick="setCookie()">
<input type="button" value="getCookie" onclick="getCookie()">
<script>
function setCookie()
```



```
{  
    document.cookie="username=Vidyalanakr Polytechnic; expires=Mon, 3 Aug 2020  
00:00:00 GMT";  
}  
function getCookie()  
{  
    if(document.cookie.length!=0)  
    {  
        var array=document.cookie.split("=");  
        alert("Name="+array[0]+" "+ "Value="+array[1]);  
    }  
    else  
    {  
        alert("Cookie not available");  
    }  
}  
</script>  
</body>  
</html>
```

Output:



4.1.6 Setting the Expiration Date of Cookie

Cookies are transient by default; the values they store last for the duration of the web browser session but are lost when the user exits the browser.

User can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie.



For example,

```
document.cookie="username=VP; expires=Tues,04 Aug 2020 00:00:00 GMT";
```

Code:

```
<html>
<head>
<script>
function writeCookie()
{
var d=new Date();
d.setTime(d.getTime()+(1000*60*60*24));
with(document.myform)
{
document.cookie="Name=" + person.value + ";expires=" +d.toGMTString();
}
}
function readCookie()
{
if(document.cookie=="")
document.write("cookies not found");
else
document.write(document.cookie);
}
</script>
</head>
<body>
<form name="myform" action="">
Enter your name:
<input type="text" name="person"><br>
<input type="Reset" value="Set C" type="button" onclick="writeCookie()">
<input type="Reset" value="Get C" type="button" onclick="readCookie()">
</form>
</body>
</html>
```

Output:



Enter your name:

Click on “Get Cookie”

cookies not found

Enter your name in text box and click on “Set Cookie”

Enter your name:

After clicking on “Get Cookie”

Name=Information Technology

Code:

```
<html>
<head>
<script>
document.cookie="username=VP; expires=Tues,04 Aug 2020 00:00:00 GMT ";
if(document.cookie)
{
    document.write("created cookie is:" +document.cookie);
    cstr=document.cookie
    var list=cstr.split("=");
    document.write("<br> Split List:" +list);
    if(list[0]=="username")
    {
        var data=list[1].split(",");
        document.write("<br> Data:" +data);
        var data=list[2].split(",");
        document.write("<br> Data:" +data);
        var data=list[3].split(",");
        document.write("<br> Data:" +data);
    }
}
```



```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

created cookie is:username=VP

Split List:username,VP

Data:VP

4.2 Browser

A **web browser** (commonly referred to as a **browser**) is a **software application** for retrieving, presenting and traversing information resources on the World Wide Web. An *information resource* is identified by a Uniform Resource Identifier/ Locator (URI/URL) and may be a web page, image, video or other piece of content. Hyperlinks present in resources enable users easily to navigate their browsers to related resources.

Although browsers are primarily intended to use the World Wide Web, they can also be used to access information provided by web servers in private networks or files in file systems.

The major web browsers are Firefox, Internet Explorer, Google Chrome, Opera, and Safari.

Browser's Components:

The browser's main components are:

1. The user interface:

This includes the address bar, back/forward button, bookmarking menu, etc. Every part of the browser displays except the window where you see the requested page.

2. The browser engine:

marshals' actions between the UI and the rendering engine.

3. The rendering engine:

responsible for displaying requested content. For example, if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.

4. Networking:

For network calls such as HTTP requests, using different implementations for different platform behind a platform-independent interface.

5. UI backend:



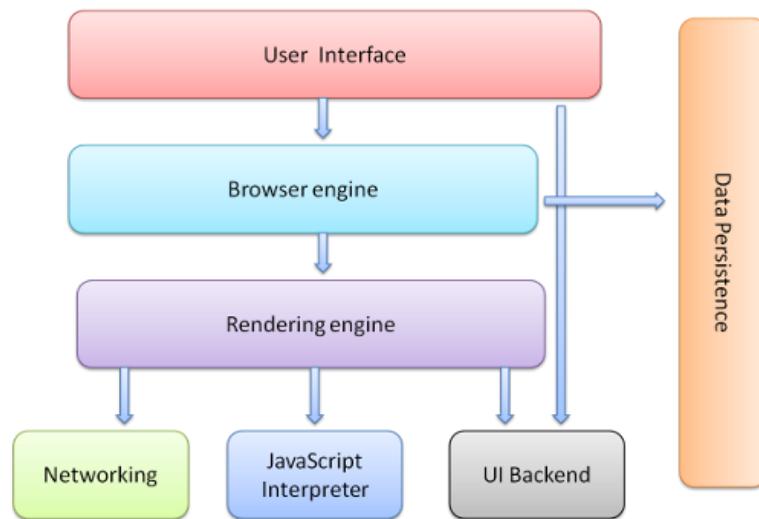
Used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.

6. JavaScript interpreter.

Used to parse and execute JavaScript code.

7. Data storage.

This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and Filesystem.



It is important to note that browsers such as Chrome run multiple instances of the rendering engine: one for each tab. Each tab runs in a separate process.

Document object Model (DOM)

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

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

As mentioned earlier, it is the object of window.

So

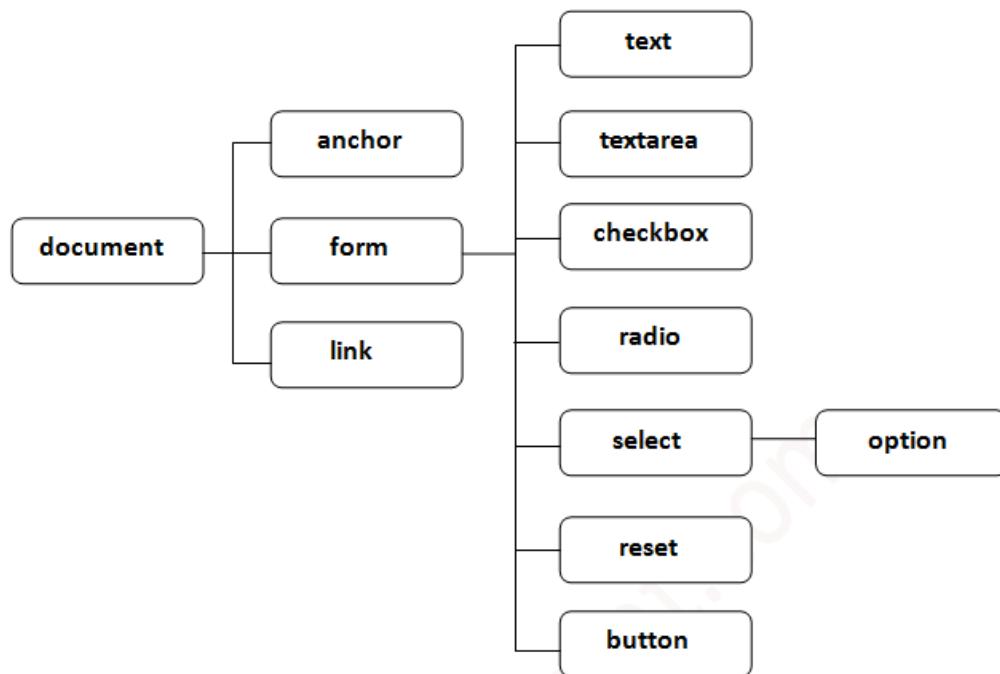
```
window.document
```

is same as

```
document
```

Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



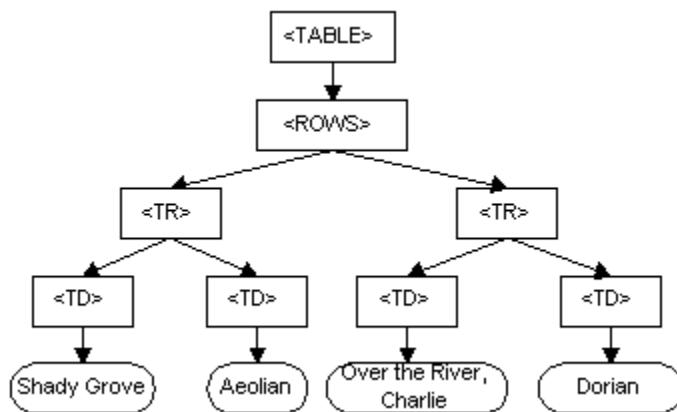
Source: <https://www.javatpoint.com/document-object-model>

The Document Object Model is a programming API for documents. The object model itself closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:



```
<TABLE>
<ROWS>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</ROWS>
</TABLE>
```

The Document Object Model represents this table like this:



DOM representation of the example table

Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

| Method | Description |
|-------------------|--|
| write("string") | writes the given string on the document. |
| writeln("string") | writes the given string on the document with newline character at the end. |
| getElementById() | returns the element having the given id value. |



| | |
|--------------------------|---|
| getElementsByName() | returns all the elements having the given name value. |
| getElementsByTagName() | returns all the elements having the given tag name. |
| getElementsByClassName() | returns all the elements having the given class name |

The `getElementsByClassName()` method returns a collection of all elements in the document with the specified class name, as an `HTMLCollection` object.

Syntax: `document.getElementsByClassName(classname);`

Code: Find out how many elements with class="example" there are in the document (using the `length` property of the `HTMLCollection` object):

```
<html>
<body>
<div class="example">
A div element with class="example"
</div>

<div class="example">
Another div element with class="example"
</div>

<p class="example">A p element with class="example"</p>

<p>Click the button to find out how many elements with class "example" there are in this document.</p>

<button onclick="myFunction()">Try it</button>

<p><strong>Note:</strong> The getElementsByClassName() method is not supported in Internet Explorer 8 and earlier versions.</p>

<p id="demo"></p>
<script>
function myFunction()
{
    var x = document.getElementsByClassName("example");
    document.getElementById("demo").innerHTML = x.length;
```



```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

A div element with class="example"

Another div element with class="example"

A p element with class="example"

Click the button to find out how many elements with class "example" there are in this document.

[Try it](#)

Note: The `getElementsByClassName()` method is not supported in Internet Explorer 8 and earlier versions.

3

Code: Change the background color of all elements with class="example":

```
<html>
<head>
<style>
.example {
    border: 1px solid black;
    margin: 5px;
}
</style>
</head>
<body>
<div class="example">
A div with class="example"
</div>

<div class="example">
Another div with class="example"
</div>
```



```
<p class="example">This is a p element with class="example".</p>

<p>This is a <span class="example">span</span> element with class="example" inside another p element.</p>

<p>Click the button to change the background color of all elements with class="example".</p>

<button class="example" onclick="myFunction()">Try it</button>

<p><strong>Note:</strong> The getElementsByClassName() method is not supported in Internet Explorer 8 and earlier versions.</p>

<script>
function myFunction()
{
    var x = document.getElementsByClassName("example");
    var i;
    for (i = 0; i < x.length; i++)
    {
        x[i].style.backgroundColor = "red";
    }
}
</script>
</body>
</html>
```

Output:



A div with class="example"

Another div with class="example"

This is a p element with class="example".

This is a element with class="example" inside another p element.

Click the button to change the background color of all elements with class="example".

Try it

Note: The `getElementsByClassName()` method is not supported in Internet Explorer 8 and earlier versions.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using `document.form1.name.value` to get the value of name field.

Here, `document` is the root element that represents the html document.

`form1` is the name of the form.

`name` is the attribute name of the input text.

`value` is the property, that returns the value of the input text.

Code: Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue()
{
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
<input type="button" onclick="printvalue()" value="print name"/>
</form>
```

Output:

Enter Name:

This page says

Welcome: Vidyalankar Polyetchnic

Javascript – innerText

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Code: In this example, we are going to display the password strength when releases the key after press.

```
<script type="text/javascript" >
function validate() {
var msg;
if(document.myForm.userPass.value.length>5){
msg="good";
}
else{
msg="poor";
}
document.getElementById('mylocation').innerText=msg;
}
</script>
<form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="mylocation">no strength</span>
</form>
```

Output:

Strength:good

Strength:poor

Application of DOM:

- 1) To render a document such as HTML page, most web browsers use an internal model similar to DOM. The nodes of every document are organized in a tree structure, called DOM tree, with topmost node named as “Document Object”. When HTML page is rendered in browser, the



browser downloads the HTML page into local memory and automatically parses it to display the page on screen.

2) When a web page is loaded, the browser creates a Document Object Model of the page, which is an object-oriented representation of an HTML document, that's act as an interface between javascript and document itself and allows the creation of dynamic web pages.

Window object

- Window object is a top-level object in Client-Side JavaScript.
- Window object represents the browser's window.
- It represents an open window in a browser.
- It supports all browsers.

The document object is a property of the window object. So, typing `window.document.write` is same as `document.write`.

- All global variables are properties and functions are methods of the window object.

Window Object Properties

| Property | Description |
|-------------|--|
| Document | It returns the document object for the window (DOM). |
| Frames | It returns an array of all the frames including iframes in the current window. |
| Closed | It returns the Boolean value indicating whether a window has been closed or not. |
| History | It returns the history object for the window. |
| innerHeight | It sets or returns the inner height of a window's content area. |
| innerWidth | It sets or returns the inner width of a window's content area. |
| Length | It returns the number of frames in a window. |
| Location | It returns the location object for the window. |
| Name | It sets or returns the name of a window. |
| Navigator | It returns the navigator object for the window. |
| Opener | It returns a reference to the window that created the window. |
| outerHeight | It sets or returns the outer height of a window, including toolbars/scrollbars. |
| outerWidth | It sets or returns the outer width of a window, including toolbars/scrollbars. |
| Parent | It returns the parent window of the current window. |
| Screen | It returns the screen object for the window. |
| screenX | It returns the X coordinate of the window relative to the screen. |
| screenY | It returns the Y coordinate of the window relative to the screen. |
| Self | It returns the current window. |



| | |
|--------|---|
| Status | It sets the text in the status bar of a window. |
| Top | It returns the topmost browser window that contains frames. |

Window Object Method

| Method | Description |
|-----------------|--|
| alert() | It displays an alert box. |
| confirm() | It displays a dialog box. |
| prompt() | It displays a dialog box that prompts the visitor for input. |
| setInterval() | It calls a function or evaluates an expression at specified intervals. |
| setTimeout() | It evaluates an expression after a specified number of milliseconds. |
| clearInterval() | It clears a timer specified by setInterval(). |
| clearTimeOut() | It clears a timer specified by setTimeout(). |
| close() | It closes the current window. |
| open() | It opens a new browser window. |
| createPopup() | It creates a pop-up window. |
| focus() | It sets focus to the current window. |
| blur() | It removes focus from the current window. |
| moveBy() | It moves a window relative to its current position. |
| moveTo() | It moves a window to the specified position. |
| resizeBy() | It resizes the window by the specified pixels. |
| resizeTo() | It resizes the window to the specified width and height. |
| print() | It prints the content of the current window. |
| scrollBy() | It scrolls the content by the specified number of pixels. |
| scrollTo() | It scrolls the content to the specified coordinates. |
| Stop() | Stops the window from loading. |

Opening a window and closing a window

The **window** object is supported by all browsers.
It represents the browser's window.

The **open()** method opens a new browser window, or a new tab, depending on your browser settings and the parameter values.



To open the window, javascript provides **open()** method.

Syntax: `window.open();`

`window.open(URL, name, specs, replace)`

| Parameter | Description |
|-------------------|---|
| <code>URL</code> | Optional. The URL of the page to open. If no URL is specified, a new blank window/tab is opened |
| <code>name</code> | Optional. The target attribute or the name of the window. The following values are supported: |

| Value | Description |
|----------------------|--|
| <code>_blank</code> | URL is loaded into a new window, or tab. This is the default |
| <code>_parent</code> | URL is loaded into the parent frame |
| <code>_self</code> | URL replaces the current page |
| <code>_top</code> | URL replaces any framesets that may be loaded |

To close the window, JavaScript provides **close ()** method.

Syntax: `window.Close();`

[Code: Open a new window](#)

```
//save as hello.html

<html>
<body>
<script>
document.write("Hello Everyone!!!!");
</script>
</body>
</html>
```

```
//save as sample.html
```

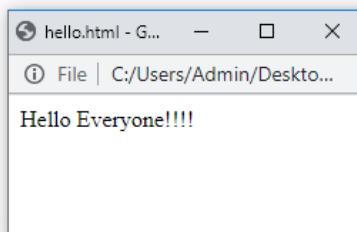
```
<html>
<body>
```



```
<script>
var ob=window.open("hello.html","",windowName","");
top=200,left=100,width=250,height=100,status");
</script>
</body>
</html>
```

Output:

← → C File | C:/Users/Admin/Desktop/Demo/window/sample.html



Code: Open a new window and close that window on button click event using open() and close().

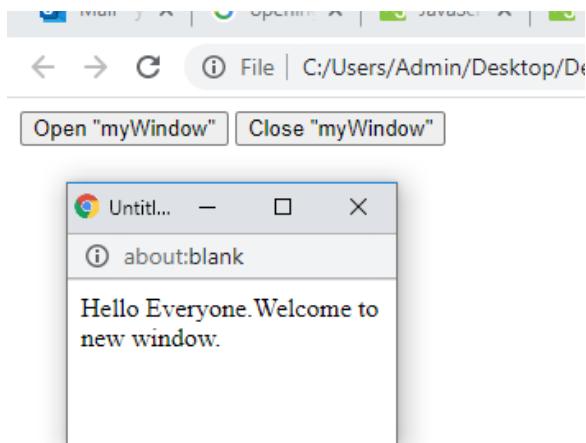
```
<html>
<body>
<button onclick="openWin()">Open "myWindow"</button>
<button onclick="closeWin()">Close "myWindow"</button>
<script>
var myWindow;
function openWin()
{
    myWindow = window.open("", "myWindow", "width=200,height=100");
    myWindow.document.write("<p>Hello Everyone.Welcome to new window.</p>");
}
function closeWin()
{
    myWindow.close();
}
</script>
</body>
</html>
```

Output:



After clicking on “open myWindow” button, new window will be open.

Click on “Close myWindow” button, newly open window will be closed.



Another example, to open vpt.edu.in website

```
<html>
<body>
<button onclick="myFunction()">Open Windows</button>
<script>
function myFunction()
{
    window.open("http://www.vpt.edu.in/");
}
</script>
</body>
</html>
```

Window position:

A new window always displayed on the screen at the location which is specified by user and location is specified by setting the left and top properties of new window as shown below:

```
window.open("http://vpt.edu.in", "windowName", top=500,left=500,width=400,height=400);
```

The **innerWidth** property returns the width of a window's content area.

The **innerHeight** property returns the height of a window's content area.



Syntax:

```
window.innerWidth  
window.innerHeight
```

The **outerWidth** property returns the outer width of the browser window, including all interface elements (like toolbars/scrollbars).

The **outerHeight** property returns the outer height of the browser window, including all interface elements (like toolbars/scrollbars).

Syntax:

```
window.outerWidth  
window.outerHeight
```

These properties are read-only.

[Code: To retrieve the dimensions of window:](#)

```
<html>  
<body>  
<div id="demo"></div>  
<script>  
var txt = "";  
txt += "<p>innerWidth: " + window.innerWidth + "</p>";  
txt += "<p>innerHeight: " + window.innerHeight + "</p>";  
txt += "<p>outerWidth: " + window.outerWidth + "</p>";  
txt += "<p>outerHeight: " + window.outerHeight + "</p>";  
document.getElementById("demo").innerHTML = txt;  
</script>  
</body>  
</html>
```

[Output:](#)



innerWidth: 606

innerHeight: 448

outerWidth: 1366

outerHeight: 728

Window.screen:

The window.screen object can be written without the window prefix.

The window.screen object contains information about the user's screen.

Properties:

| Properties | Description |
|--------------------|--|
| screen.availTop | Returns the top side of the area on the screen that is available for application windows. |
| screen.availLeft | Returns the left side of the area on the screen that is available for application windows. |
| screen.availWidth | returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar. |
| screen.availHeight | returns the height of the visitor's screen, in pixels, minus interface features like the Windows Taskbar. |
| screen.height | Returns the vertical resolution of the display screen in pixels. |
| screen.width | Returns the horizontal resolution of the display screen in pixels. |
| screen.left | Retrieves the horizontal offset of top-left corner of the current screen relative to the top-left corner of the main screen in pixels. |
| screen.top | Retrieves the vertical offset of top-left corner of the current screen relative to the top-left corner of the main screen in pixels. |

Code: To retrieve the screen properties.

```
<html>
<body>
```



```
<p id="demo"></p>
<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>
<p id="demo4"></p>
<p id="demo5"></p>
<p id="demo6"></p>
<p id="demo7"></p>
<script>
document.getElementById("demo").innerHTML =
"Available screen height is " + screen.availHeight;
document.getElementById("demo1").innerHTML =
"Available screen width is " + screen.availWidth;
document.getElementById("demo2").innerHTML =
"Available screen available Top is " + screen.availTop;
document.getElementById("demo3").innerHTML =
"Available screen available Left is " + screen.availLeft;

document.getElementById("demo4").innerHTML =
"Available screen Top is " + screen.top;
document.getElementById("demo5").innerHTML =
"Available screen Left is " + screen.left;
document.getElementById("demo6").innerHTML =
"Available screen Width is " + screen.width;
document.getElementById("demo7").innerHTML =
"Available screen Height is " + screen.height;
</script>
</body>
</html>
```

Output:



Available screen height is 728

Available screen width is 1366

Available screen available Top is 0

Available screen available Left is 0

Available screen Top is undefined

Available screen Left is undefined

Available screen Width is 1366

Available screen Height is 768

Window provides two methods which also deal with positioning of windows named scrollBy() and moveTo() method.

scrollBy(): The scrollBy() method scrolls the document by the specified number of pixels.

Syntax: window.scrollBy(xnum, ynum)

Code: Scroll the document by 100px vertically:

```
<html>
<head>
<style>
body
{
    width: 5000px;
}
button
{
    position: fixed;
}
</style>
</head>
<body>
<p>Click the button to scroll the document window by 100px horizontally.</p>
<p>Look at the horizontal scrollbar to see the effect.</p>
<button onclick="scrollWin()">Click me to scroll horizontally!</button><br><br>
```



```
<script>
function scrollWin()
{
    window.scrollBy(100, 0);
}
</script>
</body>
</html>
```

Output:

In to scroll the document window by 100px horizontally.
horizontal scrollbar to see the effect.

document window by 100px horizontally.
ar to see the effect.

The **moveTo()** method moves a window's left and top edge to the specified coordinates.

Syntax: `window.moveTo(x, y)`

The **focus()** method is used to give focus to an element (if it can be focused).

Syntax: `HTMLElementObject.focus()`

Code: simple example of `moveTo()`

```
<html>
<body>
<button onclick="openWin()">Create new window</button>
<button onclick="moveWinTo()">Move new window</button>
<button onclick="moveWinBy()">
Move the new window by 75 * 50px
</button>
<script>
var myWindow;
function openWin()
{
    myWindow = window.open("", "", "width=250, height=250");
}
function moveWinTo()
```

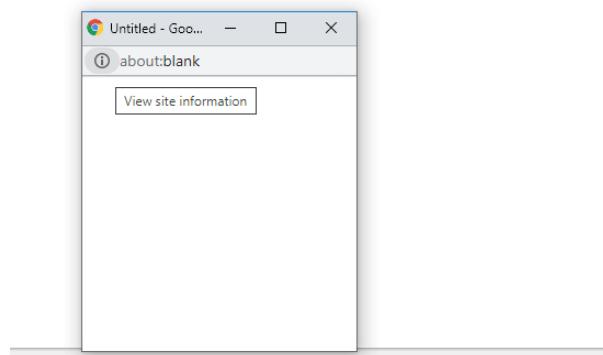


```
{  
    myWindow.moveTo(150, 150);  
    myWindow.focus();  
}  
  
function moveWinBy()  
{  
    myWindow.moveBy(75, 50);  
    myWindow.focus();  
}  
</script>  
</body>  
</html>
```

Output:

[Create new window](#) [Move new window](#) [Move the new window by 75 * 50px](#)

[Create new window](#) [Move new window](#) [Move the new window by 75 * 50px](#)



Changing the contents of window

We can change the content of opened new window as and when require.

As a new window is created and opened using open() method of window object.

In following example, we are creating only one object of window and each time same window remain open and content of window changes.

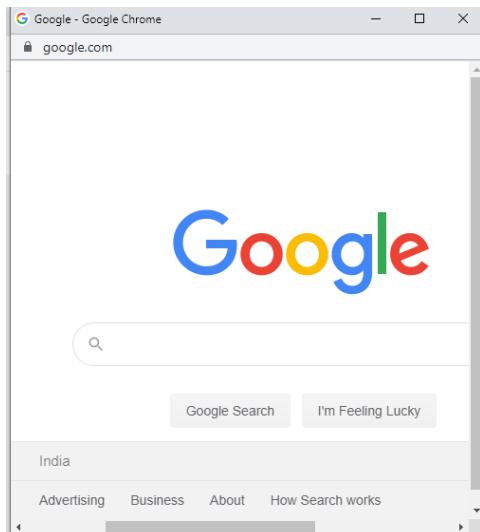
[Code: simple example of changing the contents of window.](#)

```
<html>
```



```
<body>
<script>
function openWin1(ad)
{
  myWindow = window.open(ad, "myWindow", "width=500,height=500");
}
</script>
<button value="Google" onclick="openWin1('https://www.google.com')">Google</button>
<button value="Vidyalankar" onclick="openWin1('http://vpt.edu.in')">Vidyalanakr</button>
</body>
</html>
```

Output: When you click on Google button,



When you click on Vidyalankar button,



Scrolling a Web Page

The scrollTo() method scrolls the document to the specified coordinates.

Syntax: window.scrollTo(xpos,ypos);

Where,

Xpos: Required. The coordinate to scroll to, along the x-axis (horizontal), in pixels

Ypos: Required. The coordinate to scroll to, along the y-axis (vertical), in pixels

Code: to scroll the document window to 100 pixels horizontally and vertically.

```
<html>
<head>
<style>
body
{
    width: 5000px;
    height: 5000px;
}
</style>
</head>
<body>

<script>
```



```
function scrollHorizontal()
{
    window.scrollTo(100, 0);
}

function scrollVertical()
{
    window.scrollTo(0,100);
}
</script>
<p>Click the button to scroll the document window to 100 pixels horizontally and vertically.</p>

<button onclick="scrollHorizontal()">Click me to scroll horizontally !</button>
<br><br>
<button onclick="scrollVertical()">Click me to scroll vertically!</button>

</body>
</html>
```

Output:

Click the button to scroll the document window to 100 pixels horizontally and vertically.

When user clicks on “Click me to scroll horizontally!” the output will be like this:

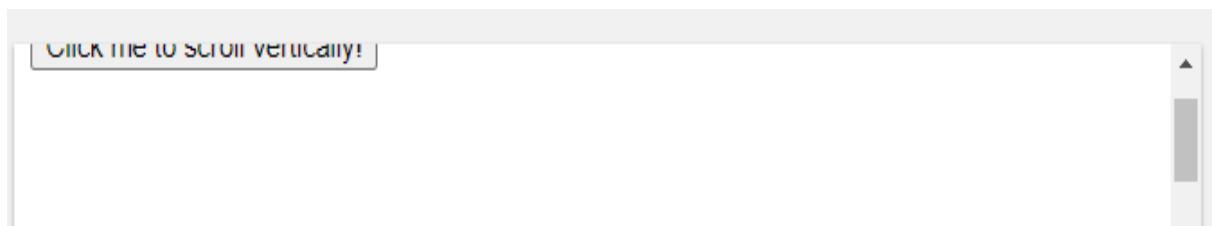


on to scroll the document window to 100 pixels horizontally and vertically.

[roll horizontally!](#)

[roll vertically!](#)

When user clicks on “Click me to scroll vertically!” the output will be like this:



Opening multiple windows at once

Creation of multiple window is possible by creating and opening window in a loop using `window.open()` method.

The only thing needs to take care is to pass proper dimension for window so that newly created window will not appear one upon another.

Code: In following example, `x` variable is assigned to top dimension of window and `y` variable is assigned to left dimension of window.

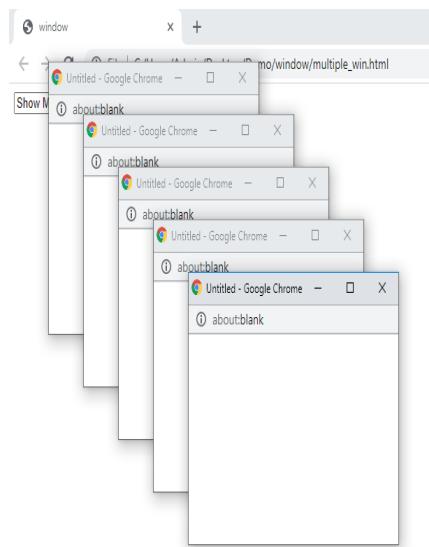
```
<html>
<head>
<title>window </title>
<script type="text/javascript">
function show( )
```



```
{  
for(i=0; i< 250 ; i=i+50)  
{  
var x=i+50;  
var y=i+50;  
winobj>window.open("",win' +i, 'top=' +x+ 'left=' +y+',width=300, height=200');  
}  
}  
</script>  
</head>  
<body>  
<input type="multiple" value="Show Multiple Windows" type="button" onclick="show()"/>  
</body>  
</html>
```

Output:

After clicking on “Show Multiple Windows” button 5 browsers will open.



Creating a web page in new window

You can place dynamic content into a new window by using the `document.write()` method. All html tags can be written inside `document.write()` method.

**Code:**

```
<html>
<head>
<title>window </title>
<script type="text/javascript">
function newWindow()
{
winobj=window.open("","winname","",width=300, height=300, left=200, top=200");
winobj.document.write('<html>');
winobj.document.write('<head>');
winobj.document.write('<title> writing Content</title>');
winobj.document.write('</head>');
winobj.document.write('<body>');
winobj.document.write('<form action="" method="post">');
winobj.document.write('<p>');
winobj.document.write('Enter Inst Name:<input type="text" name="iname">');
winobj.document.write('<br>');
winobj.document.write('<input type="submit" name="submit">');
winobj.document.write('</p>');
winobj.document.write('</form>');
winobj.document.write('</body>');
winobj.document.write('</html>');
winobj.focus();
}
</script>
</head>
<body>
<input type="button" value="New value" onclick="newWindow()">
</body>
</html>
```

Output:



← → C File | C:/Users/Admin/Desktop/Demo/window/new_window_webpage.html

New value

writing Content - Google Chrome

about:blank

Enter Inst Name:

VP-Vidyalankar

Submit

Stop()

The stop() method stops window loading.

This method is the same as clicking on the browser's stop button.

This method may be useful if the loading of an image or frame takes too long.

Syntax: `window.stop();`

```
<html>
<head>
<script>window.stop();</script>
</head>
<body>
<p>The stop() method stops this text and the iframe from loading.</p>
<p><b>Note:</b> This method does not work in Internet Explorer.</p>
<iframe src="https://vpt.edu.in/"></iframe>
</body>
```



| </html>

Javascript in URL

Another way that JavaScript code can be included on the client side is in a URL following the javascript: pseudo-protocol specifier. This special protocol type specifies that the body of the URL is arbitrary JavaScript code to be interpreted by the JavaScript interpreter. If the JavaScript code in a javascript: URL contains multiple statements, the statements must be separated from one another by semicolons. Such a URL might look like the following:

```
javascript:var now = new Date(); "The time is:" + now;
```

When the browser "loads" one of these JavaScript URLs, it executes the JavaScript code contained in the URL and displays the "document" referred to by the URL. This "document" is the string value of the last JavaScript statement in the URL. This string will be formatted and displayed just like any other document loaded into the browser. More commonly, a JavaScript URL will contain JavaScript statements that perform actions but return no value.

For example:

```
javascript:alert("Hello World!");
```

When this sort of URL is "loaded," the browser executes the JavaScript code, but, because there is no value to display as the new document, it does not modify the currently displayed document.

Javascript security

<https://snyk.io/learn/javascript-security/>

<https://blog.jscrambler.com/the-most-effective-way-to-protect-client-side-javascript-applications>

Timers

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

These time intervals are called timing events.

The two key methods to use with JavaScript are:



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

The `setTimeout()` and `setInterval()` are both methods of the HTML DOM Window object.

The `setTimeout()` Method

`window.setTimeout(function, milliseconds);`

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

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.

Code: Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<html>
<body>
<p>Click "Try it". Wait 3 seconds, and the page will alert "Hello".</p>
<button onclick="setTimeout(myFunction, 3000);">Try it</button>
<script>
function myFunction()
{
    alert('Hello');
}
</script>
</body>
</html>
```

Output:

Click "Try it". Wait 3 seconds, and the page will alert "Hello".

[Try it](#)

This page says

Hello

[OK](#)

How to Stop the Execution?



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

`window.clearTimeout(timeoutVariable)`

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

The `clearTimeout()` method uses the variable returned from `setTimeout()`:

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

If the function has not already been executed, you can stop the execution by calling the `clearTimeout()` method:

Code:

```
<html>
<body>

<p>Click "Try it". Wait 3 seconds. The page will alert "Hello".</p>
<p>Click "Stop" to prevent the first function to execute.</p>
<p>(You must click "Stop" before the 3 seconds are up.)</p>

<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>

<button onclick="clearTimeout(myVar)">Stop it</button>

<script>
function myFunction()
{
  alert("Hello");
}
</script>
</body>
</html>
```

Output:



Click "Try it". Wait 3 seconds. The page will alert "Hello".

Click "Stop" to prevent the first function to execute.

(You must click "Stop" before the 3 seconds are up.)

[Try it](#) [Stop it](#)

The setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

window.setInterval(function, milliseconds);

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

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

This example executes a function called "myTimer" once every second (like a digital watch).

Code: Display the current time:

```
<html>
<body>
<p>A script on this page starts this clock:</p>
<p id="demo"></p>
<script>
var myVar = setInterval(myTimer, 1000);
function myTimer()
{
    var d = new Date();
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
</body>
</html>
```

Output:



A script on this page starts this clock:

11:28:45 PM

How to Stop the Execution?

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

window.clearInterval(*timerVariable*)

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

The clearInterval() method uses the variable returned from setInterval():

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

code:

```
<html>
<body>
<p>A script on this page starts this clock:</p>
<p id="demo"></p>
<button onclick="clearInterval(myVar)">Stop time</button>
<script>
var myVar = setInterval(myTimer ,1000);
function myTimer()
{
    var d = new Date();
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
</body>
</html>
```

Output:

A script on this page starts this clock:

11:31:35 PM

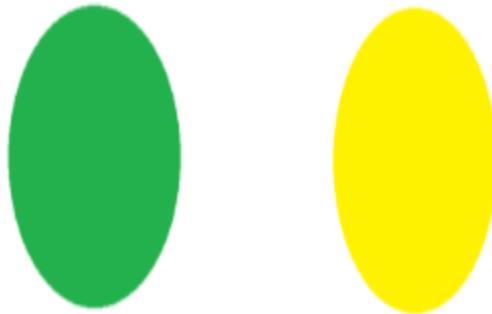


Code: Rotating images using setTimeout() method. Following example is using an array of images. Using setTimeout() method images will be rotated after 1 second.

```
<html>
<body>
<script>
var i,imgs,pic;
function init()
{
pic=document.getElementById("pic");
imgs=["red.png""blue.png""green.png""yellow.png"];
i=0;
rotate();
}

function rotate()
{
pic.src=imgs[i];
(i==(imgs.length-1))?i=0:i++;
setTimeout(rotate,1000);
}
document.addEventListener("DOMContentLoaded",init,false);
</script>
</head>
<body>
<img id="pic" width="300" height="300">
</body>
</html>
```

Output:



Code: Moving car using setTimeout() and clearTimeout() methods.

```
<html>
<head>
<title>Animation </title>
<script type="text/javascript">
var obj=null;
var animate;
function init()
{
obj=document.getElementById('car');
obj.style.position='relative';
obj.style.left='0px';
}

function start()
{
obj.style.left=parseInt(obj.style.left)+ 10 + 'px';
animate=setTimeout(start,20);
}

function stop()
{
clearTimeout(animate);
obj.style.left='0 px';
}
window.onload=init;
```



```
</script>
</head>
<body>

<br><br>
<input value="Start" type="button" onclick="start()"/>
<input value="Stop" type="button" onclick="stop()"/>
</body>
</html>
```

Output:



Code: Writing a number after a delay using setInterval () method. In this example, numbers are displayed in a textarea after a 1 second.

```
<html>
<head>
```



```
<title>number </title>
<script type="text/javascript">
var number=0;
var timerId=null;
function magic()
{
if(timerId==null)
{
timerId=setInterval("display()",1000);
}
}
function display()
{
if(number > 15)
{
clearInterval(timerId);
return;
}

document.getElementById("output").innerHTML+=number;
number++;
}
</script>
</head>
<body>
<button onclick="magic();">
Click me
</button>
<br>

<textarea id="output" rows="2" cols="20"> </textarea>
</body>
</html>
```

Output:



Click me

0123456789101112131415

Navigator Object

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

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

1. `window.navigator`

Or,

1. `navigator`



Navigator Object Properties

| No. | Property | Description |
|-----|----------------|---|
| 1 | appName | returns the name |
| 2 | appVersion | returns the version |
| 3 | appCodeName | returns the code name |
| 4 | cookieEnabled | returns true if cookie is enabled otherwise false |
| 5 | userAgent | returns the user agent |
| 6 | language | returns the language. It is supported in Netscape and Firefox only. |
| 7 | userLanguage | returns the user language. It is supported in IE only. |
| 8 | plugins | returns the plugins. It is supported in Netscape and Firefox only. |
| 9 | systemLanguage | returns the system language. It is supported in IE only. |
| 10 | mimeTypes[] | returns the array of mime type. It is supported in Netscape and Firefox only. |
| 11 | platform | returns the platform e.g. Win32. |
| 12 | online | returns true if browser is online otherwise false. |

Navigator Object Methods

| Method | Description |
|---------------|--|
| javaEnabled() | It specifies whether or not the browser is Java enabled. |

Code:

```
<html>
<body>
<script type="text/javascript">
document.write("<b>Browser: </b>" + navigator.appName + "<br><br>");
```



```
document.write("<b>Browser Version: </b>" + navigator.appVersion + "<br><br>");  
document.write("<b>Browser Code: </b>" + navigator.appCodeName + "<br><br>");  
document.write("<b>Platform: </b>" + navigator.platform + "<br><br>");  
document.write("<b>Cookie Enabled: </b>" + navigator.cookieEnabled + "<br><br>");  
document.write("<b>User Agent: </b>" + navigator.userAgent + "<br><br>");  
document.write("<b>Java Enabled: </b>" + navigator.javaEnabled() + "<br><br>");  
</script>  
</body>  
</html>
```

Output:

Browser: Netscape

Browser Version: 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36

Browser Code: Mozilla

Platform: Win32

Cookie Enabled: true

User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36

Java Enabled: false

Location Object

- Location object is a part of the window object.
- It is accessed through the '**window.location**' property.
- It contains the information about the current URL.

Location Object Properties

| Property | Description |
|----------|--|
| hash | It returns the anchor portion of a URL. |
| host | It returns the hostname and port of a URL. |
| hostname | It returns the hostname of a URL. |
| href | It returns the entire URL. |
| pathname | It returns the path name of a URL. |



| | |
|----------|---|
| port | It returns the port number the server uses for a URL. |
| protocol | It returns the protocol of a URL. |
| search | It returns the query portion of a URL. |

Location Object Methods

| Method | Description |
|-----------|--|
| assign() | It loads a new document. |
| reload() | It reloads the current document. |
| replace() | It replaces the current document with a new one. |

Code:

```
<script type="text/javascript">
    //note some parts may be blank depending on your URL
    alert("hash: " + location.hash + "\n" +
        "host: " + location.host + "\n" +
        "hostname: " + location.hostname + "\n" +
        "href: " + location.href + "\n" +
        "pathname: " + location.pathname + "\n" +
        "port: " + location.port + "\n" +
        "protocol: " + location.protocol + "\n" +
        "search: " + location.search );
</script>
```

Output:



window/location_demo_3.html

This page says

```
hash:  
host:  
hostname:  
href: file:///C:/Users/Admin/Desktop/Demo/window/  
location_demo_3.html  
pathname: /C:/Users/Admin/Desktop/Demo/window/  
location_demo_3.html  
port:  
protocol: file:  
search:
```

OK

History Object

- History object is a part of the window object.
- It is accessed through the window.history property.
- It contains the information of the URLs visited by the user within a browser window.

History Object Properties

| Property | Description |
|----------|--|
| Length | It returns the number of URLs in the history list. |
| Current | It returns the current document URL. |
| Next | It returns the URL of the next document in the History object. |
| Previous | It returns the URL of the last document in the history object. |

History Object Methods

| Method | Description |
|-----------|--|
| back() | It loads the previous URL in the history list. |
| forward() | It loads the next URL in the history list. |
| go("URL") | It loads a specific URL from the history list. |

Code: JavaScript code to show the working of history.back() function:



```
<html>
<head>
<title>GeeksforGeeks back() example</title>
</head>
<body>
<b>Press the back button</b>
<input type="button" value="Back" onclick="previousPage()">
<script>
function previousPage() {
    window.history.back();
}
</script>
</body>
</html>
```

Output:

Press the back button

```
<!DOCTYPE html>
<html>
<style>
#myProgress
{
width: 100%;
height: 30px;
position: relative;
background-color: #ddd;
}

#myBar
{
background-color: #4CAF50;
width: 5px;
height: 30px;
position: absolute;
}
```



```
</style>
<body>

<h1>JavaScript Progress Bar</h1>

<div id="myProgress">
<div id="myBar">
</div>
</div>

<br>
<button onclick="move()">Click Me</button>

<script>
function move()
{
var elem = document.getElementById("myBar");
var width = 0;
var id = setInterval(frame,200);
function frame()
{
if (width == 100)
{
clearInterval(id);
} else {
width++;
elem.style.width = width + '%';
}
}
}
</script>

</body>
</html>
```

JavaScript Progress Bar



Click Me



Toggle between two background

```
<!DOCTYPE html>
<html>
<body>

<button onclick="stopColor()">Stop Toggling</button>

<script>
var myVar = setInterval(setColor, 300);

function setColor() {
var x = document.body;
x.style.backgroundColor = x.style.backgroundColor == "yellow" ? "pink" : "yellow";
}

function stopColor() {
clearInterval(myVar);
}
</script>

</body>
</html>
```

```
<script>
// program to stop the setInterval() method after five times

let count = 0;

// function creation
let interval = setInterval(function(){

    // increasing the count by 1
    count += 1;

    // when count equals to 5, stop the function
    if(count === 5){
        clearInterval(interval);
    }

    // display the current time
    console.log(count);
})
```



V2V EDTECH LLP

DIPLOMA | DEGREE | BSCIT

```
let dateTIme= new Date();
let time = dateTIme.toLocaleTimeString();
document.write("<br>"+time);

}, 2000);

</script>
```



Unit 5

Regular Expression, Rollover and Frames

Marks: 14 (R-2, U-6, A-6)

Course Outcome: Create interactive web pages using regular expressions for validations.

Unit Outcome:

- 1) Compose relevant regular expression for the given character pattern search.
- 2) Develop JavaScript to implement validations using the given regular expression.
- 3) Create frame based on the given problem.
- 4) Create window object a per the given problem.
- 5) Develop JavaScript for creating rollover effect for the given situation.

Topics and Sub-topics:

5.1 Regular Expression

- 5.1.1 What is Regular Expression:
- 5.1.2 Language of Regular Expression
- 5.1.3 Finding Non-Matching Character
- 5.1.4 Entering a Range of Character
- 5.1.5 Matching Digits and Non-digits
- 5.1.6 Matching Punctuation and Symbols
- 5.1.7 Matching Words
- 5.1.8 Replace Text using a Regular Expression
- 5.1.9 Returned the Match Character
- 5.1.10 Regular Expression Object Properties

5.2 Frames

- 5.2.1 Create a frame
- 5.2.2 Invisible borders of frame
- 5.2.3 Calling a child window
- 5.2.4 Changing a content and focus of a child window
- 5.2.5 Writing to a child window
- 5.2.6 Accessing elements of another child window.

5.3 Rollover

- 5.3.1 Creating rollover
- 5.3.2 Text Rollover
- 5.3.3 multiple actions for rollover



5.3.4 more efficient rollover

5.1 Regular Expression

A regular expression is an object that describes a pattern of characters.

The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods that use regular expressions to perform **powerful pattern-matching and search-and-replace** functions on text.

A regular expression is very similar to a mathematical expression, except a regular expression tells the browser how to manipulate text rather than numbers by using special symbols as operators.

A **Regular Expression** is a sequence of characters that constructs a search pattern. When you search for data in a text, you can use this search pattern to describe what you are looking for.

```
/^([Reg]ular[Ex]pression$)/
```

A regular expression can be a **single character** or a more complicated pattern. It can be used for any type of text search and text replace operations. A Regex pattern consist of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /example(d+).d*/.

In JavaScript, a regular expression is an object that describes a pattern of characters. The JavaScript **RegExp** class represents regular expressions, and both String and **RegExp** define methods. It uses regular expressions to perform **pattern-matching** and **search-and-replace** functions on text.

Syntax:

A regular expression is defined with the **RegExp ()** constructor as:

```
var pattern = new RegExp(pattern, attributes);
```



or simply

```
var pattern = /pattern/attributes;
```

Here,

- **Pattern** – A string that specifies the pattern of the regular expression or another regular expression.
- **Attributes** – An optional string containing attributes that specify global, case-insensitive, and multi-line matches.

5.1.1 Language of Regular Expression

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions.

They are used to find a range of characters.

| Expression | Description |
|---------------------------|---|
| [abc] | Find any character between the brackets |
| [^abc] | Find any character NOT between the brackets |
| [0-9][A-Z][a-z][p-r][pqr] | Find any character between the brackets (any digit) |
| [^0-9] | Find any character NOT between the brackets (any non-digit) |
| (x y) | Find any of the alternatives specified |

The [abc] expression is used to find any character between the brackets.

The characters inside the brackets can be any characters or span of characters:

- [abcde..] – Any character between the brackets
- [A-Z] – Any character from uppercase A to uppercase Z
- [a-z] – Any character from lowercase a to lowercase z
- [A-z] – Any character from uppercase A to lowercase z

Syntax



```
new RegExp("[abc]")
or simply:
/[abc]/
```

Syntax with modifiers

```
new RegExp("[abc]", "g")
or simply:
/[abc]/g
```

Code: Do a global search for the characters "i" and "s" in a string:

```
<html>
<body>
<p>Click the button to do a global search for the characters "i" and "s" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Do you know if this is all there is?";
    var patt1 = /[is]/gi;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:



Click the button to do a global search for the characters "i" and "s" in a string.

Try it

i,i,s,i,s,i,s

5.1.2 Finding non-matching Characters

The [^abc] expression is used to find any character NOT between the brackets. The characters inside the brackets can be any characters or span of characters:

- ✓ [abcde..] – Any character between the brackets
- ✓ [A-Z] – Any character from uppercase A to uppercase Z
- ✓ [a-z] – Any character from lowercase a to lowercase z
- ✓ [A-z] – Any character from uppercase A to lowercase z

- Sometimes a JavaScript application prohibits certain characters from appearing within text entered into a form, such as a hyphen (-); otherwise, the character might inhibit processing of the form by the CGI program running on the web server.
- You can direct the browser to search for illegal character(s) by specifying the illegal character(s) within brackets and by placing the caret (^) as the first character in the bracket.
- Let's see how this works in the following example:
/[^-]/
- In this case, the browser is asked to determine whether the text does not contain the hyphen.
- The caret asks the browser to determine whether the following character(s) do not appear in the text.
- To find the hyphen in text, you need to escape the hyphen with the backslash, like so \-.
- Suppose you wrote the following regular expression and the browser didn't find the hyphen in the text. The browser responds with a false—this is because you are telling the browser to determine whether the hyphen appears in the text. If the hyphen appears, the browser would respond with a true.



/[\^-]/

- However, by placing a caret in the regular expression, as shown next, the browser responds with a true if the hyphen is not found in the text. This is because you are telling the browser to determine whether the hyphen does not appear in the text.
/[^-]/

Syntax

```
new RegExp("[^xyz]")
```

or simply:

```
/[^xyz]/
```

Syntax with modifiers

```
new RegExp("[^xyz]", "g")
```

or simply:

```
/\[^xyz]/g
```

Example:

```
<html> <script>
function check()
{
var exp=/[^*]/;
var res=exp.test(document.getElementById("txt1").value);
document.getElementById("demo1").innerHTML=res;
}
</script>
<body>
Enter text:<textarea id="txt1"></textarea>
<input type="button" onclick="check()" value="Check">
<p id="demo1"></p>
</body>
</html>
```

Output:



File | C:/Users/acer/Desktop/nonmatching.html

Enter text: Check

true



File | C:/Users/acer/Desktop/nonmatching.html

Enter text: Check

false

Code: Do a global search for characters that are NOT "i" and "s" in a string:

```
<html>
<body>
<p>Click the button to do a global search for characters that are NOT "i" and "s" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var str = "Do you know if this is all there is?";
    var patt1 = /[^is]/gi;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
```



```
</script>
</body>
</html>
```

Output:

Click the button to do a global search for characters that are NOT "i" and "s" in a string.

D,o, ,y,o,u, ,k,n,o,w, ,f, ,t,h, , ,a,l,l, ,t,h,e,r,e, ,?

The [^0-9] expression is used to find any character that is NOT a digit.

The digits inside the brackets can be any numbers or span of numbers from 0 to 9.

Syntax

```
new RegExp("[^0-9]")
or simply:
/[^0-9]/
```

Syntax with modifiers

```
new RegExp("[^0-9]", "g")
or simply:
/[^0-9]/g
```

Code: Do a global search for numbers that are NOT "1" and "s" in a string.

```
<html>
<body>
<p>Click the button to do a global search for numbers that are NOT "1" and "s" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
```



```
{  
var str = "12121212This is JavaScript";  
var patt1 = /[^s]/gi;  
var result = str.match(patt1);  
document.getElementById("demo").innerHTML = result;  
}  
</script>  
</body>  
</html>
```

Output:

Click the button to do a global search for numbers that are NOT "1" and "s" in a string.

[Try it](#)

2,2,2,2,T,h,i, ,i, ,J,a,v,a,c,r,i,p,t

5.1.3 Entering a range of characters

The [0-9] expression is used to find any character between the brackets.

The digits inside the brackets can be any numbers or span of numbers from 0 to 9.

Note: can used for alphabets [a-z] and [A-Z] or [p-r] or [P-R] or [pqr] or [PQR]

Syntax

```
new RegExp("[0-9]")  
or simply:  
/[0-9]/
```

Syntax with modifiers

```
new RegExp("[0-9]", "g")  
or simply:  
/\d/g
```

Code: Do a global search for the number "1" and "s" in a string:

```
<html>  
<body>
```



```
<p>Click the button to do a global search for the number "1" and "s" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var str = "12121212This is Javascript";
    var patt1 = /[1s]/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global search for the number "1" and "s" in a string.

Try it

1,1,1,1,s,s,s

The (x|y) expression is used to find any of the alternatives specified.
The alternatives can be of any characters.

Syntax:

```
new RegExp("(x|y)")  
or simply:  
/(x|y)/
```

Syntax with modifiers:

```
new RegExp("(x|y)", "g")  
or simply:  
/(x|y)/g
```



Code: Do a global search to find any of the specified alternatives (0|5|7):

```
<html> <body>
<p>Click the button to do a global search for any of the specified alternatives (0|5|7).
</p> <button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "01234567890123456789";
    var patt1 = /(0|5|7)/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script> </body> </html>
```

Output:

Click the button to do a global search for any of the specified alternatives (0|5|7).

[Try it](#)

0,5,7,0,5,7

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation.

The +, *, ?, and \$ flags all follow a character sequence.

| Sr.No. | Expression & Description |
|--------|---|
| 1 | p+ It matches any string containing one or more p's. |

Non-Explicit Quantifiers

Explicit Quantifiers



| | |
|---|---|
| 2 | p^* It matches any string containing zero or more p's. |
| 3 | $p^?$ It matches any string containing at most one p.(zero or one occurrences) |
| 4 | $p\{N\}$ It matches any string containing a sequence of N p's |
| 5 | $p\{2,3\}$ It matches any string containing a sequence of two or three p's. |
| 6 | $p\{2,\}$ It matches any string containing a sequence of at least two p's. |
| 7 | $p\$$ It matches any string with p at the end of it. |
| 8 | p It matches any string with p at the beginning of it. |

Code: The n+ quantifier matches any string that contains at least one n.

```
<html>
<body>
<p>Click the button to do a global search for at least one "o" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Hellooo World! Hello Vidyalanakr School!";

```



```
var patt1 = /o+/g;
var result = str.match(patt1);
document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global search for an "1", followed by zero or more "o" characters.

Try it

ooo,o,o,oo

Code: The n* quantifier matches any string that contains zero or more occurrences of n.

```
<html>
<body>

<p>Click the button to do a global search for at least one "o" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var str = "Hellooo World! Hello Vidyalanakr School!";
  var patt1 = /o+/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
</script>
</body>
```



```
</html>
```

Output:

Click the button to do a global search for an "1", followed by zero or more "0" characters.

[Try it](#)

1,1ooo,1,1,1o,1,1

Code: The n? quantifier matches any string that contains zero or one occurrences of n.

```
<html>
<body>
<p>Click the button to do a global search for a "1", followed by zero or one "0" characters.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>

<script>
function myFunction() {
  var str = "1, 100 or 1000?";
  var patt1 = /10?/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:



Click the button to do a global search for a "1", followed by zero or one "0" characters.

Try it

1,10,10

Code: The `n{X,Y}` quantifier matches any string that contains a sequence of X to Y n's. X and Y must be a number.

```
<html>
<body>
<p>Click the button to global search for a substring that contains a sequence of three to four digits.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "100,1000 or 10000?";
    var patt1 = /\d{3,4}/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script> </body> </html>
```

Output:

Click the button to global search for a substring that contains a sequence of three to four digits.

Try it

100,1000,1000



Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for a large sum of money using the '\d' metacharacter: `/(\d)+000/`, Here \d will search for any string of numerical character.

The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

| Sr.No. | Character & Description |
|--------|---|
| 1 | <code>.</code> (dot) a single character |
| 2 | <code>\s</code> a whitespace character (space, tab, newline) |
| 3 | <code>\S</code> non-whitespace character |
| 4 | <code>\d</code> a digit (0-9) |
| 5 | <code>\D</code> a non-digit |
| 6 | <code>\w</code> a word character (a-z, A-Z, 0-9, _) |
| 7 | <code>\W</code> a non-word character |
| 8 | <code>[\b]</code> a literal backspace (special case). |
| 9 | <code>[aeiou]</code> |



| | |
|----|--|
| | matches a single character in the given set |
| 10 | [^aeiou] matches a single character outside the given set |
| 11 | (foo bar baz) matches any of the alternatives specified |

5.1.4 Matching Digits and Nondigits

- You can have the browser check to see whether the text has digits or non-digits by writing a regular expression. The regular expression must contain either \d or \D, depending on whether you want the browser to search the text for digits (\d) or nondigits (\D).
- The \d symbol, tells the browser to determine whether the text contains digits. The browser returns a true if at least one digit appears in the text.
- You'd use this regular expression to determine whether a first name has any digits. The \D symbol is used to tell the browser to search for any nondigit in the text.
- The browser returns a true if a nondigit is found. This is the regular expression you would use to validate a telephone number, assuming the user was asked to enter digits only. If the browser finds a nondigit, the telephone number is invalid and you can notify the user who entered the information into the form.

Example: Program to check for any two-consecutive digit in input.

```
<html>
<script>
function check()
{
var exp=/\d\d/;
var str=exp.test(document.getElementById("txt").value);
document.getElementById("demo1").innerHTML=str;
}
```



```
</script>
<body>
Enter text:<textarea id="txt"></textarea>
<input type="button" onclick="check()" value="check">
<p id="demo1"></p>
</body>
</html>
```

Output:

The screenshot shows two separate runs of a web application. Both runs have the same URL: C:/Users/acer/Desktop/digits.html. The first run has the input 'ab23' and outputs 'true'. The second run has the input 'ab2' and outputs 'false'. The browser interface includes standard navigation buttons (back, forward, refresh), a file menu, and the current file path.

true

false

Following table shows the test cases on the above program.

| Pattern | Code | Input | Output |
|---------|--|-------|--------|
| \d\d/ | var exp=/\d\d/; var output=exp.test(input); | A23 | True |
| | | A2 | False |
| | | 23 | True |



| | | | |
|-------|---|-----------|-------|
| | | 23A | True |
| | | 2 | False |
| | | A1B3 | False |
| \d{4} | var exp=/\d{4} var output=exp.test(Input); | A1B2C3D4 | False |
| | | A1B2C3D | False |
| | | A1b2c3456 | True |

Code: The \d metacharacter is used to find a digit from 0-9.

```
<html>
<body>
<p>Click the button to do a global search for digits in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Give 100%!";
    var patt1 = /\d/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:



Click the button to do a global search for digits in a string.

Try it

1,0,0

Code: The \D metacharacter is used to find a non-digit character.

```
<html> <body>
<p id="demo">Click the button to do a global search for non-digit characters in a
string.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
var str = "Give 100%!";
var patt1 = /\D/g;
var result = str.match(patt1);
document.getElementById("demo").innerHTML=result;
}
</script> </body> </html>
```

Output:

Click the button to do a global search for non-digit characters in a string.

Try it



Give %!

Try it

Code: The . metacharacter is used to find a single character, except newline or other line terminators.

```
<html>
<body>
<p>Click the button to do a global search for "h.t" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "That's hot!";
    var patt1 = /h.t/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global search for "h.t" in a string.

Try it

hat,hot



5.1.5 Matching Punctuation and Symbols

- You can have the browser determine whether text contains or doesn't contain letters, punctuation, or symbols, such as the @ sign in an e-mail address, by using the \w and \W special symbols in a regular expression.
- The \w special symbol tells the browser to determine whether the text contains a letter, number, or an underscore, and the \W special symbol reverses this request by telling the browser to determine whether the text contains a character other than a letter, number, or underscore.
- You can use the following regular expression to determine whether the product name that was entered into the form on your web page contains a symbol:
/\W/
• Using \W is equivalent to using [a-zA-Z0-9].

Example:

```
<html>
<script>
function check()
{
var exp=/\w{2}/;
var str=document.getElementById("txt").value;
var res=exp.test(str);
document.getElementById("demo1").innerHTML=res;
}
</script>
<body>
Enter text:<textarea id="txt"></textarea>
<input type="button" onclick="check()" value="check">
<p id="demo1"></p>
</body>
</html>
```

Output:



← → C File | C:/Users/acer/Desktop/symbols.html

Enter text:

false

← → C File | C:/Users/acer/Desktop/symbols.html

Enter text:

true

Code: The \w metacharacter is used to find a word character.

A word character is a character from a-z, A-Z, 0-9, including the _ (underscore) character.

```
<html>
<body>
<p>Click the button to do a global search for word characters in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Give 100!";
    var patt1 = /\w/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
```



```
</script>
</body>
</html>
```

Output:

Click the button to do a global search for word characters in a string.

[Try it](#)

G,i,v,e,1,0,0

Code: The \W metacharacter is used to find a non-word character.
A word character is a character from a-z, A-Z, 0-9, including the _ (underscore) character.

```
<html>
<body>
<p>Click the button to do a global search for non-word characters in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Give 100%!";
    var patt1 = /\W/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:



Click the button to do a global search for non-word characters in a string.

Try it

,%,!

5.1.6 Matching Words

- You might want the browser to search for a particular word within the text. A word is defined by a word boundary—that is, the space between two words. You define a word boundary within a regular expression by using the \b special symbol.
- You need to use two \b special symbols in a regular expression if you want the browser to search for a word: the first \b represents the space at the beginning of the word and the second represents the space at the end of the word.

Example:

```
<html>
<script>
function check()
{
var exp=/\b\w{2}\b/;
var str=document.getElementById("txt").value;
var res=exp.test(str);
document.getElementById("demo1").innerHTML=res;
}
</script>
<body>
Enter text:<textarea id="txt"></textarea>
<input type="button" onclick="check()" value="check">
<p id="demo1"></p>
</body>
</html>
```



Output:

← → C File | C:/Users/acer/Desktop/matchingwords.html

Enter text:

true

← → C File | C:/Users/acer/Desktop/matchingwords.html

Enter text:

false

Code: The \t metacharacter is used to find a tab character.

\t returns the position where the tab character was found. If no match is found, it returns -1.

```
<html> <body>
<p>Click the button to return the position where the tab character was found in a string.
</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
    var str = "Visit Vidyalankar Polytechnic.\t To Learn JavaScript.";
    var patt1 = /\t/;
    var result = str.search(patt1);
    document.getElementById("demo").innerHTML = result;
}
```



```
}
```

```
</script> </body>
```

```
</html>
```

Output:

Click the button to return the position where the tab character was found in a string.

Try it

30

Modifiers/flag

Several modifiers are available that can simplify the way you work with **regexp**s, like case sensitivity, searching in multiple lines, etc.

| Sr.No. | Modifier & Description |
|--------|--|
| 1 | i Perform case-insensitive matching. |
| 2 | m Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary |
| 3 | g Performs a global match that is, find all matches rather than stopping after the first match. |

Code: The g modifier is used to perform a global match (find all matches rather than stopping after the first match).

Syntax:

```
new RegExp("regexp", "g")  
or simply:  
/regexp/g
```



Example: 1)

```
<html>
<body>
<p>Click the button to do a global search for "is" in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var str = "Is this all there is?";
    var patt1 = /is/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global search for "is" in a string.

[Try it](#)

is,is

Example 2):

```
<html>
<body>
<p>Click the button to do a global search for the character-span [a-h] in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```



```
<script>
function myFunction()
{
    var str = "Is this all there is?";
    var patt1 = /[a-h]/g;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global search for the character-span [a-h] in a string.

[Try it](#)

h,a,h,e,e

Code:

The i modifier is used to perform case-insensitive matching.

Syntax

```
new RegExp("regexp", "i")
or simply:
/regexp/i
```

Example:

```
<html>
<body>
<p>Click the button to do a global, case-insensitive search for "is" in a string.
</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
```



```
<script>
function myFunction() {
  var str = "Is this all there is?";
  var patt1 = /is/gi;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global, case-insensitive search for "is" in a string.

[Try it](#)

Is,is,is

Code: The m modifier is used to perform a multiline match.

The m modifier treat beginning (^) and end (\$) characters to match the beginning or end of each line of a string (delimited by \n or \r), rather than just the beginning or end of the string.

Syntax:

```
new RegExp("regexp", "m")
```

or simply:

```
/regexp/m
```

Example:

```
<html>
```

```
<body>
```



```
<p>Click the button to do a global, case-insensitive, multiline search for "is" at the beginning of each line in a string.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    var str = "\nIs th\nis h\nis?";
    var patt1 = /^is/gmi;
    var result = str.match(patt1);
    document.getElementById("demo").innerHTML = result;
}
</script>
</body>
</html>
```

Output:

Click the button to do a global, case-insensitive, multiline search for "is" at the beginning of each line in a string.

[Try it](#)

Is,is,is

RegExp Methods

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

| Sr.No. | Method & Description |
|--------|---|
| 1 | <u>exec()</u> Executes a search for a match in its string parameter. |
| 2 | <u>test()</u> Tests for a match in its string parameter. |



| | |
|---|---|
| | <u>toSource()</u> |
| 3 | Returns an object literal representing the specified object; you can use this value to create a new object. |
| | <u>toString()</u> |
| 4 | Returns a string representing the specified object. |

5.1.7 Replace Text using a Regular Expression:

- you can also use a regular expression to replace portions of the text by using the `replace()` method.
- The `replace()` method requires two parameters: a regular expression and the replacement text.
- Here's how the `replace()` method works. First, you create a regular expression that identifies the portion of the text that you want replaced.
- Then you determine the replacement text. Pass both of these to the `replace()` method, and the browser follows the direction given in the regular expression to locate the text. If the text is found, the browser replaces it with the new text that you provided.

The `replace()` method searches a string for a specified value, or a *regular expression*.

Syntax

```
string.replace(searchvalue, newvalue)
```

Example:

```
<html>
<script>
function check()
{
var exp=/:/g;
var str=document.getElementById("txt").value;
var res=str.replace(exp,"-");
document.getElementById("demo1").innerHTML=res;
}
```



```
</script>
<body>
Enter text:<textarea id="txt"></textarea>
<input type="button" onclick="check()" value="check">
<p id="demo1"></p>
</body>
</html>
```

Output:

← → ⌂ File | C:/Users/acer/Desktop/replace.html

12:03:2020

Enter text: check

12-03-2020

In the above example “:” symbol replace by “-“

Code:

```
<html>
<body>
<p>Click the button to replace "blue" with "red" in the paragraph below:</p>
<p id="demo">Mr Blue has a blue house and a blue car.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
    var str = document.getElementById("demo").innerHTML;
    var res = str.replace(/blue/g, "red");
    document.getElementById("demo").innerHTML = res;
    document.getElementById("demo").style.color = "red";
}
```



```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Output:

Click the button to replace "blue" with "red" in the paragraph below:

Mr Blue has a blue house and a blue car.

Try it

Click the button to replace "blue" with "red" in the paragraph below:

Mr Blue has a red house and a red car.

Try it

5.1.8 Returned the Match Character

- Sometimes your JavaScript application requires you to retrieve characters that match a regular expression rather than simply testing whether or not those characters exist in the text.
- You can have the browser return characters that match the pattern in your regular expression by calling the exec() method of the regular expression object.
- How to use the exec() method. First, create a regular expression that specifies the pattern that you want to match within the text. Characters that match this pattern will be returned to your JavaScript. Next, pass the exec() method the text for which you want to search. The exec() method returns an array. The first array element contains the text that matches your regular expression.

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 );
```



Code:

```
<html>
  <head>
    <title>JavaScript RegExp exec Method</title>
  </head>
  <body>
    <script type = "text/javascript">
      var str = "Javascript is an interesting scripting language";
      var re = new RegExp( "script", "g" );
      var result = re.exec(str);
      document.write("Test 1 - returned value : " + result);
      re = new RegExp( "pushing", "g" );
      var result = re.exec(str);
      document.write("<br />Test 2 - returned value : " + result);
    </script>
  </body>
</html>
```

Output:

Test 1 - returned value : script
Test 2 - returned value : null

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 );
```

Code:

```
<html>
  <head>
```



```
<title>JavaScript RegExp test Method</title>
</head>
<body>
<script type = "text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.test(str);
    document.write("Test 1 - returned value : " + result);

    re = new RegExp( "nothing", "g" );

    var result = re.test(str);
    document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```

Output:

Test 1 - returned value : true
Test 2 - returned value : false

The search() method searches a string for a specified value, and returns the position of the match.

The search value can be string or a regular expression.

This method returns -1 if no match is found.

Syntax

string.search(searchvalue)

Code:



```
<html>
<body>

<p id="demo">Mr Blue has a blue house and a blue car.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction()
{
    var str = document.getElementById("demo").innerHTML;
    var res = str.search(/red/g);
    document.getElementById("demo").innerHTML = res;
    document.getElementById("demo").style.color = "red";
}
</script>
</body>
</html>
```

Output:

Mr Blue has a blue house and a blue car.

-1

[Try it](#)

[Try it](#)

5.1.9 RegExp Properties

Here is a list of the properties associated with RegExp and their description.

| Sr.No. | Property & Description |
|--------|--|
| 1 | <u>constructor</u> Specifies the function that creates an object's prototype. |
| 2 | <u>global</u> |



| | |
|---|---|
| | Specifies if the "g" modifier is set. |
| 3 | <u>ignoreCase</u> |
| | Specifies if the "i" modifier is set. |
| 4 | <u>lastIndex</u> |
| | The index at which to start the next match. |
| 5 | <u>multiline</u> |
| | Specifies if the "m" modifier is set. |
| 6 | <u>source</u> |
| | The text of the pattern. |

In the following sections, we will have a few examples to demonstrate the usage of RegExp properties.

ignoreCase is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs case-insensitive matching, i.e., whether it was created with the "i" attribute.

Syntax:

```
RegExpObject.ignoreCase
```

Code:

```
<html>
  <head>
    <title>JavaScript RegExp ignoreCase Property</title>
  </head>
  <body>
    <script type = "text/javascript">
      var re = new RegExp( "string" );
```



```
if ( re.ignoreCase )
{
    document.write("Test1-ignoreCase property is set");
} else
{
    document.write("Test1-ignoreCase property is not set");
}

re = new RegExp( "string", "i" );

if ( re.ignoreCase )
{
    document.write("<br/>Test2-ignoreCase property is set");
} else
{
    document.write("<br/>Test2-ignoreCase property is not set");
}
</script>
</body>
</html>
```

Output:

Test1-ignoreCase property is not set
Test2-ignoreCase property is set

multiline is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs multiline matching, i.e., whether it was created with the "m" attribute.

Syntax

RegExpObject.multiline



Code:

```
<html>
  <head>
    <title>JavaScript RegExp multiline Property</title>
  </head>
  <body>
    <script type = "text/javascript">
      var re = new RegExp( "string" );
      if ( re.multiline )
      {
        document.write("Test1-multiline property is set");
      }
      else
      {
        document.write("Test1-multiline property is not set");
      }
      re = new RegExp( "string", "m" );

      if ( re.multiline )
      {
        document.write("<br/>Test2-multiline property is set");
      }
      Else
      {
        document.write("<br/>Test2-multiline property is not set");
      }
    </script>
  </body>
</html>
```



Output:

```
Test1 - multiline property is not set
Test2 - multiline property is set
```

5.2 Frames

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

5.2.1 Create a Frame

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Note – The <frame> tag deprecated in HTML5. Do not use this element.

Example

Following is the example to create three horizontal frames –

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="50%,30%,*">
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
<frame src="webpage3.html" name="bottomPage" />
</frameset>
</html>
```

Output:



Web Page 1

Web Page 2

Web Page 3

vp
vp
vp

Example

Let's put the above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically –

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset cols="50%,30%,*">
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
<frame src="webpage3.html" name="bottomPage" />
```



```
</frameset>  
</html>
```

Output:



The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag –

| Sr.No | Attribute & Description |
|-------|--|
| 1 | cols Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways – Absolute values in pixels. For example, to create three vertical frames, use <code>cols = "100, 500, 100"</code> . A percentage of the browser window. For example, to create three vertical frames, use <code>cols = "10%, 80%, 10%"</code> . Using a wildcard symbol. For example, to create three vertical frames, use <code>cols = "10%, *, 10%"</code> . In this case wildcard takes remainder of the window. |
| 2 | rows |



| | |
|---|--|
| | <p>This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use <code>rows = "10%, 90%"</code>. You can specify the height of each row in the same way as explained above for columns.</p> |
| 3 | <p>border</p> <p>This attribute specifies the width of the border of each frame in pixels. For example, <code>border = "5"</code>. A value of zero means no border.</p> |
| 4 | <p>frameborder</p> <p>This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example, <code>frameborder = "0"</code> specifies no border.</p> |
| 5 | <p>framespacing</p> <p>This attribute specifies the amount of space between frames in a frameset. This can take any integer value.</p> <p>For example:</p> <p><code>framespacing = "10"</code> means there should be 10 pixels spacing between each frame.</p> |

The <frame> Tag Attributes

Following are the important attributes of <frame> tag –

| Sr.No | Attribute & Description |
|-------|-------------------------|
| 1 | src |



| | |
|---|---|
| | <p>This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.</p> |
| 2 | <p>name</p> <p>This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into another frame, in which case the second frame needs a name to identify itself as the target of the link.</p> |
| 3 | <p>frameborder</p> <p>This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).</p> |
| 4 | <p>marginwidth</p> <p>This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example, marginwidth = "10".</p> |
| 5 | <p>marginheight</p> <p>This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example, marginheight = "10".</p> |
| 6 | <p>noresize</p> <p>By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame.</p> |



scrolling

- 7 This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example, scrolling = "no" means it should not have scroll bars.

Example:

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="30%,20%,*" frameborder="1" border="20" bordercolor="blue"
scrolling="auto" noresize>
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
<frameset cols="30%,*>
<frame src="webpage3.html" name="bottomPage" />
<frameset rows="50%,*" frameborder="0" bordercolor="red">
<frame src="webpage3.html" name="bottomPage" />
<frame src="webpage3.html" name="bottomPage" />
</frameset>
</frameset></frameset>
</html>
```

Output:



Web Page 1

Web Page 2

Web Page 3

vp
vp

Web Page 3

vp
vp

5.2.2 Invisible border of Frame

User can hide border by using frameborder attribute.

frameborder

This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).

Example:

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="30%,20%,*" frameborder="0">
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
<frameset cols="30%,*>
```

```
<frame src="webpage3.html" name="bottomPage" />
<frameset rows="50%,*" frameborder="0" bordercolor="red">
<frame src="webpage3.html" name="bottomPage" />
<frame src="webpage3.html" name="bottomPage" />
</frameset>
</frameset></frameset>
</html>
```

Output:

Web Page 1

Web Page 2



5.2.3 Calling a child window's Javascript function

Using frame in the frameset means creating child window inside the parent window. Here each frame represents a child window and frameset represents the parent window.



You can refer to another child window by referencing the frameset, which is the parent window, and then by referencing the name of the child window, followed by whatever element within the web page of the child window that you want to access.

Following example creates two child windows. First child window (name=topPage) consists of a button.

And Second child window (name=bottomPage) consist of a button.

From second child window you call the method of the first child window using the reference of parent hence when you click on button of second child window then you can call Javascript function of first child window via reference of parent.

Example:

Sample.html

```
<html>
<head>
<title>Create a Frame</title>
</head>
<frameset rows="50%,50%">
<frame src="webpage1.html" name="topPage" frameborder="0" />
<frame src="webpage2.html" name="bottomPage" frameborder="0" />
</frameset>
</html>
```

Webpage1.html

```
<html>
<head>
<title>Web Page 1</title>
<script>
function ChangeContent()
{
alert("Function Called")
}
</script>
```



```
</head>
<body>
<input name="WebPage1" value="Web Page 1" type="button" />
</body>
</html>
```

Webpage2.html

```
<html>
<head>
<title>Web Page 2</title>
</head>
<body>
<p>
<input name="WebPage2" value="Web Page 2" type="button"
onclick="parent.topPage.ChangeContent()" />
</p>
</form>
</body>
</html>
```

5.2.4.1 Changing the content of Child Window:

You can change the content of a child window from a JavaScript function by modifying the source web page for the child window. To do this, you must assign the new source to the child window's href attribute.

In this example, you were able to get a reference to the parent frame's topPage element because they are both from the same domain. At that point, you have two options: if they're in the same domain, you reference it as illustrated previously, but you can also just change the frame src attribute in the frameset to point the frame to a new page.

In the following example we'll need to modify both the WebPage1.html and WebPage2.html files. In addition, we'll need to define a new web page called WebPage3.html.

Here is the new WebPage1.html file. WebPage1.html appears in the bottom child



window, and when the Web Page 1 button is clicked, the content of the top child window changes from WebPage2.html to WebPage3.html. You'll notice that the value of the button reflects the new content.

Sample.html

```
<html>
<head>
<title>Frame</title>
</head>
<frameset rows="50%,50%">
<frame src="webpage1.html" name="topPage" frameborder="0" border="0" />
<frame src="webpage2.html" name="bottomPage" frameborder="0" border="0" />
</frameset>
</html>
```

Webpage1.html

```
<html>
<head>
<title>web page1</title>
<script>
function ChangeContent()
{
parent.topPage.location.href="webpage3.html";
}
</script>
</head>
<body>
<input name="webpage1" value="WebPage1" type="button" /><br>
</body>
</html>
```



Webpage2.html

```
<html>
<head>
<title></title>
</head>
<body>
<input name="WebPage2" value="Web Page2" type="button" />
</body>
</html>
```

Webpage3.html

```
<html>
<input name="WebPage3" value="web Page3" type="button" />
</html>
```

5.2.4.2 Changing the Focus of a Child Window

The last child window that is created has the focus by default; however, you can

give any child window the focus by changing the focus after all the web pages have loaded in their corresponding child windows. You do this by calling the focus() method of the child window, as shown next, where the focus is being given to the web page that appears in the bottomPage child window. You can call the focus() method from a JavaScript function or directly in response to an event such as the onclick event. The reference to parent.bottomPage is needed to get past the security issues.

```
parent.bottomPage.focus();
```

5.2.5 Writing to a Child Window from a JavaScript

Typically, the content of a child window is a web page that exists on the web server. However, you can dynamically create the content when you define the frameset by directly writing to the child window from a JavaScript. The JavaScript must be defined in the HTML file that defines the frameset and called when the frameset is loaded.

```
<html>
```



```
<head>
<title>Create a Frame</title>
<script>
function ChangeContent()
{
    window.topPage.document.open()
    window.topPage.document.writeln('<html >')
    window.topPage.document.writeln('<head>')
    window.topPage.document.writeln('<title>Web Page 3</title>')
    window.topPage.document.writeln('</head>')
    window.topPage.document.writeln('<body>')
    window.topPage.document.writeln('<FORM action="" method="post">')
    window.topPage.document.writeln('<P>')
    window.topPage.document.writeln(
        '<INPUT name="WebPage3" value="Web Page 3" type="button" />')
    window.topPage.document.writeln('</P>')
    window.topPage.document.writeln('</FORM>')
    window.topPage.document.writeln('</body>')
    window.topPage.document.writeln('</html>')
    window.topPage.document.close()
}
</script>
<frameset rows="50%,50%" onload="ChangeContent()">
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
</frameset>
</html>
<frame src="webpage2.html" name="bottomPage" />
</frameset>
</html>
```



5.2.6 Accessing Elements of Another Child Window

You can access and change the value of elements of another child window by directly referencing the element from within your JavaScript. You must explicitly specify the full path to the element in the JavaScript statement that references the element, and it must be from the same domain as the web page; otherwise, a security violation occurs.

```
<html>
<head>
<title>Create a Frame</title>
<script >
function ChangeContent()
{
window.topPage.document.open()
window.topPage.document.writeln('<html >')
window.topPage.document.writeln('<head>')
window.topPage.document.writeln('<title>Web Page 3</title>')
window.topPage.document.writeln('</head>')
window.topPage.document.writeln('<body>')
window.topPage.document.writeln(
'<FORM name="Form1" action="" method="post">')
window.topPage.document.writeln('<P>')
window.topPage.document.writeln('<input name="Text1" type="text"/>')
window.topPage.document.writeln(
'<INPUT name="WebPage1" value="Web Page 1" type="button" />')
window.topPage.document.writeln('</P>')
window.topPage.document.writeln('</FORM>')
window.topPage.document.writeln('</body>')
window.topPage.document.writeln('</html>')
window.topPage.document.close()
}
```



```
</script>
</head>
<frameset rows="50%,50%" onload="ChangeContent()">
<frame src="webpage1.html" name="topPage" />
<frame src="webpage2.html" name="bottomPage" />
</frameset>
</html>
```

```
<html>
<head>
<title></title>
</head>
<script>
function accessElement()
{
parent.topPage.Form1.Text1.value="Manisha";
parent.topPage.Form1.WebPage1.value="Accessed";
}
</script>
<body>
<h1>Web Page 2</h1>
<input name="WebPage2" value="Web Page2" type="button" onclick="accessElement()" />
</body>
</html>
```

Note: This Frameset and javascript is not supported by browser.

IFRAMES

You can define an inline frame with HTML tag **<iframe>**. The **<iframe>** tag is not somehow related to **<frameset>** tag, instead, it can appear anywhere in your document. The **<iframe>** tag defines a rectangular region within the document in which the browser can display a



separate document, including scrollbars and borders. An inline frame is used to embed another document within the current HTML document.

The **src** attribute is used to specify the URL of the document that occupies the inline frame.

Example:

```
<html>
<body>
<h3>A demonstration of how to access an IFRAME element</h3>
<iframe id="myFrame" src="webpage2.html"></iframe>
<iframe id="myFrame1" src="webpage1.html"></iframe>
<p>Click the button to get the URL of the iframe.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<p id="demo1"></p>
<script>
function myFunction()
{
    var x = document.getElementById("myFrame").src;
    document.getElementById("demo").innerHTML = x;
    var y = document.getElementById("myFrame1").src;
    document.getElementById("demo1").innerHTML = y;
}
</script>
</body>
</html>
```



Output:

A demonstration of how to access an IFRAME element



Click the button to get the URL of the iframe.

Try it

file:///C:/Users/Admin/Desktop/CSS_Lect/Unit%205_Regular_eXpression_Rollover_frames/frame_prog/webpage2.html
file:///C:/Users/Admin/Desktop/CSS_Lect/Unit%205_Regular_eXpression_Rollover_frames/frame_prog/webpage1.html

Key Difference: Frame is a HTML tag that is used to divide the web page into various frames/windows. Used as <frame> tag, it specifies each frame within a frameset tag. Iframe as <iframe> is also a tag used in HTML but it specifies an inline frame which means it is used to embed some other document within the current HTML document.

We can access the content of child window by using <iframe>.

In following example, we have used window.addEventListener().

Code: Parent.html

```
<html>
<head>
<script type="text/javascript">
    window.addEventListener('message', receiveMessage, false);
function receiveMessage(event)
{
    alert("got message: " + event.data);
}
</script>
```



```
</head>
<title> Parent Page </title>
<body>
    <iframe src="iframe_p.html" width="500" height="500"></iframe>
</body>
</html>
```

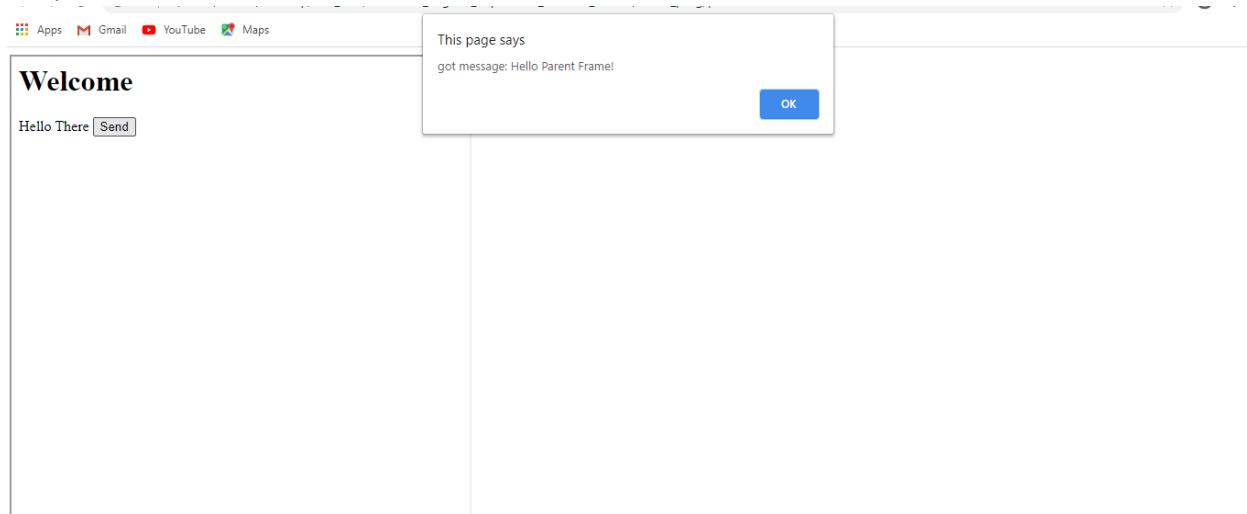
iframe_p.html

The **window.postMessage()** method safely enables cross-origin communication between Window objects; e.g., between a page and a pop-up that it produced, or between a page and an iframe embedded within it.

```
<html>
<head>
    <script>
        function send()
        {
            window.parent.postMessage('Hello Parent Frame!', '*');
        }
    </script>
</head>
<title> IFrame Test </title>
<body>
    <h1> Welcome </h1>
    <p> Hello There </body>
<button onclick="send()">Send</button>
</body>
</html>
```



Output:



5.3 Rollover

Rollover means a webpage changes when the user moves his or her mouse over an object on the page. It is often used in advertising. There are two ways to create rollover, using plain HTML or using a mixture of JavaScript and HTML. We will demonstrate the creation of rollovers using both methods.

The keyword that is used to create rollover is the <onmouseover> event.

For example, we want to create a rollover text that appears in a text area. The text “*What is rollover?*” appears when the user place his or her mouse over the text area and the rollover text changes to “*Rollover means a webpage changes when the user moves his or her mouse over an object on the page*” when the user moves his or her mouse away from the text area.

The HTML script is shown in the following example:

```
<html>
<head></head>
```



```
<Body>
<textarea rows="2" cols="50" name="rollovertext" onmouseover="this.value='What is
rollover?'" 
onmouseout="this.value='Rollover means a webpage changes when the user moves his or
her mouse over an object on the page'"></textarea>
</body>
</html>
```

In following example, we create a rollover effect that can change the color of its text using the style attribute.

```
<p
onmouseover="this.style.color='red'"
onmouseout="this.style.color='blue'">
Move the mouse over this text to change its color to red. Move the mouse away to
change the text color to blue.
</p>
```

Using an iFrame as Link Target

An iframe can also be used as a target for the hyperlinks.

An iframe can be named using the name attribute. This implies that when a link with a target attribute with that name as value is clicked, the linked resource will open in that iframe.

Example:

```
<html>
<body>

<h2>Iframe – Target for a Link</h2>

<iframe src="demo_iframe.htm" name="myframe" height="300px" width="100%" 
title="Iframe Example using Link"></iframe>
```



```
<p><a href="https://vpt.edu.in" target="myframe">Vidyalankar Website</a></p>
```

```
<p>When the target attribute of a link matches the name of an iframe, the link will open in  
the iframe.</p>
```

```
</body>  
</html>
```

Output:

Iframe - Target for a Link

This page is displayed in an iframe

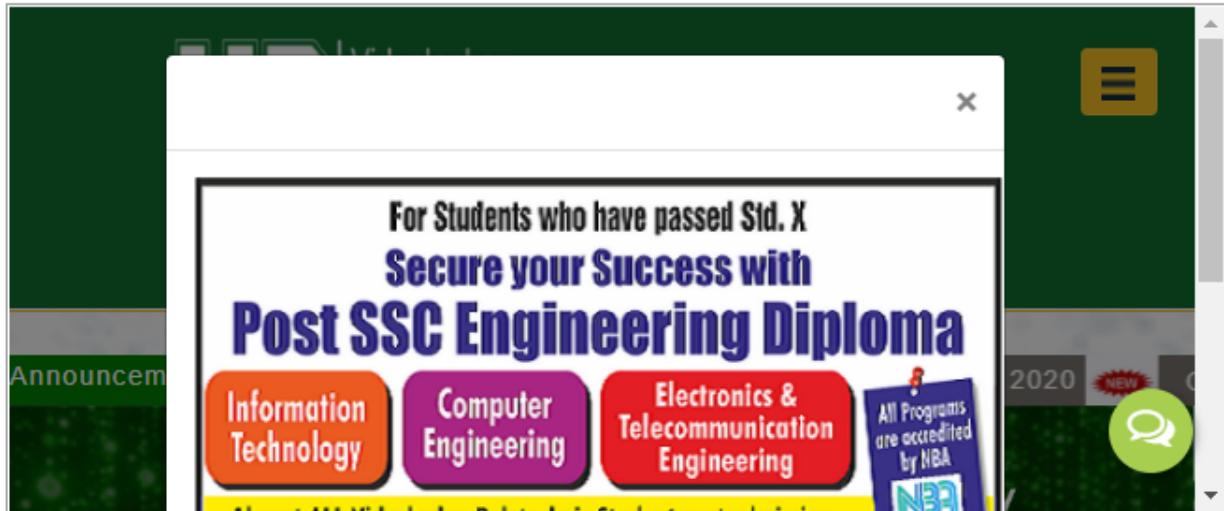
[Vidyalankar Website](https://vpt.edu.in)

When the target attribute of a link matches the name of an iframe, the link will open in the iframe.

After clicking on Vidyalankar Website, vpt.edu.in will open



Iframe - Target for a Link



[Vidyalankar Website](#)

When the target attribute of a link matches the name of an iframe, the link will open in the iframe.



5.3.1 Creating Rollover

Though HTML can be used to create rollovers, it can only perform simple actions. If you wish to create more powerful rollovers, you need to use JavaScript. To create rollovers in JavaScript, we need to create a JavaScript function.

Code: Changing image using onmouseover and onmouseout (rollover and rollback)

```
<html>
<head>
<title>JavaScript Image Rollovers</title>
</head>
<body>

</img>
</body>
</html>
```

5.3.2 Text Rollover

We can also create a rollover and rollback for text using the **onmouseover** and **onmouseout**.

Code: Changing image (rollover and rollback)

```
<html>
<head>
<title>
text rollovers</title>
</head>
<body>
<table border="1" width="100%">
<tbody>
<tr valign="top">
```



```
<td width="50%">
<a></a></td>
<td><a onmouseover="document.clr.src='blue.png'">
<b><u> Blue Color</u></b></a>
<br>
<a onmouseover="document.clr.src='red.png'">
<b><u> Red Color</u></b></a>
<br>
<a onmouseover="document.clr.src='green.png'">
<b><u> Green Color</u></b></a>
</td>
</tr>
</tbody>
</table>
</body>
</html>
```

Output:

| | |
|--|--|
| | <p><u>Blue Color</u> <u>Red Color</u> <u>Green Color</u></p> |
|--|--|



5.3.3 Multiple actions for a Rollover

User want to change the image as well as its description on the web page whenever the user moves the mouse cursor over the image.

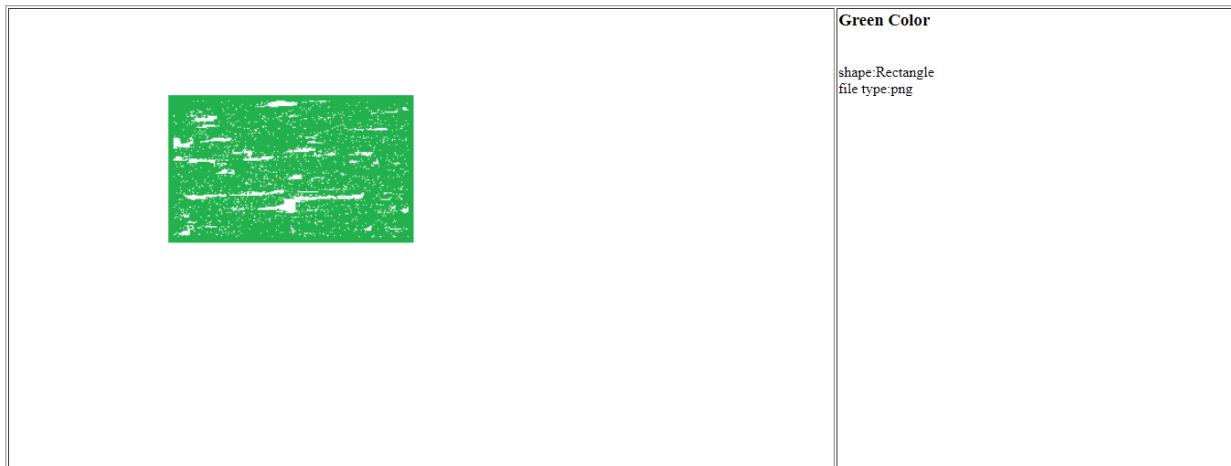
Code: change image and its description.

```
<html> <head>
<title>
text rollovers</title>
<script>
function update_details()
{
document.getElementById("i1").src="blue.png";
document.getElementById("detail").innerHTML=<h3> Blue Color </h3> <br> shape:Circle
<br> file type:png";
}
</script>
</head>
<body>
<table id="t1" border="1" width="100%">
<tbody>
<tr valign="top">
<td width="50%">
<a></a></td>
<td id="detail">
<h3> Green Color </h3>
<br> shape:Rectangle
<br> file type:png
</td>
</tr>
```

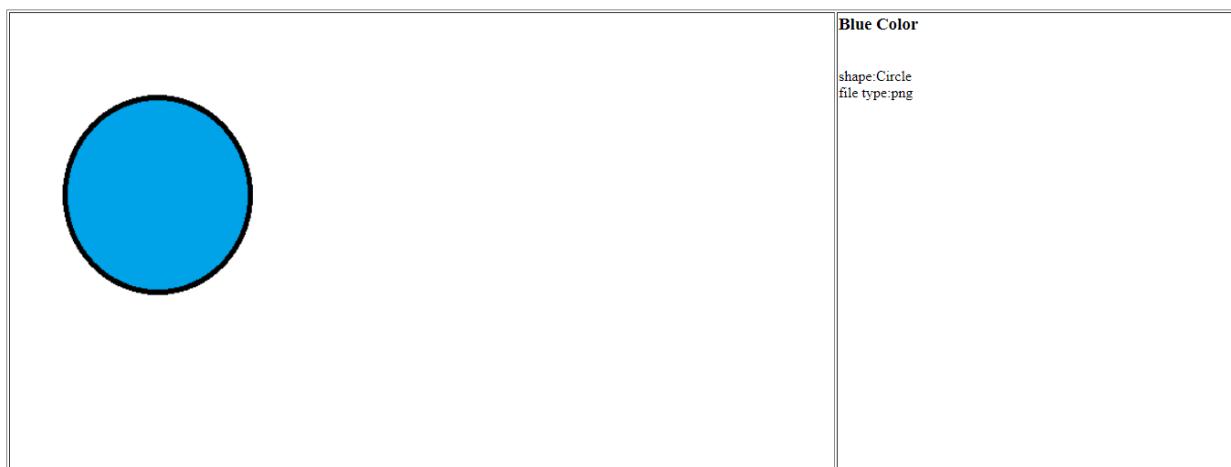


```
</tbody>
</table>
</body>
</html>
```

Output:



When user moves mouse over an image, description and image will change.



Code: Multiple rollover and rollback.



```
<html>
<head>
<title>
text rollovers</title>
<script>
function open_new_window(clrname)
{
if(clrname==1)
{
document.clr.src="red.png";
mwin=window.open("",myadwin,"height=100,width=150,left=500,top=200");
mwin.document.write("looks like red color");
}
if(clrname==2)
{
document.clr.src="green.png";
mwin=window.open("",myadwin,"height=100,width=150,left=500,top=200");
mwin.document.write("looks like green color");
}

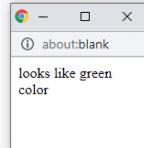
if(clrname==3)
{
document.clr.src="blue.png";
mwin=window.open("",myadwin,"height=100,width=150,left=500,top=200");
mwin.document.write("looks like blue color");
}
}
</script>
</head>
<body>
```

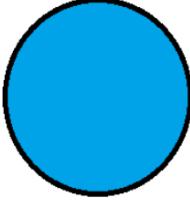
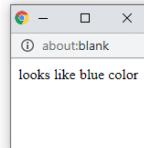


```
<table border="1" width="100%">
<tbody>
<tr valign="top">
<td width="50%">
<a></a></td>
<td><H2>
<a onmouseover="open_new_window(1)" onmouseout="mwin.close()">
<b><u>RED</u></b></a>
<br><br>
<a onmouseover="open_new_window(2)" onmouseout="mwin.close()">
<b><u>GREEN</u></b></a>
<br><br>
<a onmouseover="open_new_window(3)" onmouseout="mwin.close()">
<b><u>BLUE</u></b></a>
</H2>
</td>
</tr>
</tbody>
</table>
</body>
</html>
```

Output:



| | | |
|---|---|---|
|  |  | <u>RED</u> <u>GREEN</u> <u>BLUE</u> |
|---|---|---|

| | | |
|--|---|---|
|  |  | <u>RED</u> <u>GREEN</u> <u>BLUE</u> |
|--|---|---|

5.3.4 More efficient Rollover

We can avoid the delay by loading all images once when they are referred first time on web page.

Code:

```
<html>
<head>
<title>
```



```
text rollovers</title>
<script>
b=new Image;
r=new Image;
g=new Image;
if(document.images)
{
b.src='blue.png';
r.src='red.png';
g.src='green.png';
}
else
{
b.src="";
r.src="";
g.src="";
document.clr="";
}

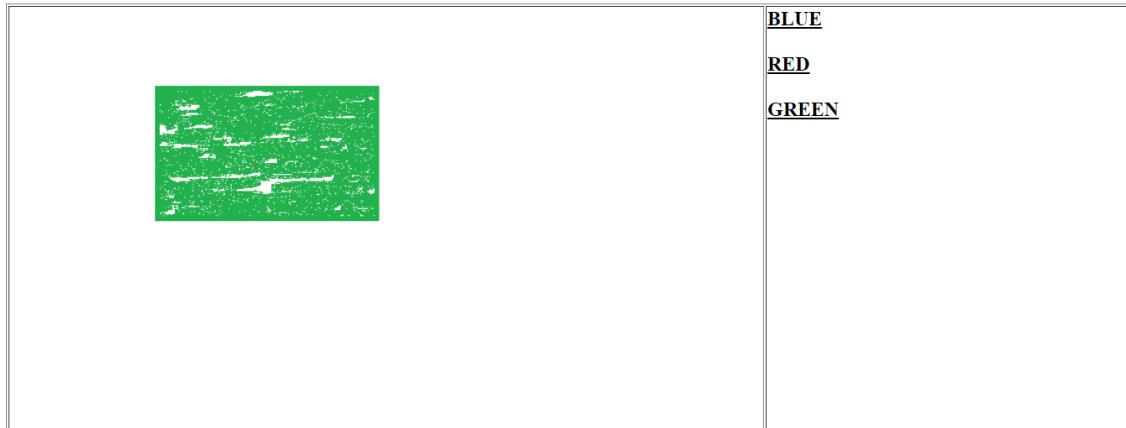
</script>
</head>
<body>
<table border="1" width="100%">
<tbody>
<tr valign="top">
<td width="50%">
<a></a></td>
<td><H2>
<a onmouseover="document.clr.src='blue.png'">
<b><u>BLUE</u></b></a>
```



```
<br><br>
<a onmouseover="document.clr.src='red.png'">
<b><u>RED</u></b></a>
<br><br>
<a onmouseover="document.clr.src='green.png'">
<b><u>GREEN</u></b></a>
</H2>
</td>
</tr>
</tbody>
</table>
</body>
</html>
```

Output:

| | |
|--|---|
|  | <u>BLUE</u> <u>RED</u> <u>GREEN</u> |
|--|---|



BLUE

RED

GREEN

```
<html>
<head>
<title>JavaScript Regular expression to valid an email address</title>
</head>
<body>
<script>
function valid_email(str)
{
```



```
var mailformat = /^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$/;

if(mailformat.test(str))
{
    alert("Valid email address!");
}
else
{
    alert("You have entered an invalid email address!");
}

valid_email('yogita.jore@gmail.com');

</script>
</body>
</html>
```



Unit 6

Menus, Navigation and Web Page Protection

Marks: 14 (R-2, U-4, A-6)

Course Outcome: Create menus and navigations in web pages.

Unit Outcome:

- 1) Develop Javascript to manage the given status bar.
- 2) Develop javascript to create the given banner.
- 3) Develop javascript to create the given slideshow.
- 4) Develop javascript to create the given menu.
- 5) Develop javascript to protect a webpage in a specified manner.

Topics and Sub-topics:

- 6.1 Status bar-builds a static message, changing the message using rollover, moving the message using rollover
- 6.2 Banner-loading and displaying banner advertisement. Linking a banner advisement to url
- 6.3 Slide show – creating a slideshow
- 6.4 Menus-creating a pulldown menu, dynamically changing a menu, validating a menu selection, Floating menu, chain select menu, Tab menu, Pop-up menu, sliding menu, Highlighted menu, Folding a tree menu, context menu, scrollable menu, side bar menu
- 6.5 Protective web page-Hiding your code, disabling the right mouse button, javascript, concealing e-mail address
- 6.6 Frameworks of javascript and its application.

| | | |
|--|--|--|
| | | |
|--|--|--|



6.1. Status Bar

The **status** property of the Window interface was originally intended to set the text in the status bar at the bottom of the browser window. However, the HTML standard now requires setting `window.status` to have no effect on the text displayed in the status bar.

Syntax:

```
window.status = string;  
var value = window.status;
```

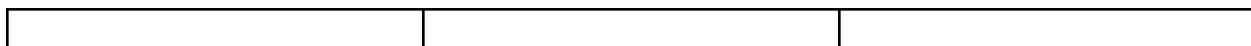
```
<html>  
<head>  
<title>JavaScript Status Bar</title></head>  
<body>  
  <a href="http://www.vpt.edu.in"  
    onMouseOver="window.status='Vidyalankar';return true"  
    onMouseOut="window.status='';return true">  
    Vidyalankar  
  </a>  
</body>  
</html>
```

Output:

A screenshot of a web browser window. The address bar shows "www.vpt.edu.in". The main content area displays the text "Vidyalankar" in blue, underlined font. At the bottom of the browser window, the status bar shows the text "Vidyalankar". The browser interface includes standard navigation buttons (back, forward, search), a file menu, and various toolbar icons.

Note: "window.status" does not support any browser.

6.2 Banner





The banner advertisement is the hallmark of every commercial web page. It is typically positioned near the top of the web page, and its purpose is to get the visitor's attention by doing all sorts of clever things.

Nearly all banner advertisements are in a file format such as a GIF, JPG, TIFF, or other common graphic file format. Some are animated GIFs, which is a series of images contained in one file that rotate automatically on the screen. Some are Flash movies that require the visitor to have a browser that includes a Flash plug-in. Many banner advertisements consist of a single graphical image that does not contain any animation and does not require any special plug-in.

You need to do three things to incorporate a banner advertisement in your web page:

1. Create several banner advertisements using a graphics tool such as Photoshop. You'll want to make more than one advertisement so you can rotate them on your web page using a JavaScript.
2. Create an `` element in your web page with the height and width necessary to display the banner advertisement.
3. Build a JavaScript that loads and displays the banner advertisements in conjunction with the `` element.

6.2.1 Loading and Displaying Banner Advertisements

The banners should all be the same size so they look professional as they rotate on your web page. The best way to do this is to create an empty banner and then copy it for each banner advertisement that you want to build. This assures that all the banners will be the same size.

You can then use each copy to design each ad. Next, create an image element on your web page using the `` tag. You'll need to set four attributes of the `` tag: `src`, `width`, `height`, and `name`. Set

the `src` attribute to the file name of the first banner advertisement that you want to display. Set the `width` and `height` attributes to the width and height of the banner. Set the `name` attribute to a unique name for the image element. You'll be using the `name` attribute in the JavaScript when you change from one banner to the next. The image element (banner) should be centered in the page using the `<center>` tag within the `<body>` tag of your web page. The final step is to build the JavaScript that will rotate the banners on your web page. You'll define the JavaScript in the `<head>` tag of the web page.

The JavaScript must do the following to load the banner:

1. Load banner advertisements into an array.
2. Determine whether the browser supports the image object.
3. Display a banner advertisement.
4. Pause before displaying the next banner advertisement.

You load the banner advertisements into an array by declaring an `Array()` object and initializing it with the file name of each banner advertisement. For example, suppose you have three banner advertisements that are contained in the `1.jpg`, `2.jpg`, and `3.jpg` files. Here's how you'd load them into an `Array()` object:

```
Banners = new Array('1.jpg','2.jpg','3.jpg')
```

| | | |
|--|--|--|
| | | |
|--|--|--|



Example:

```
<html>
<head>
<title>Banner Ads</title>
<script>
Banners = new Array('1.jpg','2.jpg','3.jpg');
CurrentBanner = 0;
function DisplayBanners()
{
if (document.images);
{
CurrentBanner++;
if (CurrentBanner == Banners.length)
{
CurrentBanner = 0;
}
document.RotateBanner.src= Banners[CurrentBanner];
setTimeout("DisplayBanners()",1000);
}
}
</script>
</head>
<body onload="DisplayBanners()">
<center>

</center>
</body>
</html>
```

6.2.2 Linking Banner Advertisements to URLs

A banner advertisement is designed to encourage the visitor to learn more information about a product or service that is being advertised. To get additional information, the visitor is expected to click the banner so that a new web page opens. You can link a banner advertisement to a web page by inserting a hyperlink into your web page that calls a JavaScript function rather than the URL of a web page. The JavaScript then determines the URL that is associated with the current banner and loads the web page that is associated with the URL.

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<html>
<head>
<title>Link Banner Ads</title>
<script language="Javascript" type="text/javascript">

Banners = new Array('1.jpg","2.jpg","3.jpg')
BannerLink = new Array(
'google.com/"vpt.edu.in/', 'msbte.org.in/');
CurrentBanner = 0;
NumOfBanners = Banners.length;
function LinkBanner()
{
document.location.href =
"http://www." + BannerLink[CurrentBanner];
}
function DisplayBanners() {
if (document.images) {
CurrentBanner++
if (CurrentBanner == NumOfBanners) {
CurrentBanner = 0
}
document.RotateBanner.src= Banners[CurrentBanner]
setTimeout("DisplayBanners()",1000)
}
}
</script>
</head>
<body onload="DisplayBanners()" >
<center>
<a href="javascript: LinkBanner()"></a>
</center>
</body>
</html>
```

6.3 Slideshow:

A slideshow is similar in concept to a banner advertisement in that a slideshow rotates multiple images on the web page. However, unlike a banner advertisement, a slideshow

| | | |
|--|--|--|
| | | |
|--|--|--|



gives the visitor the ability to change the image that's displayed: the visitor can click the Forward button to see the next image and the Back button.

Creating a slideshow:

First, set the **slideIndex** to 1. (First picture)

Then call **showDivs()** to display the first image.

When the user clicks one of the buttons call **plusDivs()**.

The plusDivs() function **subtracts** one or **adds** one to the slideIndex.

The **showDivs()** function hides (**display="none"**) all elements with the class name "mySlides", and displays (**display="block"**) the element with the given slideIndex.

If the slideIndex is **higher than** the number of elements (x.length), the slideIndex is set to zero.

If the slideIndex is **less than** 1 it is set to number of elements (x.length).

```
<html>
<title>slideshow</title>
<body>
<h2 class="w3-center">Manual Slideshow</h2>
<div class="w3">
  
  
  
  

  <button class="aa" onclick="plusDivs(-1)">Back</button>
  <button class="bb" onclick="plusDivs(1)">Forward</button>
</div>

<script>
var slideIndex = 1;
showDivs(slideIndex);

function plusDivs(n)
{
  showDivs(slideIndex += n);
}

function showDivs(n)
{
  var i;
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
var x = document.getElementsByClassName("mySlides");
if (n > x.length)
{
    slideIndex = 1
}

if (n < 1)
{
    slideIndex = x.length
}
for (i = 0; i < x.length; i++)
{
    x[i].style.display = "none";
}
x[slideIndex-1].style.display = "block";
}
</script>
</body>
</html>
```

Automatic Slideshow:

```
<html>
<head>
<title>Automatic Slideshow</title>
</head>
<body>
<h2 class="aa">Automatic Slideshow</h2>
<div class="aa" style="max-width:500px">
    
    
    
    
</div>
<script>
var myIndex = 0;
auto_slide_show();
```



```
function auto_slide_show()
{
    var i;
    var x = document.getElementsByClassName("mySlides");
    for (i = 0; i < x.length; i++) {
        x[i].style.display = "none";
    }
    myIndex++;
    if (myIndex > x.length) {myIndex = 1}
    x[myIndex-1].style.display = "block";
    setTimeout(auto_slide_show, 2000); // Change image every 2 seconds
}
</script>
</body>
</html>
```

6.4 Menus:

A menu consists of a set of options which are presented to the user. Menus are common in graphical user interface such as Windows or Mac OS.

The Menu component provides the pull-down menu element that's common in most graphical user interfaces (GUIs). Using a familiar GUI element will reduce the interface learning curve of your web site or application for new users, as well as help all users more easily find what they're looking for. Having menus that contain links to sections at various levels in your web site can improve both the navigation of the site and the real estate of your web pages.

6.4.1 Creating pull-down menu:

- Also known as drop-down menus.
- Clicking a menu title causes the menu items to appear to drop down from that position and be displayed.

Code: Create a simple menu: If user not selected any menu, alert should be displayed.

Select Fruit:

```
<select id="ddlFruits">
<option value=""></option>
<option value="1">Apple</option>
```



```
<option value="2">Mango</option>
<option value="3">Orange</option>
</select>
<input type="submit" value="Validate" onclick="return Validate()" />
<script type="text/javascript">
    function Validate()
{
    var ddlFruits = document.getElementById("ddlFruits");
    if (ddlFruits.value == "") {
        //If the "Please Select" option is selected display error.
        alert("Please select an option!");
        return false;
    }
    return true;
}
</script>
```

Code: Design a drop-down menu for various colours. After selecting any colour from menu, background colour should be changed.

- 1) demo2.html
- 2) demo2-script.js
- 3) demo2-style.css

HTML file:

```
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Demo 2</title>
    <link rel="stylesheet" href="demo2-style.css">
</head>
<body>
    <div class="container">
        <label for="color">Choose a Background Color:</label>
        <select name="color" id="color" class="color" onchange="changeColor()">
            <option value="white">White</option>
            <option value="black">Black</option>
```



```
<option value="blue">Blue</option>
<option value="red">Red</option>
<option value="yellow">Yellow</option>
</select>
</div>
<script type="text/javascript" src="demo2-script.js"></script>
</body>
</html>
```

JavaScript file:

```
changeColor = () => {
    var color = document.getElementById("color").value;
    switch(color){
        case "white":
            document.body.style.backgroundColor = "white";
            document.body.style.color = "black";
            document.getElementById("color").style.backgroundColor = "white";
            document.getElementById("color").style.color = "black";
            break;
        case "black":
            document.body.style.backgroundColor = "black";
            document.body.style.color = "white";
            document.getElementById("color").style.backgroundColor = "black";
            document.getElementById("color").style.color = "white";
            break;
        case "blue":
            document.body.style.backgroundColor = "blue";
            document.body.style.color = "white";
            document.getElementById("color").style.backgroundColor = "blue";
            document.getElementById("color").style.color = "white";
            break;
        case "red":
            document.body.style.backgroundColor = "red";
            document.body.style.color = "white";
            document.getElementById("color").style.backgroundColor = "red";
```



```
document.getElementById("color").style.color = "white";
break;
case "yellow":
    document.body.style.backgroundColor = "yellow";
    document.body.style.color = "black";
    document.getElementById("color").style.backgroundColor = "yellow";
    document.getElementById("color").style.color = "black";
    break;
default:
    document.body.style.backgroundColor = "white";
    document.body.style.color = "black";
    document.getElementById("color").style.backgroundColor = "white";
    document.getElementById("color").style.color = "black";
    break;
}
}
```

CSS file:

```
* {
    box-sizing: border-box;
}

body {
    font-family: "Calibri", "Roboto", sans-serif;
    -ms-overflow-style: none; /* IE and Edge */
    scrollbar-width: none; /* Firefox */
}

body::-webkit-scrollbar {
    display: none;
}

.container{
    margin: 10%;
    text-align: center;
}

.color{
    width: 30%;
```



```
outline: none;  
height: 30px;  
background: transparent;  
}
```

6.4.2 Dynamically changing a Menu:

Code: Following example provides two radio buttons to the user one is for fruits and another is for vegetables.

When user will select the fruits radio button, the option list should present only the fruits names to user and when user will select the vegetable radio button, the option list should present only the vegetable names to user so that user can select an appropriate element of interest.

```
<html>  
<body>  
<html>  
<script type="text/javascript">  
function modifyList(x)  
{  
with(document.forms.myform)  
{  
if(x ==1)  
{  
optionList[0].text="Kiwi";  
optionList[0].value=1;  
optionList[1].text="Pine-Apple ";  
optionList[1].value=2;  
optionList[2].text="Apple";  
optionList[2].value=3;  
}  
  
if(x ==2)  
{  
optionList[0].text="Tomato";
```



```
optionList[0].value=1;
optionList[1].text="Onion ";
optionList[1].value=2;
optionList[2].text="Cabbage ";
optionList[2].value=3;
}

}

}

</script>
</head>
</body>
<form name="myform" action=" " method="post">
<select name="optionList" size="3">
<option value=1>Kiwi
<option value=1>Pine-Apple
<option value=1>Apple
</select>
<br>
<input type="radio" name="grp1" value=1 checked="true"
onclick="modifyList(this.value)"> Fruits

<input type="radio" name="grp1" value=2 onclick="modifyList(this.value)"> Vegetables
</form>
</body>
</html>
```

Output:

Kiwi
Pine-Apple
Apple

Fruits Vegetables

Tomato
Onion
Cabbage

Fruits Vegetables



6.4.3 Validating Menu Selections:

Code: Following example provides four list elements as name of branches. When you select a branch from list, selected branch will be displayed as output.

```
<html>
<body>
<p>Select Program from list:</p>
<select id="mySelect" onchange="myFunction()">
    <option value="CO">Computer Engg</option>
    <option value="IF">Information Technology</option>
    <option value="EJ">Electronics and Tele</option>
    <option value="CE">Chemical Engg</option>
</select>
<p id="demo"></p>
<script>
function myFunction()
{
    var x = document.getElementById("mySelect").value;
    document.getElementById("demo").innerHTML = "You selected: " + x;
}
</script>
</body>
</html>
```

Output:

Select Program from list:

You selected: IF

Code:

```
<html>
<script language="Javascript">
function validate()
{
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
if(document.form.city.selectedIndex=="")  
{  
    alert ( "Please select city!");  
    return false;  
}  
  
var sel = document.getElementById("city");  
//get the selected option  
var selectedText = sel.options[sel.selectedIndex].text;  
alert("You have selected : "+selectedText);  
return true;  
}  
</script>  
<form name="form" method="post" onSubmit="return validate()"><pre>  
Select your City <select name="city" id="city">  
    <option value="Select">Select</option>  
    <option value="Delhi">Delhi</option>  
    <option value="Jaipur">Jaipur</option>  
    <option value="Agra">Agra</option>  
    <option value="Bangalore">Bangalore</option>  
    <option value="Pune">Pune</option>  
</select>  
<input type="submit" name="Submit" value="Submit">  
</pre></form>  
</html>
```

Output:

Select your City

Submit

Jaipur

Select

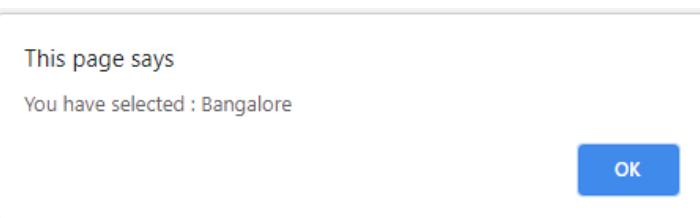
Delhi

Jaipur

Agra

Bangalore

Pune





6.4.4. Floating Menu

Also known as "fixed menus" and "hovering menus", floating menus stay in a fixed position when you scroll the page. They appear to "float" on top of the page as you scroll.

Code:

```
<html>
<title>Example</title>
<style>
body {
    background-image: url('/pix/samples/bg1.gif');
}
main {
    margin-bottom: 200px;
}
.floating-menu {
    font-family: sans-serif;
    background: yellowgreen;
    padding: 5px;
    width: 130px;
    z-index: 100;
    position: fixed;
}
.floating-menu a,
.floating-menu h3 {
    font-size: 0.9em;
    display: block;
    margin: 0 0.5em;
    color: white;
}
</style>
<main>
<p>Scroll down and watch the menu remain fixed in the same position, as though it was floating.</p>
<nav class="floating-menu">
    <h3>Floating Menu</h3>
```

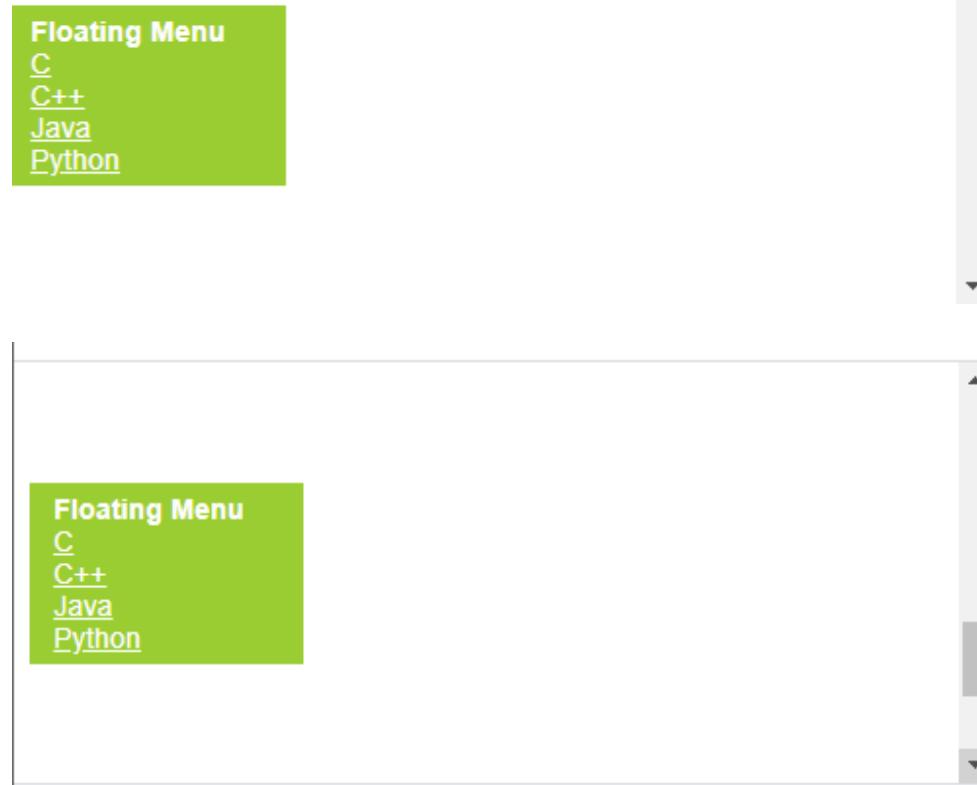


```
<a href="c_sub.txt">C</a>
<a href="C++_sub.txt">C++</a>
<a href="java_sub.txt">Java</a>
<a href="python_sub.txt">Python</a>
</nav>
</main>
```

Output: scroll down the page to observe the output:

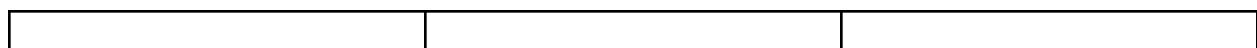


Scroll down and watch the menu remain fixed in the same position, as though it was floating.



6.4.5. Chain Select Menu

Chained selects menu lets you “chain” multiple form selects list together so that the selection in a “parent” list can tailor the options available in a “child” list.





Code:

```
<html>
<head><title>chained menu</title></head>
<script>
var stateObject = {
    "Maharashtra": {
        "Mumbai": ["Wadala", "Nerul"],
        "Pune": ["Aundh", "Kothrud"]
    },
    "Karnataka": {
        "Banglore": ["Mysoor", "Ooty"],
    }
}
window.onload = function ()
{
    var aaa = document.getElementById("aaa"),
        bbb = document.getElementById("bbb"),
        ccc = document.getElementById("ccc");
    for (var state in stateObject)
    {
        aaa.options[aaa.options.length] = new Option(state, state);
    }
    aaa.onchange = function ()
    {
        bbb.length = 1; // remove all options bar first
        ccc.length = 1; // remove all options bar first
        if (this.selectedIndex < 1) {
            bbb.options[0].text = "Please select city first"
            ccc.options[0].text = "Please select area first"
            return; // done
        }
        bbb.options[0].text = "Please select city"
        for (var citi_name in stateObject[this.value]) {
            bbb.options[bbb.options.length] = new Option(citi_name, citi_name);
        }
    }
}
```



```
        }
        if (bbb.options.length==2)
        {
            bbb.selectedIndex=1;
            bbb.onchange();
        }
    }

aaa.onchange(); // reset in case page is reloaded
bbb.onchange = function ()
{
    ccc.length = 1; // remove all options bar first
    if (this.selectedIndex < 1)
    {
        ccc.options[0].text = "Please select area first"
        return; // done
    }
    ccc.options[0].text = "Please select area first"
    var cities = stateObject[aaa.value][this.value];
    for (var i = 0; i < cities.length; i++) {
        ccc.options[ccc.options.length] = new Option(cities[i], cities[i]);
    }
    if (ccc.options.length==2)
    {
        ccc.selectedIndex=1;
        ccc.onchange();
    }
}
}

</script>
</body>
<form name="myform" id="myForm">
<select name="optone" id="aaa" size="1">
    <option value="" selected="selected">Select state</option>
</select>
<br>
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<br>
<select name="opttwo" id="bbb" size="1">
    <option value="" selected="selected">Please select city first</option>
</select>
<br>
<br>
<select name="optthree" id="ccc" size="1">
    <option value="" selected="selected">Please select area first</option>
</select>
</form>
</body>
</html>
```

Output:

| | | |
|----------------------------|--|---|
| Select state ▾ | Maharashtra ▾ | Mumbai ▾ |
| Please select city first ▾ | Please select city ▾ Please select city Mumbai Pune | Please select area first ▾ Please select area first Wadala Nerul |
| Karnataka ▾ | Banglore ▾ | Please select area first ▾ Please select area first Mysoor Ooty |



6.4.6 Tab Menu

Using tab menu, more complete description is displayed below the tab bar as the visitor clicks the mouse cursor over the tabs.

2-ways to create tab menu:

- a) Using button
- b) Using target selector

Code: In following example, created 3 buttons using <button>

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {font-family: Arial; }

/* Style the tab */
.tab {
    overflow: hidden;
    border: 1px solid #ccc;
    background-color: #f1f1f1;
}

/* Style the buttons inside the tab */
.tab button {
    background-color: inherit;
    float: left;
    border: none;
    outline: none;
    cursor: pointer;
    padding: 14px 16px;
    transition: 0.3s;
    font-size: 17px;
}

/* Change background color of buttons on hover */

```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
.tab button:hover {  
    background-color: #ddd;  
}  
  
/* Create an active/current tablink class */  
.tab button.active {  
    background-color: #ccc;  
}  
  
/* Style the tab content */  
.tabcontent {  
    display: none;  
    padding: 6px 12px;  
    border: 1px solid #ccc;  
    border-top: none;  
}  
</style>  
</head>  
<body>  
  
<h2>Tabs using button </h2>  
<p>Click on the buttons inside the tabbed menu:</p>  
  
<div class="tab">  
    <button class="tablinks" onclick="openCity(event, 'Mumbai')">Mumbai</button>  
    <button class="tablinks" onclick="openCity(event, 'Bhopal')">Bhopal</button>  
    <button class="tablinks" onclick="openCity(event, 'Panaji')">Panaji</button>  
</div>  
  
<div id="Mumbai" class="tabcontent">  
    <h3>Mumbai</h3>  
    <p>Mumbai is the capital city of Maharashtra.</p>  
</div>  
  
<div id="Bhopal" class="tabcontent">
```



```
<h3>Bhopal</h3>
<p>Bhopal is the capital of MadhyaPradesh.</p>
</div>
```

```
<div id="Panaji" class="tabcontent">
  <h3>Panaji</h3>
  <p>Panajiis the capital of Goa.</p>
</div>
<script>
function openCity(evt, cityName)
{
  var i, tabcontent, tablinks;
  tabcontent = document.getElementsByClassName("tabcontent");
  for (i = 0; i < tabcontent.length; i++)
  {
    tabcontent[i].style.display = "none";
  }
  tablinks = document.getElementsByClassName("tablinks");
  for (i = 0; i < tablinks.length; i++)
  {
    tablinks[i].className = tablinks[i].className.replace(" active", "");
  }
  document.getElementById(cityName).style.display = "block";
  evt.currentTarget.className += " active";
}
</script>
</body>
</html>
```



Output:

Tabs using button

Click on the buttons inside the tabbed menu:

| | | |
|--|--------|--------|
| Mumbai | Bhopal | Panaji |
| Bhopal Bhopal is the capital of MadhyaPradesh. | | |

Tabs using button

Click on the buttons inside the tabbed menu:

| | | | |
|---|--------|--------|--|
| Mumbai | Bhopal | Panaji | |
| Panaji Panajiis the capital of Goa. | | | |

Code: Following example shows how to create tab menu by using target selector<a>.

```
<html>
<head>
<style>
:target
{
    color:white;
    border: 2px solid #F4D444;
    background-color:green;
}
</style>
</head>
<body>
<p><a href="#news1">Mumbai is capital of Maharashtra.</a></p>
<p><a href="#news2">Bhopal is capital of Madhyapradesh.</a></p>
<p>Click on the links above and the :target selector highlight the current active HTML anchor.</p>
<h3>
<p id="news1"><b>Mumbai</b></p>
<p id="news2"><b>Bhopal</b></p>
</h3>
</body>
</html>
```



Output:

[Mumbai is capital of Maharashtra.](#)

[Bhopal is capital of Madhyapradesh.](#)

Click on the links above and the :target selector highlight the current active HTML anchor.

Mumbai

Bhopal

[Mumbai is capital of Maharashtra.](#)

[Bhopal is capital of Madhyapradesh.](#)

Click on the links above and the :target selector highlight the current active HTML anchor.

Mumbai

Bhopal

Code:

```
<html>
<head>
<style>
.tab div {
    display: none;
}

.tab div:target {
    display: block;
}
</style>
</head>
<body>

<div class="tab">
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<a href="#link1">Link 1</a>
<a href="#link2">Link 2</a>
<a href="#link3">Link 3</a>

<div id="link1">
  <h3>Content to Link 1</h3>
  <p>Hello World!</p>
</div>

<div id="link2">
  <h3>Content to Link 2</h3>
  <h4>Great success!</h4>
</div>

<div id="link3">
  <h3>Content to Link 3</h3>
  <p>Yeah!</p>
</div>

</div>

</body>
</html>
```

Output:

| | | |
|--|--|-------|
| Link 1 Link 2 Link 3 | Link 1 Link 2 Link 3 | |
| Content to Link 2 Content to Link 3 | | |
| Link 1 Link 2 Link 3 | Great success! | Yeah! |

| | | |
|--|--|--|
| | | |
|--|--|--|



6.4.7. Popup Menu:

A popup menu appears as the user moves the mouse cursor over a parent menu item. The popup menu contains child menu items that are associated with the parent menu item.

Code:

```
<html>
<head>
<style>
.dropbtn {
    background-color: Blue;
    color: white;
    padding: 16px;
    font-size: 16px;
    border: none;
}
.dropdown {
    position: relative;
    display: inline-block;
}

.dropdown-content {
    display: none;
    position: absolute;
    background-color: red;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
}

.dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}
```

| | | |
|--|--|--|
| | | |
|--|--|--|



{}

```
.dropdown-content a:hover {background-color: #ddd;}

.dropdown:hover .dropdown-content {display: block;}

.dropdown:hover .dropbtn {background-color: #3e8e41;}

</style>
</head>
<body>
<h2>Hoverable Dropdown</h2>
<p>Move the mouse over the button to open the dropdown menu.</p>
<div class="dropdown">
  <button class="dropbtn">Programs:</button>
  <div class="dropdown-content">
    <a href="#">CO</a>
    <a href="#">IF</a>
    <a href="#">EJ</a>
  </div>
</div>
</body>
</html>
```

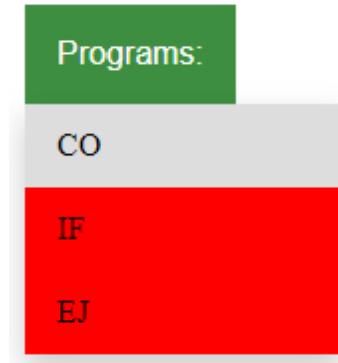
Output:

| | | |
|--|--|--|
| | | |
|--|--|--|



Hoverable Dropdown

Move the mouse over the button to open the dropdown menu.



6.4.8. Sliding Menu:

The slide-in menu appears as a block that floats on the left /right side of the web page. It seems to come alive when the user moves the mouse over the block.

Code:

```
<html>
<head>
<style>
#menu
{
    position: fixed;
    right: -8.5em;
    top: 20%;
    width: 8em;
    background: pink;
    color: red;
    margin: -1;
    padding: 0.5em 0.5em 0.5em 2.5em;
}
#menu:hover
{
    right: 0
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
}

#menu
{
transition: 0.2s
}

#menu a
{
position: relative;
left: 0;
}

#menu a:focus
{
left: -7em;

}

#menu a { transition: 0.1s }
#menu:hover a:focus {
left: 0;
background: none;
}

</style>
</head>
<body>
<h3>
<ul id=menu>
<li><a href="#home">Home</a>
<li><a href="#prog">Programs</a>
<li><a href="#vision">Vision</a>
<li><a href="#mission">Mission</a>
</ul>
</div>
</body>
</html>
```



6.4.9. Highlighted Menu:

User can highlight menu by using following methods:

- 1) When user performs onmouseover()
- 2) When user performs onclick()

Code:

```
<html>
<head>
<title>Highlighted Menu Effect</title>
<style>
.link
{
text-decoration: none;
padding: 10px 16px;
background-color:pink;
font-size: 20px;
}

.active, .link:hover
{
background-color:gray;
color:white;
}
</style>
</head>
<body>
Move the mouse over menus:<br><br>
<div id="me">
    <a href="#file" class="link">File</a>
    <a href="#edit" class="link">Edit</a>
    <a href="#view" class="link">View</a>
    <a href="#exit" class="link">Exit</a>
</div>
</body>
</html>
```

Output:

Move the mouse over menus:

File Edit View Exit

6.4.10. Folding Tree Menu:

Also known as cascading tree.

The folding tree menu looks like a tree which consists of one or more closed folders, each of these folders further consist of some menu items.

Code:

```
<html>
<head>
<style>
ul, #myUL {
    list-style-type: none;
}

#myUL {
    margin: 0;
    padding: 0;
}

.caret {
    cursor: pointer;
    -webkit-user-select: none; /* Safari 3.1+ */
    -moz-user-select: none; /* Firefox 2+ */
    -ms-user-select: none; /* IE 10+ */
    user-select: none;
}

.caret::before {
    content: "\25B6";
    color: black;
}
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
display: inline-block;
margin-right: 6px;
}

.caret-down::before {
-ms-transform: rotate(90deg); /* IE 9 */
-webkit-transform: rotate(90deg); /* Safari */
transform: rotate(90deg);
}

.nested {
display: none;
}

.active {
display: block;
}

</style>
</head>
<body>

<h2>Folding Tree Menu</h2>
<p>A tree menu represents a hierarchical view of information, where each item can have a number of subitems.</p>
<p>Click on the arrow(s) to open or close the tree branches.</p>

<ul id="myUL">
<li><span class="caret">India</span>
<ul class="nested">
<li>Karnataka</li>
<li>Tamilnadu</li>
<li><span class="caret">Maharashtra</span>
<ul class="nested">
<li>Mumbai</li>
<li>Pune</li>
```



```
<li><span class="caret">Navi Mumbai</span>
  <ul class="nested">
    <li>Nerul</li>
    <li>Vashi</li>
    <li>Panvel</li>

  </ul>
</li>
</ul>
</li>
</ul>
</li>
</ul>
<script>
var toggler = document.getElementsByClassName("caret");
var i;
for (i = 0; i < toggler.length; i++) {
  toggler[i].addEventListener("click", function() {
    this.parentElement.querySelector(".nested").classList.toggle("active");
    this.classList.toggle("caret-down");
  });
}
</script>

</body>
</html>
```

Output:

| | | |
|--|--|--|
| | | |
|--|--|--|



Tree Menu

A tree menu represents a hierarchical view of information, where each item can have a number of subitems.

Click on the arrow(s) to open or close the tree branches.

- ▼ India
 - Karnataka
 - Tamilnadu
 - ▼ Maharashtra
 - Mumbai
 - Pune
 - ▼ Navi Mumbai
 - Nerul
 - Vashi
 - Panvel

6.4.11. Context Menu:

The context menu appears on the web page when the user clicks the right button on the screen.

Code:

```
<html>
<head>
<style>
div {
    background: yellow;
    border: 1px solid black;
    padding: 10px;
}
</style>
</head>
<body>

<div contextmenu="mymenu">
<p>Right-click inside this box to see the context menu!

<menu type="context" id="mymenu">
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<menuitem label="Refresh" onclick="window.location.reload();"  
icon="ico_reload.png"></menuitem>  
<menu label="Share on...">  
  <menuitem label="Twitter" icon="ico_twitter.png"  
onclick="window.open('//twitter.com/intent/tweet?text=' +  
window.location.href);"></menuitem>  
  <menuitem label="Facebook" icon="ico_facebook.png"  
onclick="window.open('//facebook.com/sharer/sharer.php?u=' +  
window.location.href);"></menuitem>  
</menu>  
<menuitem label="Email This Page"  
onclick="window.location='mailto:?body='+window.location.href;"></menuitem>  
</menu>  
  
</div>  
  
<p>This example currently only works in Firefox!</p>  
  
</body>  
</html>
```

Output:

Right-click inside this box to see the context menu!

This example currently only works in Firefox!

← → ⌂ ☆

Refresh

Share on... >

Email This Page

Twitter

Facebook

6.4.12. Scrollable Menu:

Scrollbar is different from other menu as it provides two arrowheads.

2-ways to implement this type of menu:

- 1) Horizontal Scrollable Menu:

Code:

```
<!DOCTYPE html>
```



```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
div.scrollmenu {
    background-color: #333;
    overflow: auto;
    white-space: nowrap;
}

div.scrollmenu a {
    display: inline-block;
    color: white;
    text-align: center;
    padding: 14px;
    text-decoration: none;
}

div.scrollmenu a:hover {
    background-color: #777;
}
</style>
</head>
<body>

<div class="scrollmenu">
    <a href="#home">Home</a>
    <a href="#news">News</a>
    <a href="#contact">Contact</a>
    <a href="#about">About</a>
    <a href="#support">Support</a>
    <a href="#blog">Blog</a>
    <a href="#tools">Tools</a>
    <a href="#base">Base</a>
    <a href="#custom">Custom</a>

```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<a href="#more">More</a>
<a href="#logo">Logo</a>
<a href="#friends">Friends</a>
<a href="#partners">Partners</a>
<a href="#people">People</a>
<a href="#work">Work</a>
</div>

<h2>Horizontal Scrollable Menu</h2>
<p>Resize the browser window to see the effect.</p>
</body>
</html>
```

Output:



Horizontal Scrollable Menu

Resize the browser window to see the effect.

- 2) Vertical Scrollable Menu:

Code:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
.vertical-menu {
  width: 200px;
  height: 150px;
  overflow-y: auto;
}

.vertical-menu a {
  background-color: #eee;
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
color: black;  
display: block;  
padding: 12px;  
text-decoration: none;  
}  
  
.vertical-menu a:hover {  
background-color: #ccc;  
}  
  
.vertical-menu a.active {  
background-color: #4CAF50;  
color: white;  
}  
</style>  
</head>  
<body>  
<h1>Vertical Scroll Menu</h1>  
<div class="vertical-menu">  
 <a href="#" class="active">Home</a>  
 <a href="#">Link 1</a>  
 <a href="#">Link 2</a>  
 <a href="#">Link 3</a>  
 <a href="#">Link 4</a>  
 <a href="#">Link 5</a>  
 <a href="#">Link 6</a>  
 <a href="#">Link 7</a>  
 <a href="#">Link 8</a>  
 <a href="#">Link 9</a>  
 <a href="#">Link 10</a>  
</div>  
</body>  
</html>
```

Output:

| | | |
|--|--|--|
| | | |
|--|--|--|



Vertical Scroll Menu



6.4.13. Side Bar Menu:

The side bar menu displays a menu on the side of the web page.

Code:

```
<html>
<head>
<style>
.sidebar
{
height: 100%;
width:100px;
position:fixed;
background-color: pink;
padding-top:20px;
}

.sidebar a
{
text-decoration:none;
font-size:20px;
color:red;
display:block;
}

.sidebar a:hover
{
```



```
color:white;
}

.main
{
margin-left:160px;
padding:0px 10px;
}

</style>
</head>
<body>

<div class="sidebar">
<a href="#home"> Home</a>
<a href="#vision">Vision </a>
<a href="#mission"> Mission</a>
<a href="#prog">Programs</a>
</div>

<div class="main">
<h2> Side bar Menu </h2>
</div>
</body>
</html>
```

Output:

A screenshot of a web browser window. On the left, there is a vertical pink sidebar containing a list of links: "Home", "Vision", "Mission", and "Programs", each underlined in red. To the right of the sidebar, the main content area has a grey header with the text "Side bar Menu" in bold black font. The rest of the page is blank white space.

Side bar Menu

[Home](#)
[Vision](#)
[Mission](#)
[Programs](#)

6.5 Protecting your webpage:

| | | |
|--|--|--|
| | | |
|--|--|--|



V2V EDTECH LLP

DIPLOMA | DEGREE | BSCIT

There is nothing secret about your web page. Anyone with a little computer knowledge can use a few mouse clicks to display your HTML code, including your JavaScript, on the screen. In this, you'll learn how to hide your JavaScript and make it difficult for malicious hackers to extract e-mail addresses from your web page.

| | | |
|--|--|--|
| | | |
|--|--|--|



6.5.1 Hiding Your Code

- Every developer has to admit that, on occasion, they've peeked at the code of a web page or two by right-clicking and choosing View Source from the context menu.
- In fact, this technique is a very common way for developers to learn new techniques for writing HTML and Javascript. However, some developers don't appreciate a colleague snooping around their code and then borrowing their work without permission. This is particularly true about javascript, which are typically more time-consuming to develop than using HTML to build a web page.
- In reality, you cannot hide your HTML code and JavaScript from prying eyes, because a clever developer can easily write a program that pretends to be a browser and calls your web page from your web server, saving the web page to disk, where it can then be opened using an editor. Furthermore, the source code for your web page—including your JavaScript—is stored in the cache, the part of computer memory where the browser stores web pages that were requested by the visitor.
- A sophisticated visitor can access the cache and thereby gain access to the web page source code.
- However, you can place obstacles in the way of a potential peeker. First, you can disable use of the right mouse button on your site so the visitor can't access the View Source menu option on the context menu. This hide both your HTML code and your JavaScript from the visitor. Nevertheless, the visitor can still use the View menu's Source option to display your source code. In addition, you can store your JavaScript on your web server instead of building it into your web page. The browser calls the JavaScript from the web server when it is needed by your web page.
- Using this method, the JavaScript isn't visible to the visitor, even if the visitor views the source code for the web page.

6.5.1.1 Disabling the Right Mouse Button

The following example shows you how to disable the visitor's right mouse button while the browser displays your web page. All the action occurs in the JavaScript that is defined in the <head> tag of the web page.

```
<html>
<head>
<script>
window.onload = function()
{
document.addEventListener("contextmenu", function(e)
{
e.preventDefault();
```

| | | |
|--|--|--|
| | | |
|--|--|--|



```
}, false);}  
</script>  
<body>  
<h3>Right click on screen, Context Menu is disabled</h3>  
</body>  
</html>
```

The `preventDefault()` method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

Syntax

```
event.preventDefault()
```

```
<html>  
<body>  
<a id="myAnchor" href="https://w3schools.com/">Go to W3Schools.com</a>  
<script>  
document.getElementById("myAnchor").addEventListener("click", function(event){  
    event.preventDefault()  
});  
</script>  
</body>  
</html>
```



6.5.1.2 Hiding JavaScript

You can hide your JavaScript from a visitor by storing it in an external file on your web server. The external file should have the .js file extension. The browser then calls the external file whenever the browser encounters a JavaScript element in the web page. If you look at the source code for the web page, you'll see reference to the external .js file, but you won't see the source code for the JavaScript.

The next example shows how to create and use an external JavaScript file. First you must tell the browser that the content of the JavaScript is located in an external file on the web server rather than built into the web page. You do this by assigning the file name that contains the JavaScript to the `src` attribute of the `<script>` tag.

Next, you need to define empty functions for each function that you define in the external JavaScript file.

webpage.html

```
<html>
<head>
<script src="mycode.js" languages="javascript" type="text/javascript">
</script>
<body>
<h3> Right Click on screen, Context Menu is disabled</h3>
</body>
</html>
```

mycode.js

```
window.onload=function()
{
document.addEventListener("contextmenu", function(e)
{
e.preventDefault();
}, false);
}
```



6.5.2 Concealing Your E-mail Address

- Many of us have endured spam at some point and have probably blamed every merchant we ever patronized for selling our e-mail address to spammers.
- While e-mail addresses are commodities, it's likely that we ourselves are the culprits who invited spammers to steal our e-mail addresses.
- Here's what happens: Some spammers create programs called bots that surf the Net looking for e-mail addresses that are embedded into web pages, such as those placed there by developers to enable visitors to contact them. The bots then strip these e-mail addresses from the web page and store them for use in a spam attack.
- This technique places developers between a rock and a hard place. If they place their e-mail addresses on the web page, they might get slammed by spammers.
- If they don't display their e-mail addresses, visitors will not be able to get in touch with the developers.
- The solution to this common problem is to conceal your e-mail address in the source code of your web page so that bots can't find it but so that it still appears on the web page.
- Typically, bots identify e-mail addresses in two ways: by the mailto: attribute that tells the browser the e-mail address to use when the visitor wants to respond to the web page, and by the @ sign that is required of all e-mail addresses. Your job is to confuse the bots by using a JavaScript to generate the e-mail address dynamically. However, you'll still need to conceal the e-mail address in your JavaScript, unless the JavaScript is contained in an external JavaScript file, because a bot can easily recognize the mailto: attribute and the @ sign in a JavaScript.
- Bots can also easily recognize when an external file is referenced.
- To conceal an e-mail address, you need to create strings that contain part of the e-mail address and then build a JavaScript that assembles those strings into the e-mail address, which is then written to the web page.
- The following example illustrates one of many ways to conceal an e-mail address.
- It also shows you how to write the subject line of the e-mail. We begin by creating four strings:
 - The first string contains the addressee and the domain along with symbols &, *, and _ (underscore) to confuse the bot.
 - The second and third strings contain portions of the mailto: attribute name. Remember that the bot is likely looking for mailto:
 - The fourth string contains the subject line. As you'll recall from your HTML training, you can generate the TO, CC, BCC, subject, and body of an e-mail from within a web page.
- You then use these four strings to build the e-mail address. This process starts by using the replace() method of the string object to replace the & with the @ sign and the * with a period (.). The underscores are replaced with nothing, which is the same as simply removing the underscores from the string.
- All the strings are then concatenated and assigned to the variable b, which is then assigned the location attribute of the window object. This calls the e-mail program on the visitor's computer and populates the TO and Subject lines with the strings generated by the JavaScript.

| | | |
|--|--|--|
| | | |
|--|--|--|



```
<html>
<head>
<title>Conceal Email Address</title>
<script>

function CreateEmailAddress()
{
var x = 'abcxyz*c_o_m'
var y = 'mai'
var z = 'lto'
var s = '?subject=Customer Inquiry'
x = x.replace('&','@')
x = x.replace('*"..')
x = x.replace('_"')
x = x.replace('_"')
var b = y + z +':'+ x + s
window.location=b;
}

</script>
</head>
<body>
<input type="button" value="send" onclick="CreateEmailAddress()">
</body>
</html>
```



6.6 Frameworks of JavaScript and its application

JavaScript is a multi-paradigm language that supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. JavaScript was initially used only for the client-side. However, these days, JavaScript is used as a server-side programming language as well. To summarize, in just a simple sentence – JavaScript is the language of the web.

JavaScript framework is an application framework written in JavaScript where the programmers can manipulate the functions and use them for their convenience.

Frameworks are more adaptable for the designing of websites, and hence, most of the website developers prefer it. JavaScript frameworks are a type of tool that makes working with JavaScript easier and smoother. These frameworks also make it possible for the programmer to code the application as a device responsive.

Following are the most used framework of JavaScript:

1) React



React is not among the earliest disruptive JavaScript-based Web frameworks. But it is the most disruptive and influential JavaScript-based Web framework. Jordan Walke and a group of Facebook Engineers created React in 2013 as a Component-based Web Framework with one-way data flow and changed the Front-end Web Development forever. It also introduced many other concepts like functional, declarative programming, immutable state, which was uncommon in Front-end development. The other breakthrough of React was to introduce the Virtual DOM, which gives better user experience and performance gain.

Features

- **Declarative:** Creates interactive and dynamic UI for websites and mobile applications. React updates efficiently and render the right components when data changes. Declarative views make the code readable and easy to debug.
- **Virtual DOM:** For every DOM object, there is a corresponding "virtual DOM object." It creates a virtual copy of the original DOM and is a representation of a DOM object,
- **Event handling:** React has its fully compatible W3C object model event system created. It also provides a cross-browser interface to a native event, meaning no need to worry about incompatible event names and fields. React reduces memory head by as event system is implemented through event delegation and has a pool of event objects.

| | | |
|--|--|--|
| | | |
|--|--|--|

- **JSX:** JSX is a markup syntax that closely resembles HTML. JSX makes writing React components easier by making the syntax almost identical to the HTML injected into the web page.
- **Performance:** React uses one-way data binding with an application architecture called Flux controls. ReactJS helps update the View for the user and, Flux controls the application workflow. Virtual DOM adds advantages as it compares the new data with original DOM and updates the View automatically.
- **React Native:** React Native is a custom renderer for React; it uses native components instead of web components like React as building blocks. It also serves access to these platforms' features, apart from transforming React code to work on iOS and Android.
- **Component-Based:** In React, everything is a component of the web page divided into small components to create a view(or UIs). Every part of the application visuals would be wrapped inside a self-contained module known as a component. Components in ReactJS use to define the visuals and interactions in applications.

2. Node.js



- In 2009, Ryan Dahl created the asynchronous, event-driven Server-Side JavaScript runtime Node.js and brought JavaScript in the uncharted territory of Back-end development.
- Ryan Dahl has used the popular JavaScript Engine V8 and C++ libraries. Since then, the popularity of both Node.js and JavaScript has skyrocketed.
- With Node Package Manager NPM and countless numbers of frameworks/libraries, Node.js has surpassed many other established Server-side frameworks.
- Because of its Asynchronous Event-Driven nature and lightweight, fast runtime, Node.js is especially suited for I/O heavy applications like Web, IoT, Serverless.
- Node.js is one of the primary driving force to improve JavaScript as a programming language and to increase the popularity of JavaScript.

Features:

- **It is swift:**
The library of Node.js is fast when it comes to code execution, as it is built on the V8 JavaScript engine of Google Chrome.
- **I/O is asynchronous and Event-Driven:**
All the APIs are asynchronous, which means that its server does not wait for the API to come back with data. Here the server calls the APIs one by one and keeps

| | | |
|--|--|--|
| | | |
|--|--|--|

moving to the next one while using a notification mechanism of Events to generate a response from the API, called previously. This makes it fast too.

- **Single-threaded:**
Node.js, along with event looping, follows a single-threaded model.
- **Highly scalable:**
Node.js follows an event mechanism that makes it possible for the server to respond in a non-blocking manner, which eventually makes it scalable.
- **No buffering:**
When it comes to uploading audio and video files, Node.js cuts down the processing time significantly. It does not buffer any data, and here the application gets out the data in chunks.
- **Open source:**
Being open-source, Node.js's community has come up with several amazing models that can be used to add better capabilities to the Node.js applications.
- **License:**
It has been released under MIT license.

3. Vue.js



In modern days where Web frameworks are backed by Giant Tech companies, Vue.js is an exception. In 2014, an ex-Google Engineer Evan You decided to combine the good parts of AngularJS (View Layer) and the good parts of React (Virtual DOM) and created Vue.js. Today, Vue.js is one of the most popular JavaScript-based Web frameworks. One of the key design goals of Evan You was to lower the barrier into JavaScript-based front-end development. Vue.js is one of the easiest Front-end frameworks where developers can write SPA applications with minor effort.

Developers can use Vue.js as an End-to-End framework with Routing, State management like Angular, or as only a view layer like React. It also offers Angular like two-way data-binding with additional Reactivity and React like rendering using Virtual DOM.

Features:

- **Virtual DOM:** Vue.js utilizes virtual DOM. Virtual DOM is a clone of the principal DOM element. The virtual DOM absorbs every change intended for the DOM presents in the form of JavaScript data structures, which are compared with the original data structure.
- The viewers view final changes that reflect in the real DOM. The method is creative and cost-effective; also, the changes are done quickly.

| | | |
|--|--|--|
| | | |
|--|--|--|

- | | | |
|--|--|--|
| | | |
|--|--|--|
- **Data Binding:** This feature facilitates to manipulate or assign values to HTML attributes., change the style, assign classes with v-bind available, which is a binding directive.
 - **CSS Transitions and Animations:** This feature provides several methods to apply a transition to HTML elements when added, updated, or removed from the DOM. Its features consist of a built-in component that wraps the element responsible for returning the transition effect.
 - **Template:** It provides HTML-based templates that bind the DOM with the Vue.js instance data. The templates are compiled into Virtual DOM Render functions. A developer can use the render functions template and can replace the template with the render function.
 - **Methods:** We use methods when an event occurs that isn't necessarily related to the instance data being mutated or want to change a component's state. Methods do not keep records of any dependencies but can take arguments.
 - **Complexity:** Vue.js is simpler in terms of API and design. A web developer builds simple applications in a single day.

4. Angular



In AngularJS, Google had created one of the earliest hot JavaScript-based Front-end frameworks in 2010. But once Facebook released React, it exposed the design flaws of AngularJS, and it quickly became an outdated framework. As a result, the Google team has created an entirely new SPA framework and released it as Angular in 2016. Although Angular and AngularJS have similar names, in reality, they are two different frameworks. Unlike React, it is an end-to-end Framework with "Out-of-the-box" support of everything one needs to develop an Enterprise-grade Web App. Also, Angular is the first significant framework that has embraced TypeScript and played a considerable role in making TypeScript popular.

Features:

- Angular.js is an end-to-end framework with "out of the box" support to develop Enterprise Application. In Angular CLI, it has one of the best Command-Line Tool in the JavaScript landscape.
 - With TypeScript and separating the template from styling and business logic, it is especially suited for the enterprise-grade large code-base.
 - It is inherently the most secure Front-end framework with built-in features like DOM sanitization.
- | | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

- Although Google is not backing Angular the same way as Facebook is backing React, it is still putting enough resources so that Angular remains an attractive and innovative framework. Recently it has added Lazy Loading, Differential loading to improve loading time of modules.
- In Angular 9, it releases a new rendering Engine Ivy to improve startup time, response time, and to reduce bundle size.

5. Express



When Node.js appeared in 2009, TJ Holowaychuk has created Express.js based on the minimalistic Web Framework Sinatra. It is a minimalistic Web framework to develop Web application and REST API. It is also less opinionated and very fast. Many other JavaScript-based Web frameworks are based on Express. Today, Express.js is the most popular JavaScript-based Web application framework hands down.

Features:

- Express.js is almost the default JavaScript Server Side framework.
- Express is the complete Application framework with middleware, routing, template.
- Express supports MVC pattern with View system supporting 14+ templating engines.
- It also offers robust routing.
- Express also supports content negotiation.

6. Next.js



React is a very unopinionated framework where React-Core just offers the view layer. There was always a need for an end-to-end, opinionated framework based on React. Tim Neutkens and a group of Software Engineers from the Dutch company Zeit has created Next.js as an end-to-end, higher-level Web Framework on top of React and Node.js. Next.js offers both Server-Rendered and Static Web sites for Web, Desktop, and Mobile devices.

Features:

- Next.js is built upon the two most popular and battle-hardened JavaScript frameworks: React and Node.js.

| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

- It also offers “Build once, runs everywhere,” i.e., a Next.js can run on Web, Mobile, and Desktop.
- Next.js offers excellent Server-Side rendering with exceptional SEO support and fast startup.
- It offers automatic code splitting and filesystem-based routing.
- It also supports easy-to-use data fetching and built-in CSS support.

7. Meteor



In 2012, a group of Engineers had created Meteor as an isomorphic, open-source full-stack JavaScript framework based on Node.js. It also supports building end-to-end applications for Web, Mobile, Desktop platform and integrates well with popular front-end frameworks like React, Vue.js, Angular, Svelte. It is also a “Batteries Included” framework with “Out-of-the-box” support for Enterprise-grade App development.

Features:

- Meteor is a full-stack framework to develop the complete stack: Frontend-to-Backend.
- For front-end development, it has its own template engine. But developers can use Meteor with other popular front-end frameworks like Angular, React, Vue.js or Svelte.
- It is a cross-platform framework and can develop an application for Web, Mobile, and Desktop.
- Meteor has integrated JavaScript stack, which enables different integrating technologies (e.g., MongoDB database, React front-end) with minimum effort.
- It is an Isomorphic platform sharing the same API on client-side and server-side.

8. Svelte



In 2016, a Guardian Software Engineer Rich Harris had the groundbreaking idea to develop a JavaScript framework with no framework-specific Runtime and released Svelte. The idea was to use the Svelte compiler, which would compile framework-specific code to plain JavaScript, HTML, CSS, and render the compiled code to the browser. Although the

| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

concept was not new in software development, it was uncharted territory in Front-end development. The other significant contribution of Svelte is to add first-class support of reactivity, which leads to faster, improved performance without Virtual DOM. Today, it is arguably the hottest Front-end framework with tremendous traction and interest in the industry.

Features:

- It is a compile-time framework and does not need any framework-specific runtime. It has the smallest bundle size among all frameworks.
- Svelte performs the DOM rendering via reactive programming, which is faster than Virtual DOM most times. As a result, Svelte gives the fastest rendering among all frameworks.
- Svelte is just a View layer like React-Core, and it is an unopinionated framework.
- Svelte supports both client-side and server-side rendering with excellent SEO support.
- Developers can use Svelte to develop a Web app, Cross-platform Mobile App development, or Desktop app development.

9. Koa

ko

oa

In 2013, the core members of Express.js led by TJ Holowaychuk had created Koa as a lightweight, modern, expressive, and robust middleware framework for Web Applications and APIs. Koa is hugely modular with tiny Core with no middleware. However, middleware is available as separate modules.

Features:

- Koa has a lightweight, smaller Core with no out-of-the-box Middleware bundle.
- Koa has a highly modular architecture and offers pluggable middleware Modules.
- Koa supports cascading middleware in a stack-like manner, which allows to perform actions downstream then and manipulate the response upstream.
- Koa uses `async/await` instead of callback and supports cleaner, expressive code with better error handling.
- In terms of performance, it outperforms Express.js.

10. Ember.js



| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

Inspired by the Ruby on Rails principle “Convention over Configuration,” Yehuda Katz from Apple has created Ember.js as a highly opinionated, end-to-end framework in 2012. Ember.js is a strictly backward compatible framework introducing no significant breaking changes since its inception. Where other frameworks from that era (Backbone.js, AngularJS) are diminishing in popularity, Ember.js is still giving a reliable, productive framework to fulfill the need of modern Front-end development.

Features:

- End-to-end opinionated cohesive framework focusing on “Convention over Configuration.”
- Instead of one Tech giant, Ember is backed by several Tech Giant like LinkedIn, Yahoo. As a result, it is not driven by one corporation’s needs.
- Ember’s Data library is the best to access data across multiple sources at once, set up asynchronous relationships.
- In Ember CLI, it has the best CLI among all JavaScript frameworks, which helps to scaffold and generating all the necessary codes with the right structure, including all dependencies.
- In its latest release Ember Octane, it has introduced HTML first and component first approach with improved support for state management and reactivity.

11. Backbone.js



It is one of the most popular JavaScript frameworks. It is effortless to understand and learn. It can be used to create Single Page Applications. The development of this framework involves the idea that all the server-side functions must flow through an API, which would help achieve complex functionalities by writing less code.

Features:

- BackboneJS uses JavaScript functions, making the development of applications and the frontend in a much easier.
- Building blocks such as models, views, events, routers, and collections are provided for assembling the client-side web applications.
- It is a simple library that helps in separating business and user interface logic.
- It is a free and open-source library and contains over 100 available extensions.
- It is a backbone for any project and helps in the organization of the code.
- BackboneJS has a soft dependency on jQuery and a hard dependency on Underscore.js.
- It allows us to create client-side web applications or mobile applications in a well-structured and organized format.

| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

12.Aurelia



Aurelia framework is the latest version of JavaScript, which can be used to implement any interface. It is the next generation of the framework for developing far more robust websites. The framework of Aurelia can extend the HTML for various purposes, including data binding. Also, its modern architecture ensures that the purpose of toll is for interpretation client-side and server-side at a time.

Features:

- Components: Components are building blocks of the Aurelia framework and are composed of JavaScript view-model pairs and HTML views.
- Web Standards: It is one of the cleanest modern frameworks. It completely focuses on web standards without unnecessary abstractions.
- Extensible: The framework facilitates an easy way to integrate with the other needed tools.
- Commercial Support: This framework offers commercial and enterprise support.
- License: Aurelia is open-sourced and licensed under MIT license.

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<script>
var subjectObject = {
  "Front-end": {
    "HTML": ["Links", "Images", "Tables", "Lists"],
    "CSS": ["Borders", "Margins", "Backgrounds", "Float"],
    "JavaScript": ["Variables", "Operators", "Functions", "Conditions"]
  },
  "Back-end": {
    "PHP": ["Variables", "Strings", "Arrays"],
    "SQL": ["SELECT", "UPDATE", "DELETE"]
  }
}
window.onload = function() {
  var subjectSel = document.getElementById("subject");

```

| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

```
var topicSel = document.getElementById("topic");
var chapterSel = document.getElementById("chapter");
for (var x in subjectObject) {
    subjectSel.options[subjectSel.options.length] = new Option(x, x);
}
subjectSel.onchange = function() {
    //empty Chapters- and Topics- dropdowns
    chapterSel.length = 1;
    topicSel.length = 1;
    //display correct values
    for (var y in subjectObject[this.value]) {
        topicSel.options[topicSel.options.length] = new Option(y, y);
    }
}
topicSel.onchange = function() {
    //empty Chapters dropdown
    chapterSel.length = 1;
    //display correct values
    var z = subjectObject[subjectSel.value][this.value];
    for (var i = 0; i < z.length; i++) {
        chapterSel.options[chapterSel.options.length] = new Option(z[i], z[i]);
    }
}
</script>
</head>
<body>
```

<h1>Cascading Dropdown Example</h1>

```
<form name="form1" id="form1" action="/action_page.php">
Subjects: <select name="subject" id="subject">
    <option value="" selected="selected">Select subject</option>
</select>
<br><br>
Topics: <select name="topic" id="topic">
    <option value="" selected="selected">Please select subject first</option>
</select>
<br><br>
Chapters: <select name="chapter" id="chapter">
    <option value="" selected="selected">Please select topic first</option>
```

| | | |
|--|--|--|
| | | |
|--|--|--|

| | | |
|--|--|--|
| | | |
|--|--|--|

```
</select>
<br><br>
<input type="submit" value="Submit">
</form>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML and CSS Slideshow</title>
    <style>
        body {
            font-family: Helvetica, sans-serif;
            padding: 5%;
            text-align: center;
            font-size: 50px;
        }

        /* Styling the area of the slides */

        #slideshow {
            overflow: hidden;
            height: 510px;
            width: 728px;
            margin: 0 auto;
        }

        /* Style each of the sides
        with a fixed width and height */

        .slide {
            float: left;
            height: 510px;
            width: 728px;
        }
    </style>
</head>
<body>
    <div id="slideshow">
        <div class="slide">
            <img alt="A small thumbnail image of a slide." data-bbox="111 447 529 500"/>
        </div>
        <div class="slide">
            <img alt="A small thumbnail image of a slide." data-bbox="111 500 529 553"/>
        </div>
        <div class="slide">
            <img alt="A small thumbnail image of a slide." data-bbox="111 553 529 606"/>
        </div>
        <div class="slide">
            <img alt="A small thumbnail image of a slide." data-bbox="111 606 529 659"/>
        </div>
    </div>
</body>
</html>
```

| | | |
|--|--|--|
| | | |
|--|--|--|

```
/* Add animation to the slides */

.slide-wrapper {

    /* Calculate the total width on the
    basis of number of slides */
    width: calc(728px * 4);

    /* Specify the animation with the
    duration and speed */
    animation: slide 10s ease infinite;
}

/* Set the background color
of each of the slides */

.slide:nth-child(1) {
    background: green;
}

.slide:nth-child(2) {
    background: pink;
}

.slide:nth-child(3) {
    background: red;
}

.slide:nth-child(4) {
    background: yellow;
}

/* Define the animation
for the slideshow */

@keyframes slide {

    /* Calculate the margin-left for
    each of the slides */
    20% {
        margin-left: 0px;
    }
    40% {
        margin-left: calc(-728px * 1);
    }
    60% {
        margin-left: calc(-728px * 2);
    }
    80% {
        margin-left: calc(-728px * 3);
    }
}
```

| | | |
|--|--|--|
| | | |
|--|--|--|

```
    </style>
</head>

<body>

    <!-- Define the slideshow container -->
    <div id="slideshow">
        <div class="slide-wrapper">

            <!-- Define each of the slides
            and write the content -->
            <div class="slide">
                <h1 class="slide-number">
                    GeeksforGeeks
                </h1>
            </div>
            <div class="slide">
                <h1 class="slide-number">
                    A computer science portal
                </h1>
            </div>
            <div class="slide">
                <h1 class="slide-number">
                    This is an example of
                </h1>
            </div>
            <div class="slide">
                <h1 class="slide-number">
                    Slideshow with HTML and CSS only
                </h1>
            </div>
        </div>
    </div>
</body>
</html>
```

| | | |
|--|--|--|
| | | |
|--|--|--|