Cookies & Browser Data

Unit - IV

Cookies

- The goal of a web site programmer should be to make the web site experience as easy and pleasant for the users as possible.
- Apart from well-designed web pages with easily navigable layouts, learning about users and using information gained about them is also very effective.
- E.g. Amazon's, it incorporates one click purchasing system. By already knowing the user's purchasing details, such as credit-card number and delivery address, you can allow the user to go from viewing a book to buying it in just one click.
- Also, based on information, such as the previous purchases and browsing patterns of the user, it's possible to make book suggestions.

Cookies

- Such personalization requires that information about users be stored somewhere in between their visits to the web site.
- Accessing the user’s local file system from a web application is pretty much off limits because of security restrictions included in browsers.
- However we can store small amounts of information in a special place on the user’s local disk, using what is called a *cookie.*
- *Cookie* property of *document* object can be used to create and retrieve cookie data from within a JavaScript code.

Cookies

- Cookies are small text files that a browser stores in the visitor’s computer.

- Cookies were invented to solve the problem "how to remember information about the user“

- A cookie is basically a string-value pairs separated by semi-colons.

  e.g. “color=red;expires=Fri, 5 Aug 2016 01:00:00 UTC;”
- *a*

Cookies

- Cookies contains 5 variable length fields:
  - Expires: The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
  - Domain: The domain name of your site.
  - Path: The path to the directory or web page that set the cookie.
  - Secure: If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
  - Name=value: Cookies are set and retrieved in the form of key-value pairs

Cookies

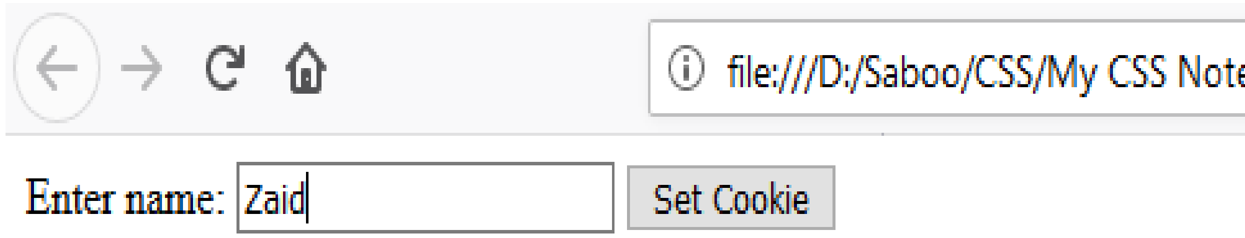
- Storing Cookies:
  - A cookie can be created by assigning string-value to the document.cookie object.
  - *document.cookie = "key1 = value1;key2 = value2;expires = date";*
  - Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript **escape()** function to encode the value before storing it in the cookie.

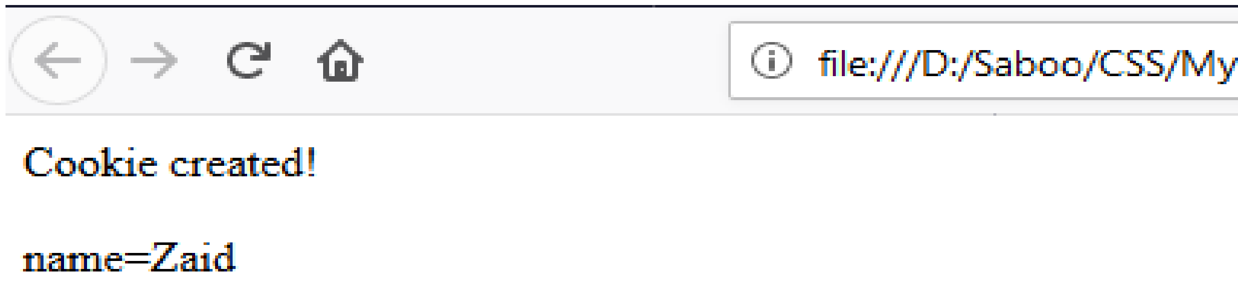
Cookies

- Reading Cookies:
  - A cookie value can be read by using document.cookie object.
  - Document.cookie string will keep a list of name=value pairs separated by semicolons,
  - where name is the name of the cookie and value is its string value.
  - You can use strings' **split()** function to break a string into key and values

Cookies

```
<html>
  <head>   <script>
        function WriteCookie() {
          if( document.myform.customer.value == "" ) {
            alert("Enter some value!");
            return;   }
          cookievalue = escape(document.myform.customer.value) + ";";
          document.cookie = "name=" + cookievalue;
          document.write ("Cookie created! <br>");
          var allcook=document.cookie;
          document.write("<br>"+allcook);  }
    </script>   </head>
  <body>   <form name = "myform">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie();"/>
    </form> </body>
</html>
```





Cookie created!

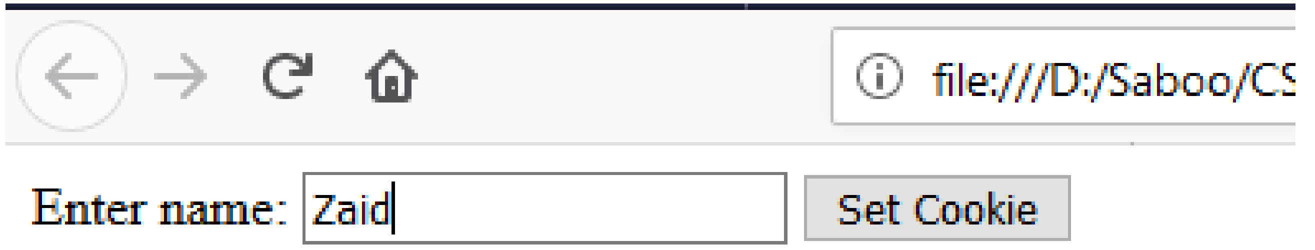
name=Zaid

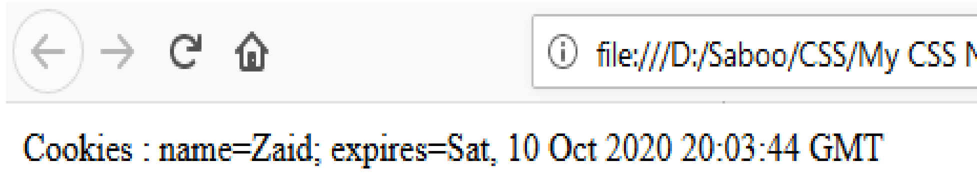
Cookies

- Setting Cookies expiry date:
    - Life of a cookie can be extended beyond the current browser session by setting an expiration date and saving the expiry date within the cookie.
    - It can be achieved by setting ‘expires’ attribute to a date and time.
    - expires=Thu, 18 Dec 2013 12:00:00 UTC

Cookies

```
<html>
  <head> <script>
        function WriteCookie() {
          var now = new Date();
          now.setMonth( now.getMonth() + 1 );
          cookievalue = escape(document.myform.customer.value) + ";"
          document.cookie = "name=" + cookievalue;
          document.cookie = "expires=" + now.toUTCString() + ";"
         var allcook = document.cookie;
          document.write ("Cookies : " + allcook);   }
    </script> </head>
  <body> <form name = "myform" action = "">
      Enter name: <input type = "text" name = "customer"/>
      <input type = "button" value = "Set Cookie" onclick = "WriteCookie()"/>
    </form> </body>
</html>
```





```
Cookies : name=Zaid; expires=Sat, 10 Oct 2020 20:03:44 GMT
```

Browser

- The Window object represents the browser's frame on a page or document.
- If you have a page with no frames, there will be just one window object.
- However, if you have more than one frame, there will be one window object for each frame.

Opening new Windows

- The *window* object has an *open()* method, which opens up a new window.

Syntax: *window.open(URL, name, specs, replace)*

where:

URL: specifies the URL of the page to open. If no URL is specified, a new window/tab with about:blank is opened.

name: specifies the target attribute or the name of the window. Following values are supported:

_blank: URL is loaded into a new window or tab.

_parent: URL is loaded into the parent window.

_self: URL replaces the current page.

_top: URL replaces any framesets that may be loaded.

name: The name of the window.

Opening new Windows

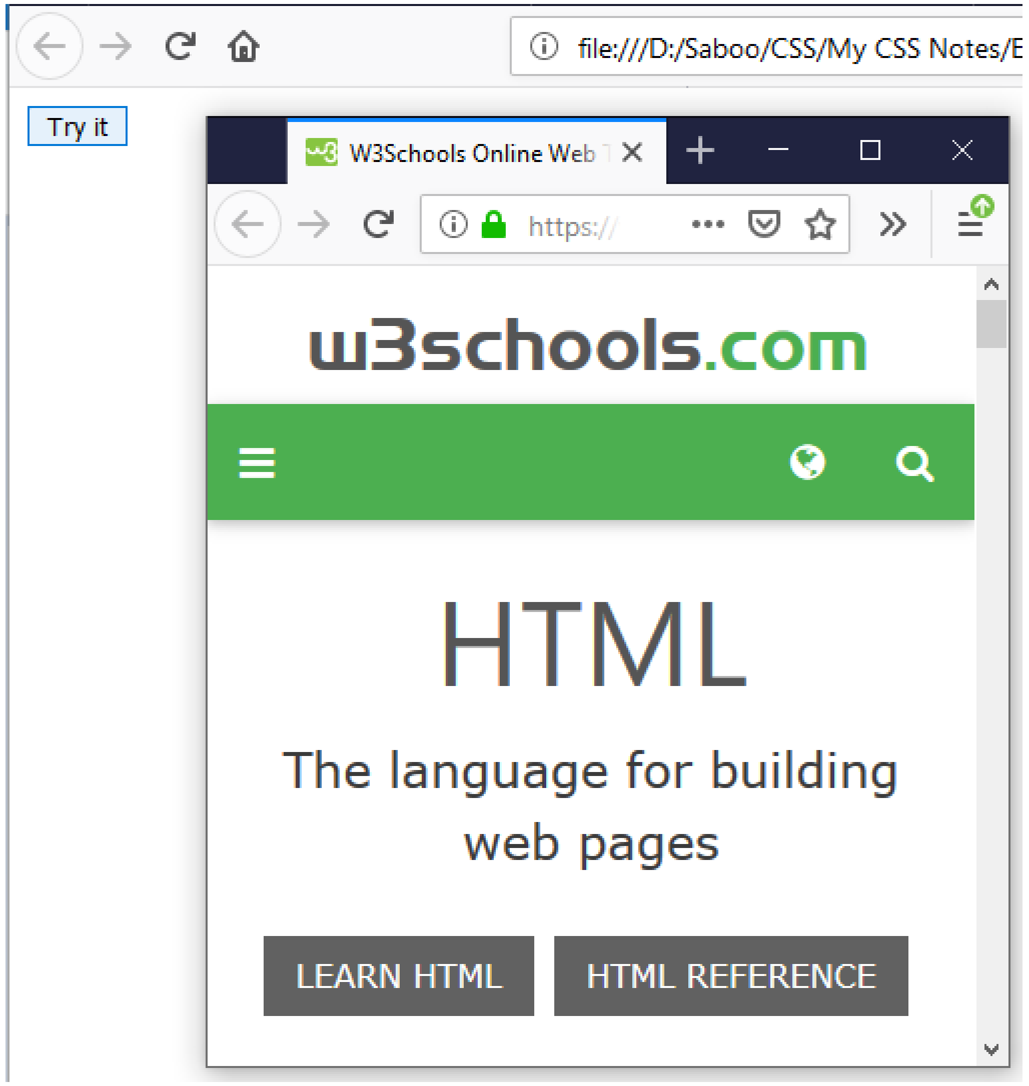
specs: A comma separated lists of items. E.g. fullscreen, height,width, left, top, menubar, scrollbars, titlebar, toolbar.

replace: Specifies whether the URL creates a new entry or replaces the current entry in the history list. (true/false)

- When you click same button multiple times to open a window, it will replace the previous child window with a new window with overwritten data.(only 1 window is visible)
- To open multiple windows from single button (keeping each child window), use different window name for each click.(provide a rand number for window name)

Opening new Windows

```
<html><body>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
 window.open("https://www.w3schools.com",
  "_blank“,"toolbar=yes,scrollbars=yes,
  resizable=yes,top=500,left=500,width=400,
  height=400");}
</script>
</body></html>
```



New window focus

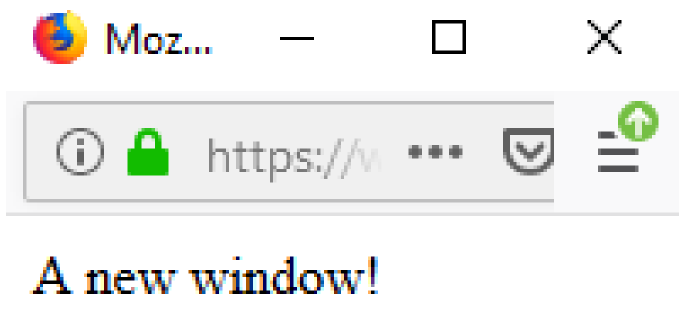
- *focus()* method sets focus to the current window.

- This method makes a request to bring the current window to the foreground.

  Syntax: *window.focus()*

New window focus

```
<html>
<body>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  var myWindow = window.open("", "", "width=200,height=100");
  myWindow.document.write("<p>A new window!</p>");
  myWindow.focus();
}
</script>
</body>
</html>
```



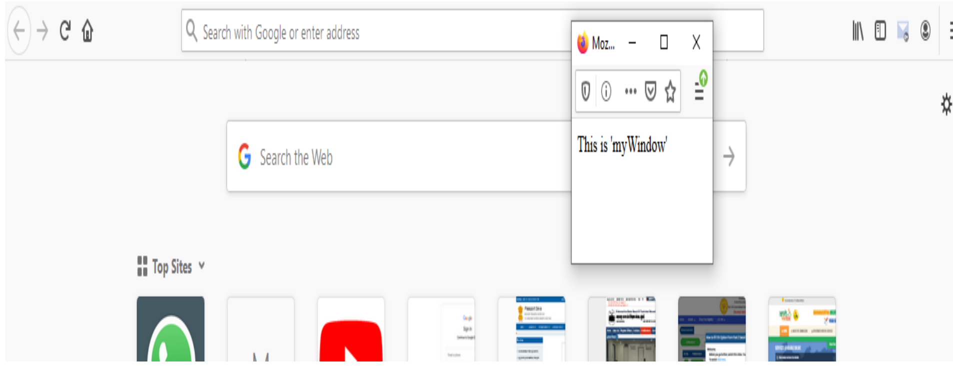
A new window!

Window Position

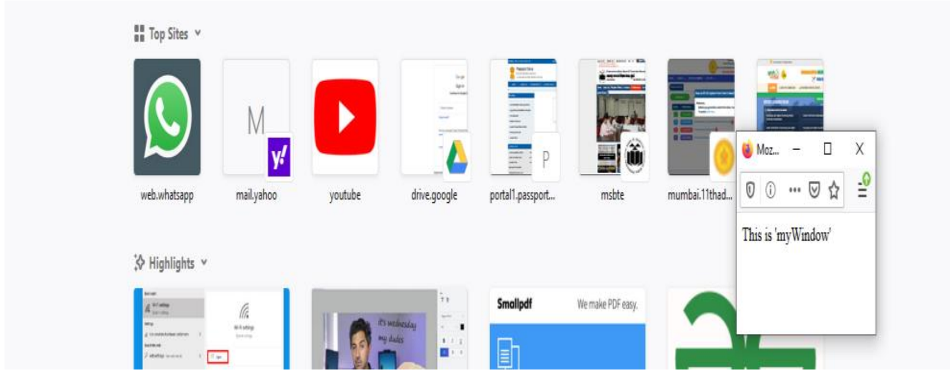
- In addition to opening and closing windows, it’s also possible to move and resize windows.
  - moveby(x,y): moves a window a specified number of pixels.
  - moveto(x,y): moves a window’s left and top edge to the specified coordinates.
  - resizeby(width, height): resizes a window by the specified amount, relative to its current size.
  - resizeto(x,y): resizes a window to the specified width and height.

Window Position

```
<html>
<body>
<p>Open "myWindow" and move the new window 250px relative to its current position:</p>
<button onclick="openWin()">Open "myWindow"</button>
<button onclick="moveWin()">Move "myWindow"</button>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "myWindow", "width=200, height=100");
  myWindow.document.write("<p>This is 'myWindow'</p>");}
function moveWin() {
  myWindow.moveBy(250, 250);
  myWindow.focus();}
</script>
</body>
</html>
```



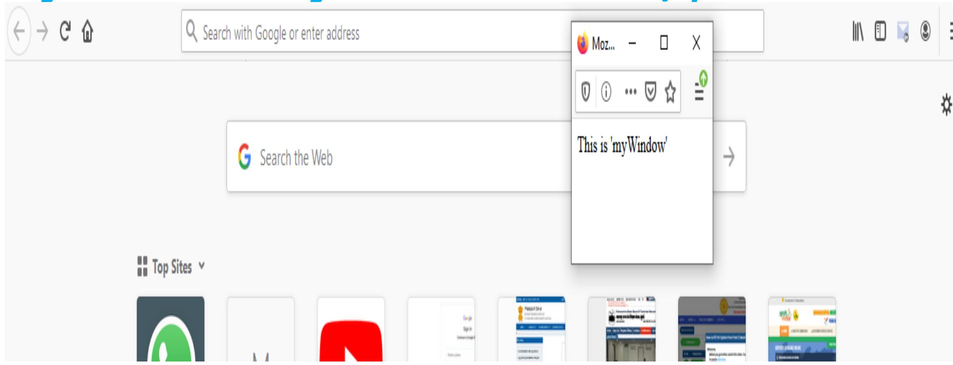
This is 'myWindow'



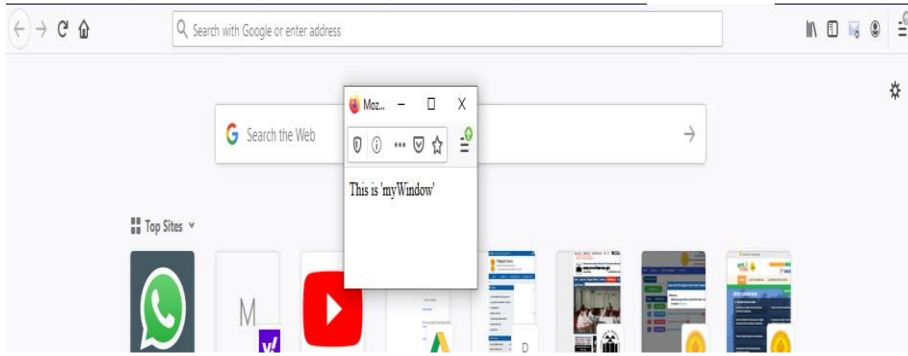
This is 'myWindow'

Window Position

```
<html>
<body>
<p>Open "myWindow" and move the new window to the top left corner of the screen:</p>
<button onclick="openWin()">Open "myWindow"</button>
<button onclick="moveWin()">Move "myWindow"</button>
<script>
var myWindow;
function openWin() {
  myWindow=window.open("", "myWindow", "width=200, height=100");
  myWindow.document.write("<p>This is 'myWindow'</p>");}
function moveWin() {
  myWindow.moveTo(500, 100);
  myWindow.focus();}
</script>
</body>
</html>
```



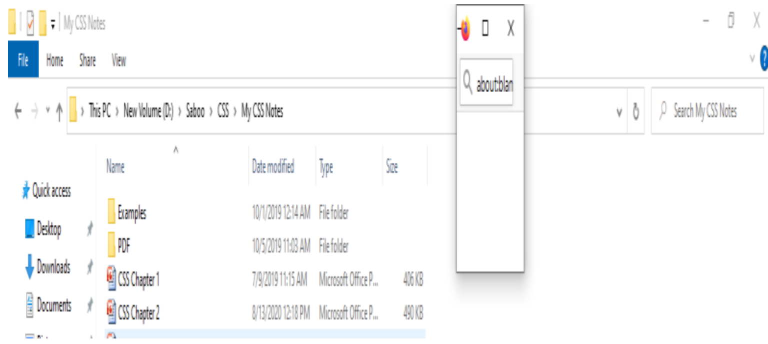
This is 'myWindow'



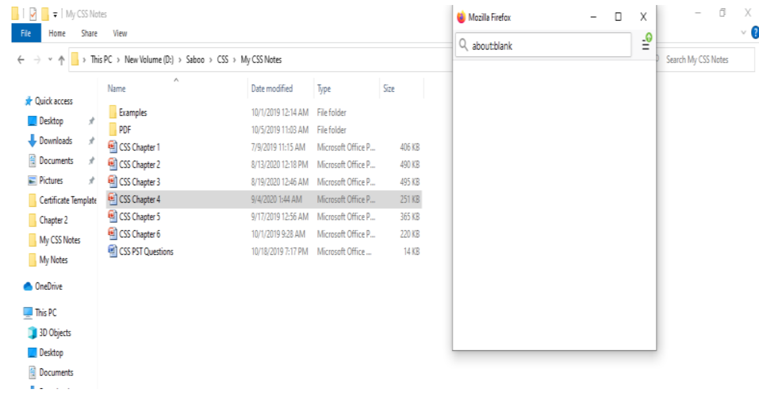
This is 'myWindow'

Window Position

```
<html>
<body>
<p>Open a new window, and resize the width and height to 250px:</p>
<p><b>Tip:</b> Press the "Resize window" multiple times (the window will increase 250px for each
   press).</p>
<button onclick="openWin()">Create window</button>
<button onclick="resizeWin()">Resize window</button>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "", "width=100, height=100");}
function resizeWin() {
  myWindow.resizeBy(250, 250);
  myWindow.focus();}
</script>
</body>
</html>
```



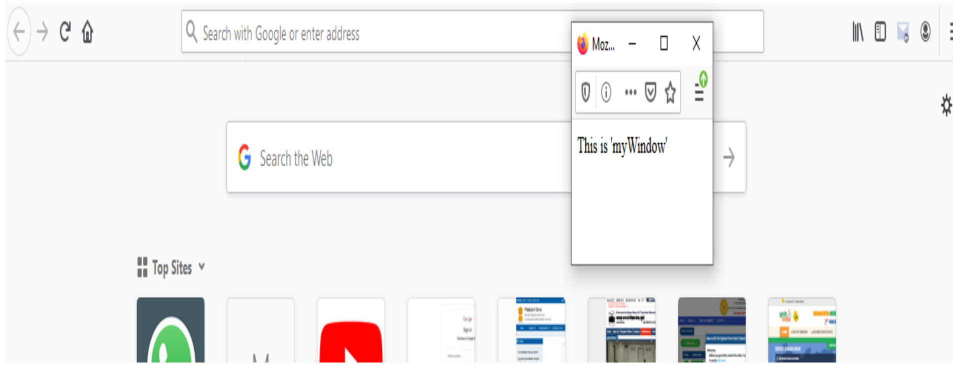
Name	Date modified	Type	Size
Examples	10/1/2019 12:14 AM	File folder	
PDF	10/5/2019 11:03 AM	File folder	
CSS Chapter 1	7/9/2019 11:15 AM	Microsoft Office P...	406 KB
CSS Chapter 2	8/13/2020 12:18 PM	Microsoft Office P...	490 KB

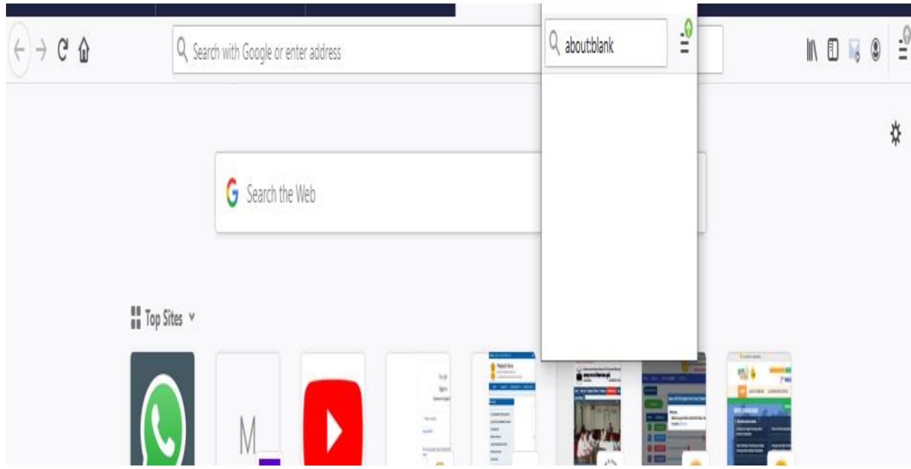


Name	Date modified	Type	Size
Examples	10/1/2019 12:14 AM	File folder	
PDF	10/5/2019 11:03 AM	File folder	
CSS Chapter 1	7/9/2019 11:15 AM	Microsoft Office P...	406 KB
CSS Chapter 2	8/13/2020 12:18 PM	Microsoft Office P...	490 KB
CSS Chapter 3	8/19/2020 12:46 AM	Microsoft Office P...	495 KB
CSS Chapter 4	9/4/2020 1:44 AM	Microsoft Office P...	251 KB
CSS Chapter 5	9/17/2019 12:56 AM	Microsoft Office P...	365 KB
CSS Chapter 6	10/1/2019 9:28 AM	Microsoft Office P...	220 KB
CSS PST Questions	10/18/2019 7:17 PM	Microsoft Office ...	14 KB

Window Position

```
<html>
<body>
<p>Open a new window, and resize the width and height to 500px:</p>
<button onclick="openWin()">Create window</button>
<button onclick="resizeWin()">Resize window</button>
<script>
var myWindow;
function openWin() {
  myWindow = window.open("", "", "width=100, height=100");}
function resizeWin() {
  myWindow.resizeTo(250, 250);
  myWindow.focus();}
</script>
</body>
</html>
```





Changing the content of window

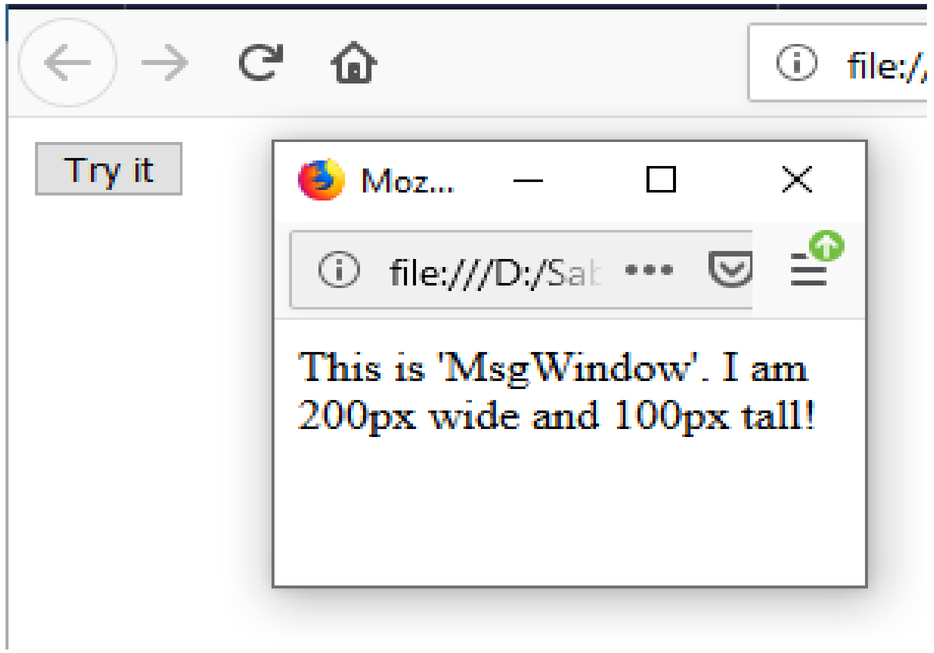
- When an HTML document is loaded into a web browser, it becomes a document object.
- This object is the root node of the HTML document.

- *document.write()* is used to write HTML expressions or JavaScript code to a document.

- It is often used to write some text to an output stream opened by the *document.open()* method.

Changing the content of window

```
<html><body>
<button onclick="myFunction()">Try it</button>
<script>
Var a=0;
function myFunction() {
var n = “w”+a;
var myWindow = window.open("", n, "width=200,height=100");
 myWindow.document.write("<p>This is 'MsgWindow'. I am 200px wide and
  100px tall!</p>");
a++;}</script>
</body>
</html>
```



This is 'MsgWindow'. I am 200px wide and 100px tall!

Closing a window

- *close()* method closes the current window.

  Syntax: *window.close()*

```
<html><body>
<button onclick="openWin()">Open w3schools.com in a new window</button>
<button onclick="closeWin()">Close the new window (w3schools.com)</button>
<script>
var myWindow;
function openWin() {
 myWindow = window.open("https://www.w3schools.com", "_blank", "width=500,
  height=500");}
function closeWin() {
 myWindow.close();}
</script>
</body></html>
```

Scrolling a window

- Following methods can be used to scroll a window.
  - scrollby()
  - scrollto()
- For this method to work, the visibility property of the window's scrollbar must be set to true.

Scrolling a window

- scrollby():
  - It scrolls the document by the specified number of pixels.

Syntax: *window.scrollby(xnum,ynum)*

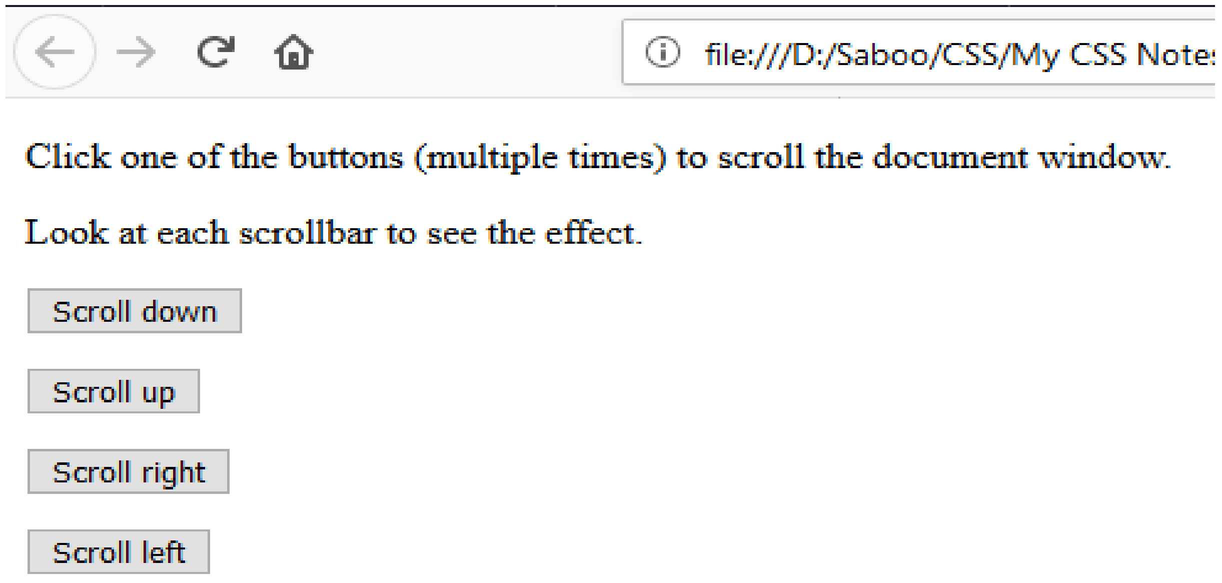
Where:

xnum: how many pixels to scroll by, along x-axis. +ve value will scroll to the right, - ve value will scroll to the left.

ynum: how many pixels to scroll by, along y-axis. +ve value will scrolldown, - ve value will scrollup.

Scrolling a window

```
<html>
<head>
<style>
body {
  height: 7500px;
  width: 5000px;}
button {
  position: fixed;}
</style></head>
<body>
<p>Click one of the buttons (multiple times) to scroll the document window.</p>
<p>Look at each scrollbar to see the effect.</p>
<button onclick="scrollWin(0, 50)">Scroll down</button><br><br>
<button onclick="scrollWin(0, -50)">Scroll up</button><br><br>
<button onclick="scrollWin(100, 0)">Scroll right</button><br><br>
<button onclick="scrollWin(-100, 0)">Scroll left</button><br><br>
<script>
function scrollWin(x, y) {
  window.scrollBy(x, y);}
</script></body></html>
```


file:///D:/Saboo/CSS/My CSS Note

Click one of the buttons (multiple times) to scroll the document window.

Look at each scrollbar to see the effect.

Scrolling a window

- scrollto():
  - It scrolls the document to the specified coordinates.

Syntax: *window.scrollto(xpos,ypos)*

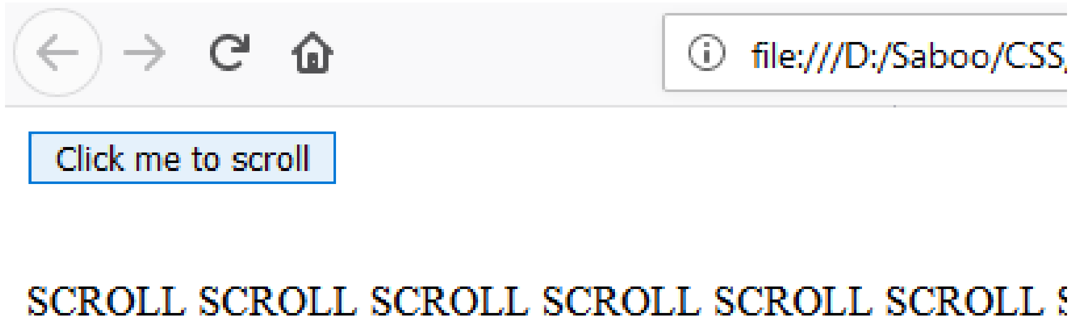
Where:

xpos: The coordinate to scroll to along x-axis.

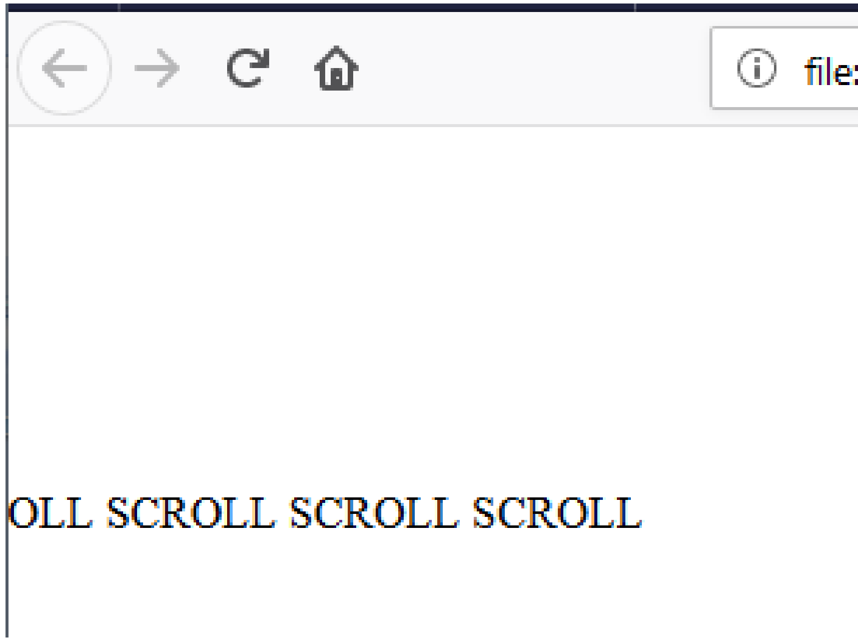
ypos: The coordinate to scroll to along y-axis.

Scrolling a window

```
<html><head><style>
body {
  width: 5000px;}
</style></head><body>
<button onclick="scrollWin()">Click me to scroll</button>
<br><br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<br><p>SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL</p>
<script>
function scrollWin() {
  window.scrollTo(300, 500);}
</script></body></html>
```



SCROLL SCROLL SCROLL SCROLL SCROLL SCROLL S



OLL SCROLL SCROLL SCROLL

Timers

- There are two types of Timers in JavaScript:
    - One-shot Timer
    - Regular Interval Timer

- One-shot timer triggers just once after a certain period of time and second type of timer continually triggers at set of intervals.

- Common uses for timers include advertisement banner pictures that change at regular intervals or display the changing time in a web page.

Timers

- One-shot Timer:

  - It can be created using *setTimeout()* method of *window* object.

    Syntax: *window.setTimeout(function,milliseconds);*

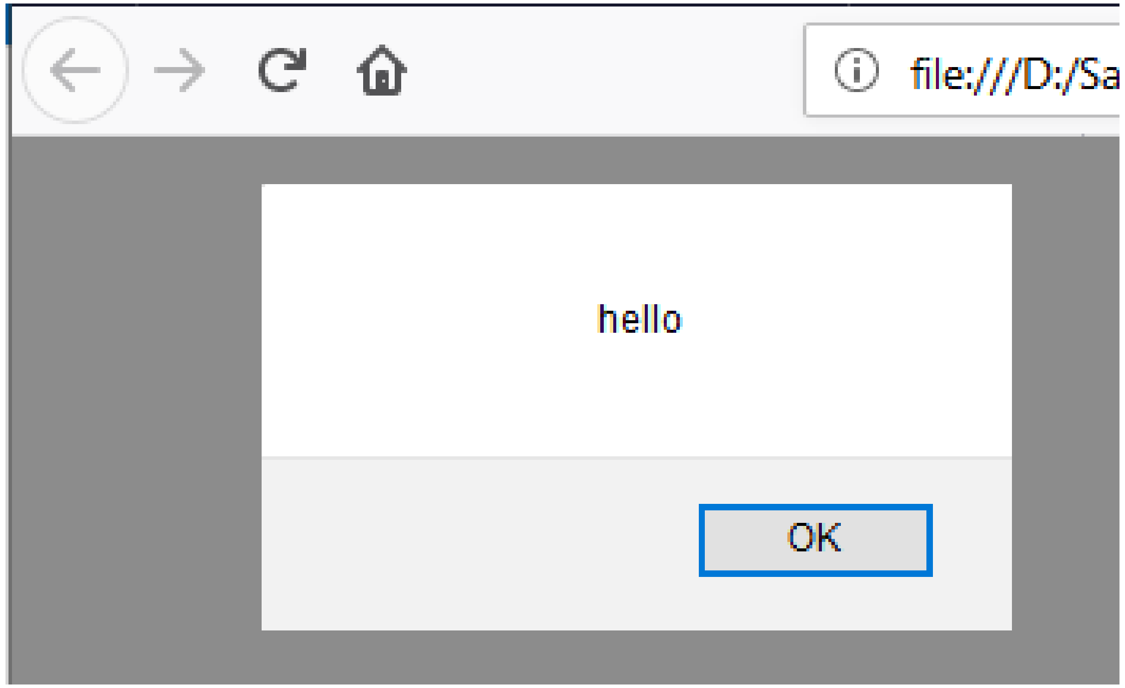
    Where:

    function: it is a function you want to execute.
    millisecond: millisecond delay until the code is executed.

  - Method returns the timer's unique ID, an integer value. It can be used to stop the timer firing .

Timers

```
<html>
<head>
</head>
<body onload="window_onload()">
<script>
var timerID;
function window_onload()
{
setTimeout(func,3000); }
function func()
{
  alert("hello");}
</script></body>
</html>
```



file:///D:/Sa

hello

Timers

- One-shot Timer:

  - *clearTimeout()* method is used to stop the execution of the function specified in *setTimeout()*.

    Syntax: window.clearTimeout(timeoutVariable)

    where:

    timeoutVariable: it is the variable returned from setTimeout() method.

  - E.g. myVar = setTimeout(*function*, *milliseconds*);
    clearTimeout(myVar);

Timers

- Regular Interval Timer:

  - setInterval() method is used to repeat a given function at every given time-interval.

    Syntax: *window.setInterval(function,milliseconds);*

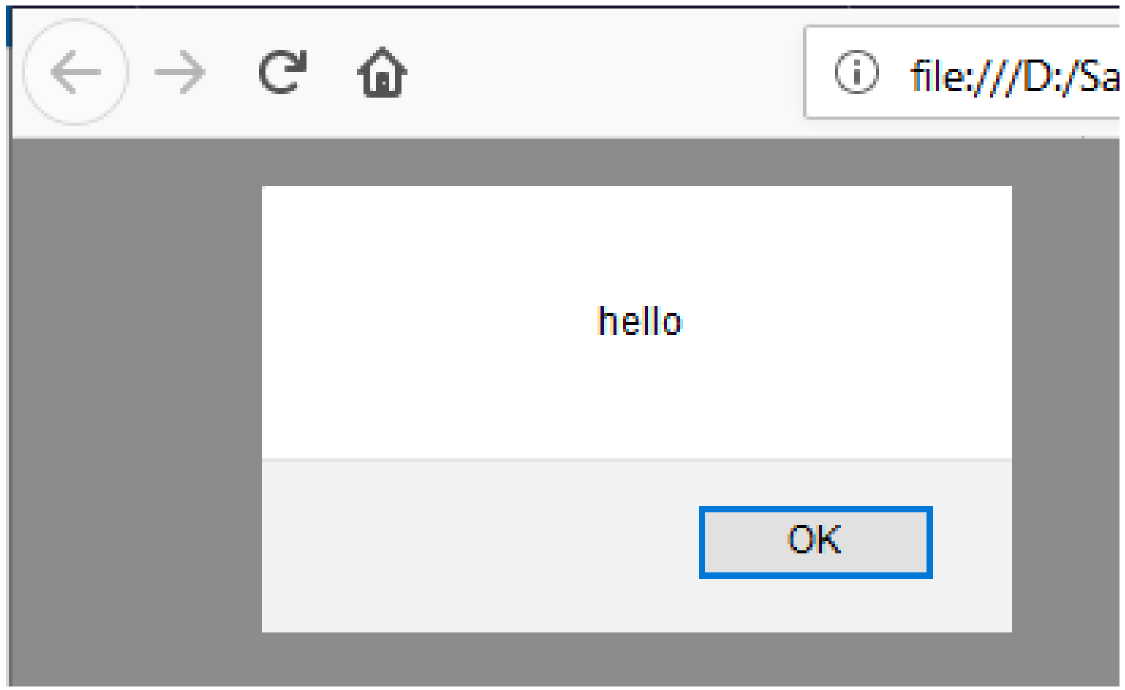
    Where:

    function: it is a function you want to execute.
    millisecond: millisecond delay until the code is executed.

  - Method returns the timer’s unique ID, an integer value. It can be used to stop the timer firing .

Timers

```
<html>
<head>
</head>
<body onload="window_onload()">
<script>
var timerID;
function window_onload()
{
setInterval(func,3000); }
function func()
{
  alert("hello"); }
</script></body>
</html>
```


file:///D:/Sa

hello

Timers

- Regular Interval Timer:

  - *clearInterval()* method is used to stop the execution of the function specified in *setTimeout()*.

    Syntax: window.clearInterval(timerVariable)

    where:

    timerVariable: it is the variable returned from setTimeout() method.

  - E.g. myVar = setInterval(*function*, *milliseconds*);
    clearInterval(myVar);

Browser Location

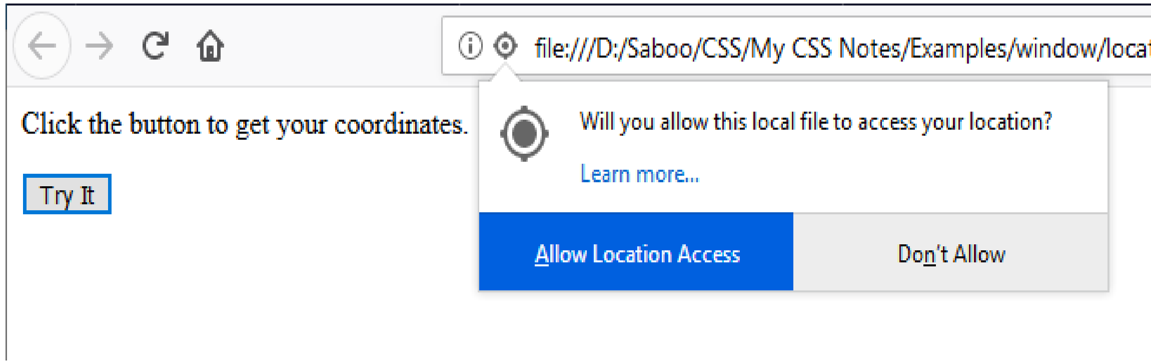
- The HTML Geolocation API is used to get the geographical position of a user. Since this can compromise privacy, the position is not available unless the user approves it.

- *getCurrentPosition()* method is used to get user’s current location.

- It returns the latitude and longitude of the user’s current location.

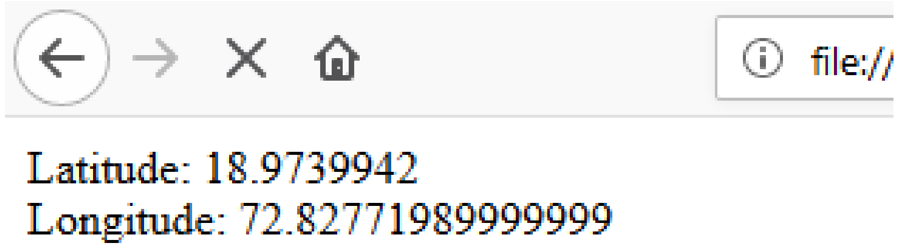
Browser Location

```
<html>
<body>
<p>Click the button to get your coordinates.</p>
<button onclick="getLocation()">Try It</button>
<script>
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else { document.write("Geolocation is not supported by this browser.");
  }}
function showPosition(position) {
document.write("Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude);
}</script></body></html>
```



Click the button to get your coordinates.

Will you allow this local file to access your location?



```
Latitude: 18.9739942
Longitude: 72.82771989999999
```

Browser Location

- *watchPosition()*: Returns the current position of the user and continues to return updated position as user moves.

- *clearWatch()*: Stops the watchPosition() method.

Browser History

- The History object contains the URLs visited by the user (within a browser window).

- The history object is part of the window object and is accessed through the window.history property.

- Property:
  - ❑ Length: Returns the number of URLs in the history list.
    - ✓It returns atleast 1, because the list includes the currently loaded page.
    - ✓Maximum length is 50.
    - ✓This property is read-only.
    - ✓Syntax: history.length

Browser History

- Methods:
  - ❑ back(): Loads the previous URL in the history list.
    - ✓It is same as clicking the back button in a browser.
    - ✓Syntax: *history.back();*
  - ❑forward(): Loads the next URL in the history list.
    - ✓It is same as clicking the forward button in a browser.
    - ✓Syntax: *history.forward();*
  - ❑go(): Loads a specific URL from the history list.
    - ✓Syntax: *history.go(number/URL);*
    - ✓The parameter can either be a number which goes to the URL within the specific position (-1 goes back one page, 1 goes forward one page), or a string. The function will go to the first URL that matches the string.