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

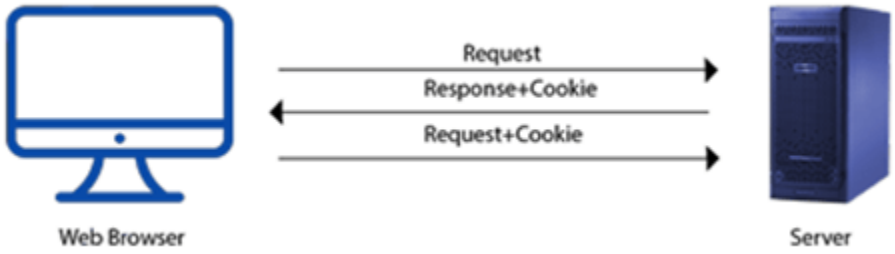
All cookies are not created equal. There are 3 types of them:

- 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:

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,

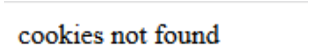
```
document.cookie="username=VP; expires=Tues,04 Aug 2020 00:00:00 GMT ";
```

Code:

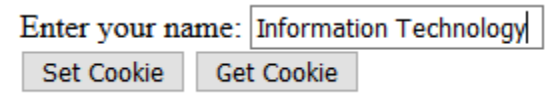
```
<html>
<head>
<script>
function writeCookie()
{
var d=new Date();
d.setTime(d.getTime()+(1000*60*60*24));
with(document.myform)
{
document.cookie="Name=" + person.value + ";expires=" +d.toGMTString();
}
}
function readCookie()
{
if(document.cookie=="")
document.write("cookies not found");
else
document.write(document.cookie);
}
</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:

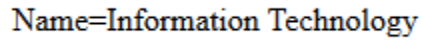
Click on “Get Cookie”



Enter your name in text box and click on “Set Cookie”



After clicking on ““Get Cookie”



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:

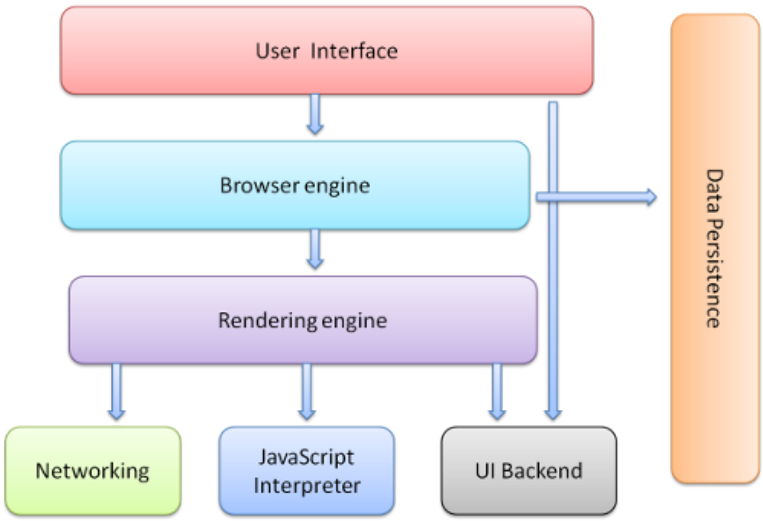
Used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.

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

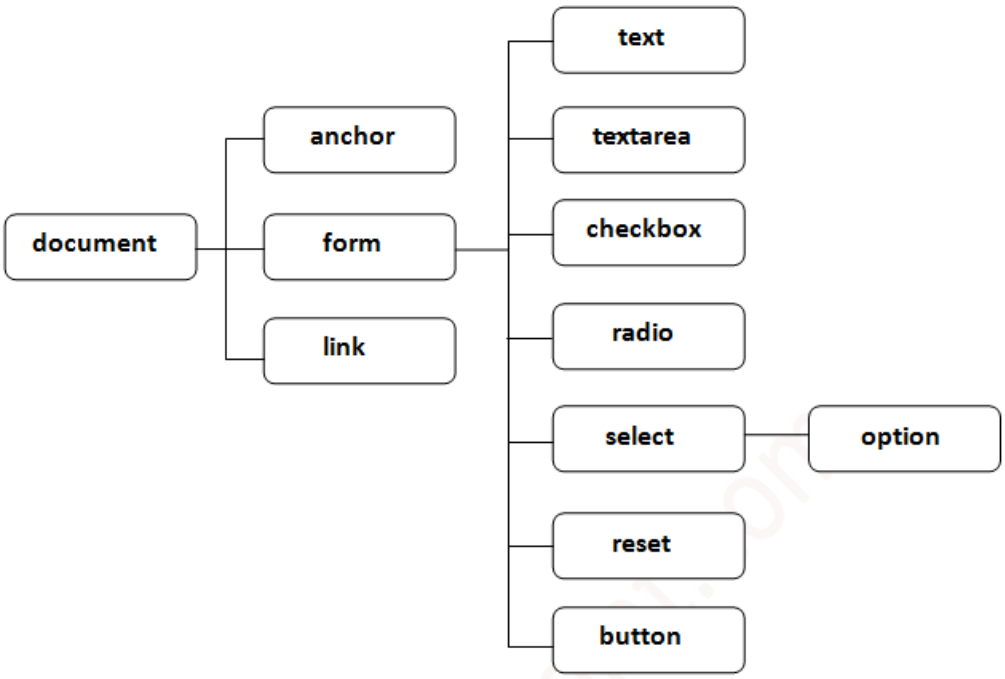
```
window.document
```

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.

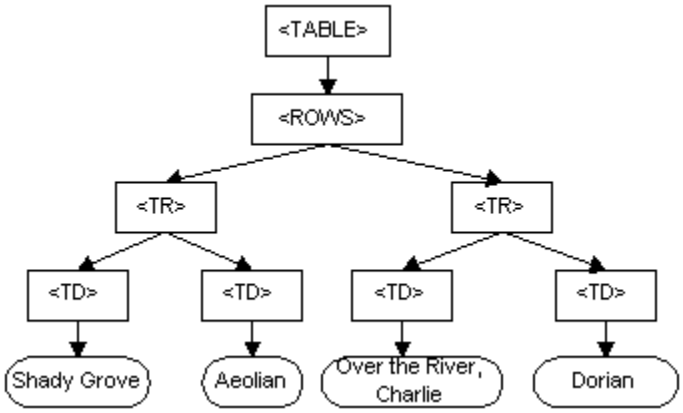


Source: https://www.javatpoint.com/document-object-model

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.

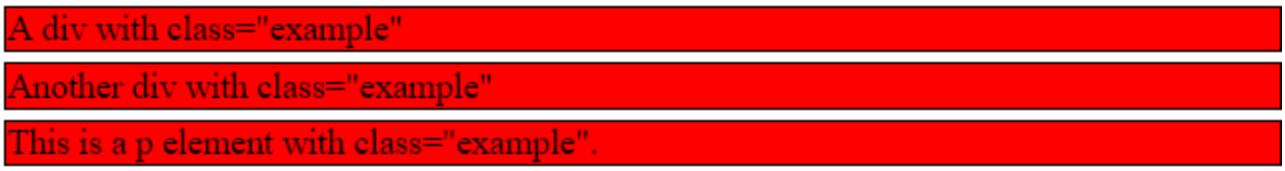
3

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:



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.

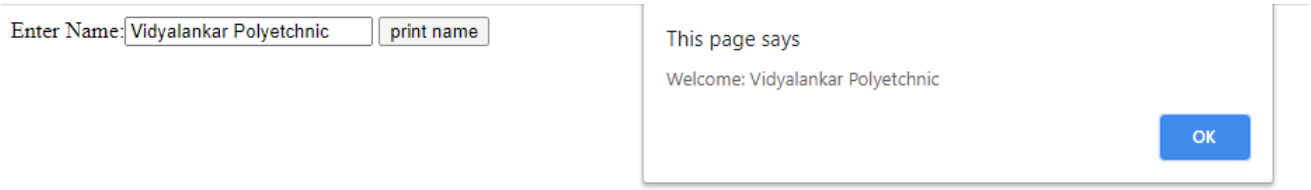
name is the attribute name of the input text.

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:



Javascript – innerText

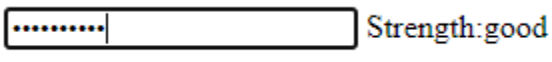
The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

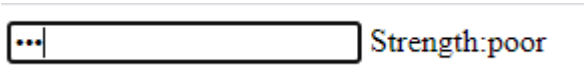
It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

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:





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
URL	Optional. The URL of the page to open. If no URL is specified, a new blank window/tab is opened
name	Optional. The target attribute or the name of the window. The following values are supported:

Value	Description
_blank	URL is loaded into a new window, or tab. This is the default
_parent	URL is loaded into the parent frame
_self	URL replaces the current page
_top	URL replaces any framesets that may be loaded

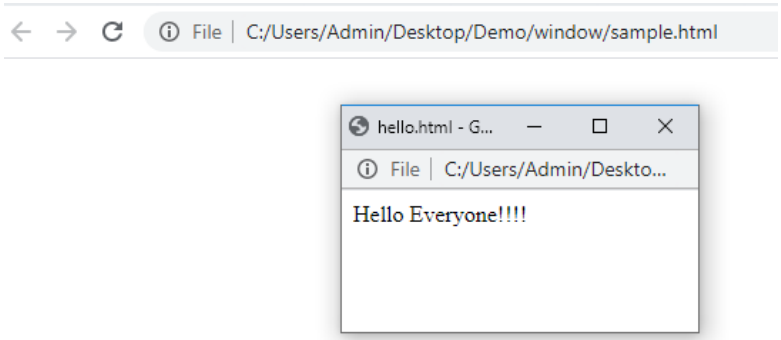
To close the window, JavaScript provides **close ()** method.
Syntax: window. Close ();

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:



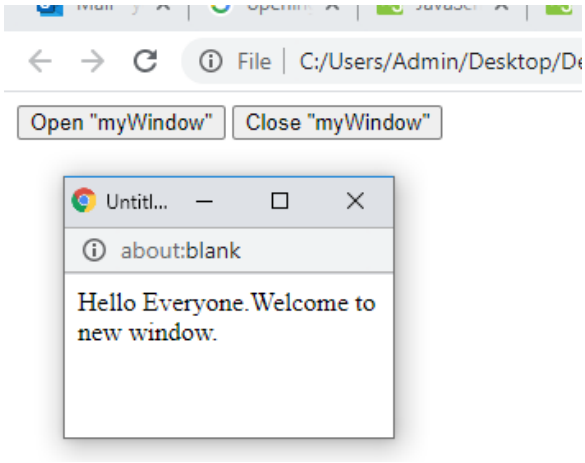
Code: Open a new window and close that window on button click event using open( ) and close ( ).

```
<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:

Proprties	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 avilable Top is " + screen.availTop;
document.getElementById("demo3").innerHTML =
"Available screen avilable 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 avilable Top is 0

Available screen avilable 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.

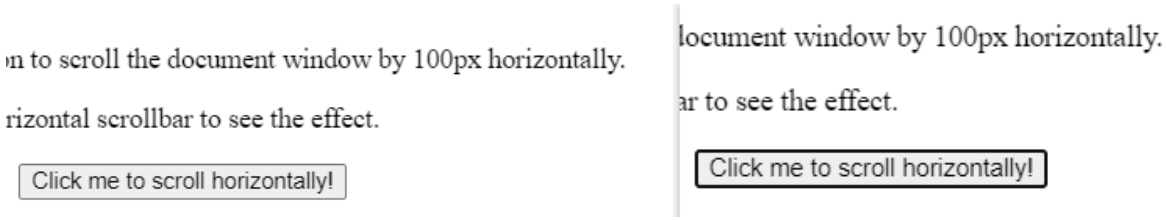
Syntax: window.scrollBy(xnum, ynum)

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:



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).

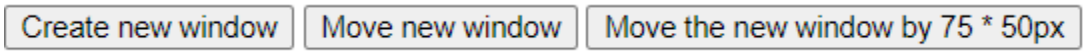
Syntax: *HTMLElementObject*.focus()

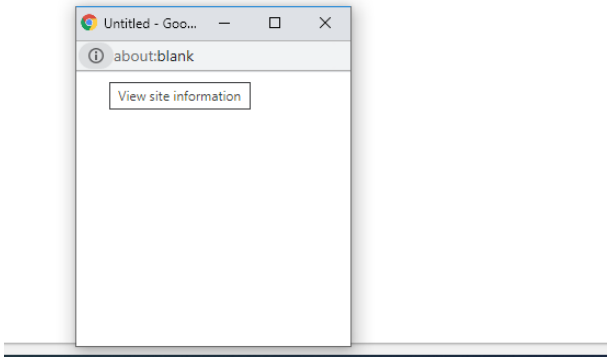
Code: simple example of moveTo ( )

```
<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:





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.

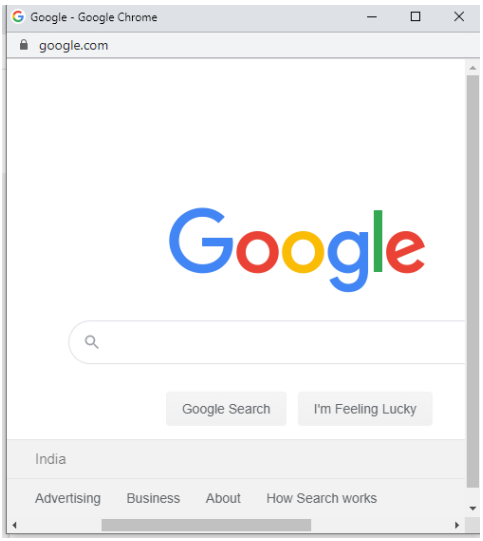
In following example, we are creating only one object of window and each time same window remain open and content of window changes.

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

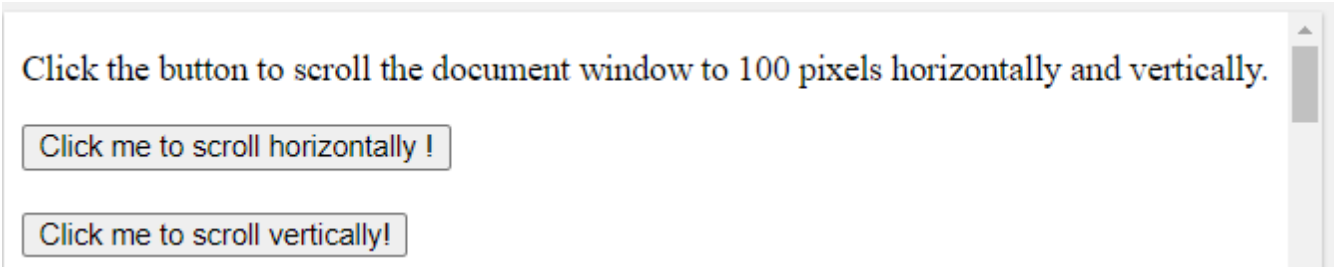
Ypos: Required. The coordinate to scroll to, along the y-axis (vertical), in pixels

Code: to scroll the document window to 100 pixels horizontally and vertically.

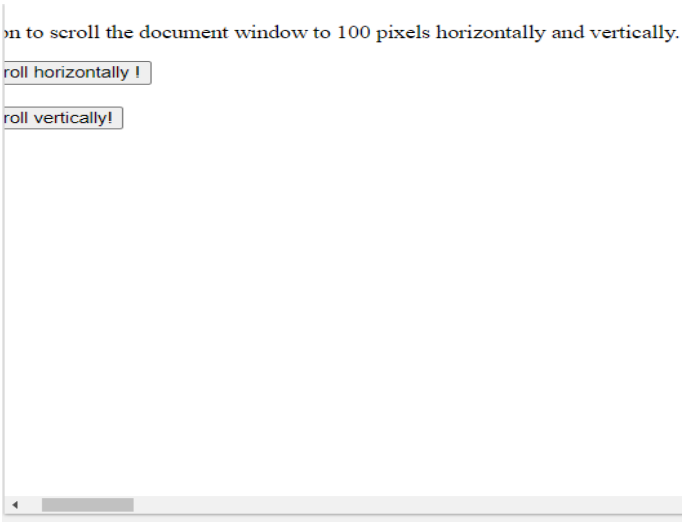
```
<html>
<head>
<style>
body
{
  width: 5000px;
  height: 5000px;
}
</style>
</head>
<body>

<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>
```

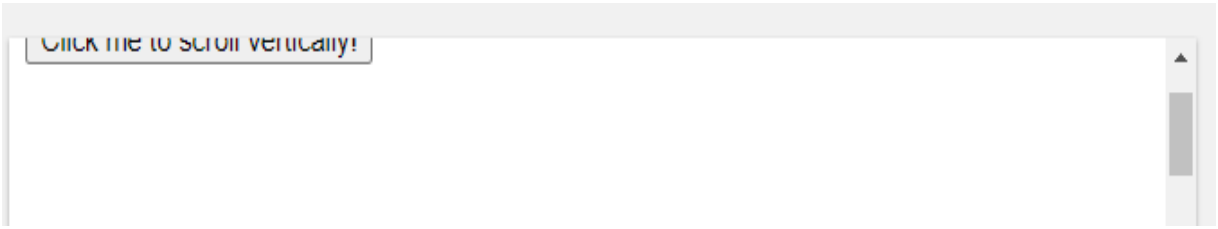
Output:



When user clicks on “Click me to scroll horizontally!” the output will be like this:



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.

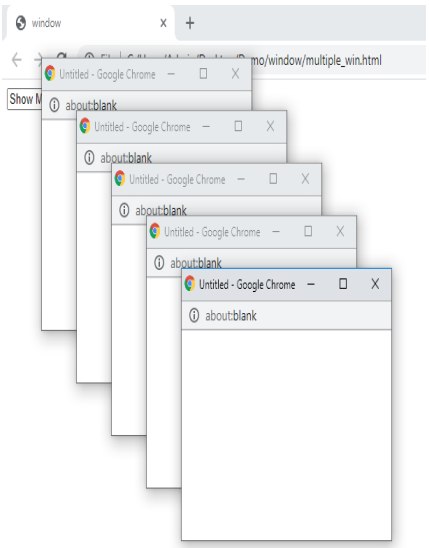
Code: In following example, x variable is assigned to top dimension of window and y variable is assigned to left dimension of window.

```
<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.



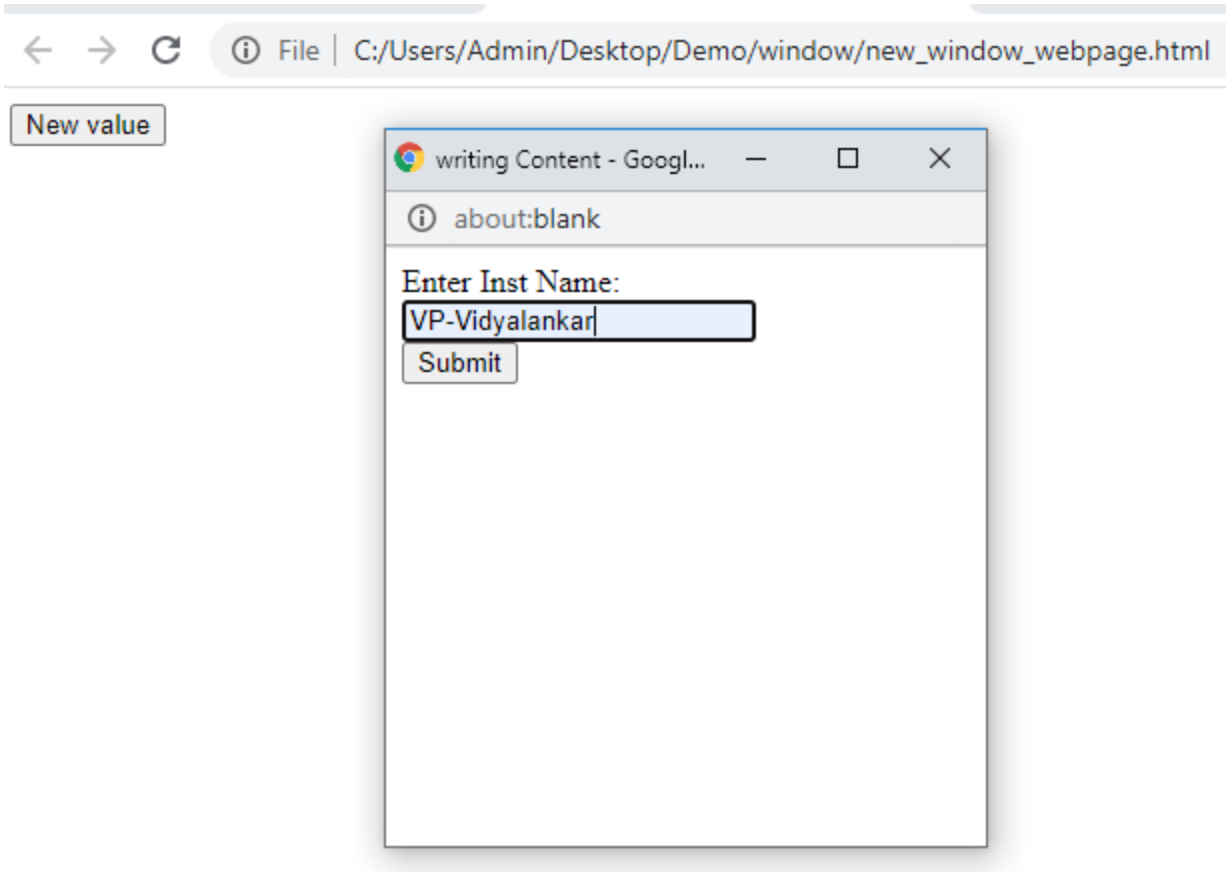
Creating a web page in new window

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:



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.

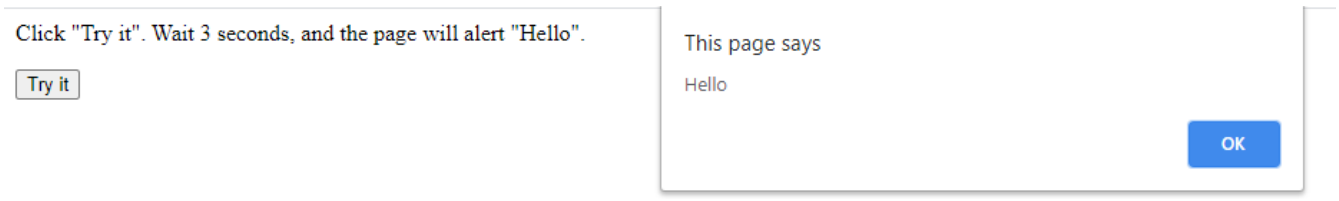
The first parameter is a function to be executed.

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:



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:

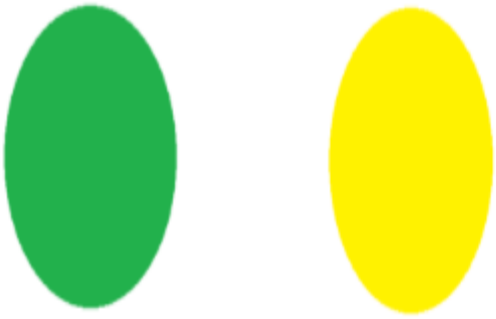
11:31:35 PM

Stop time

Code: Rotating images using setTimeout(0 method. Following example is using an array of images. Using setTimeout(0 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>
<img id="car" src="car.jpg">
<br><br>
<input value="Start" type="button" onclick="start()"/>
<input value="Stop" type="button" onclick="stop()"/>
</body>
</html>
```

Output:

Start Stop

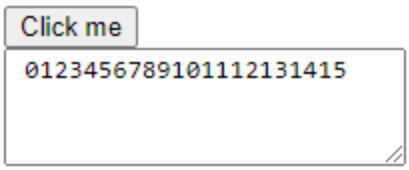
Start Stop

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:



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.

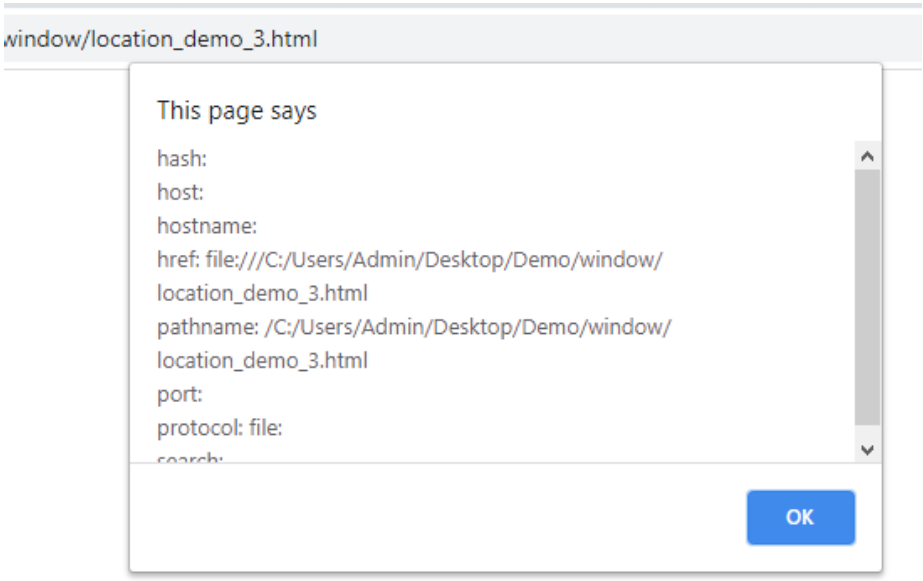
Location Object Methods

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:



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.

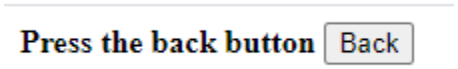
History Object Methods

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:




```
<!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>
```

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
```

```
    let dateTime= new Date();
    let time = dateTime.toLocaleTimeString();
   document.write("<br>"+time);

}, 2000);

</script>
```