Client-side Technologies

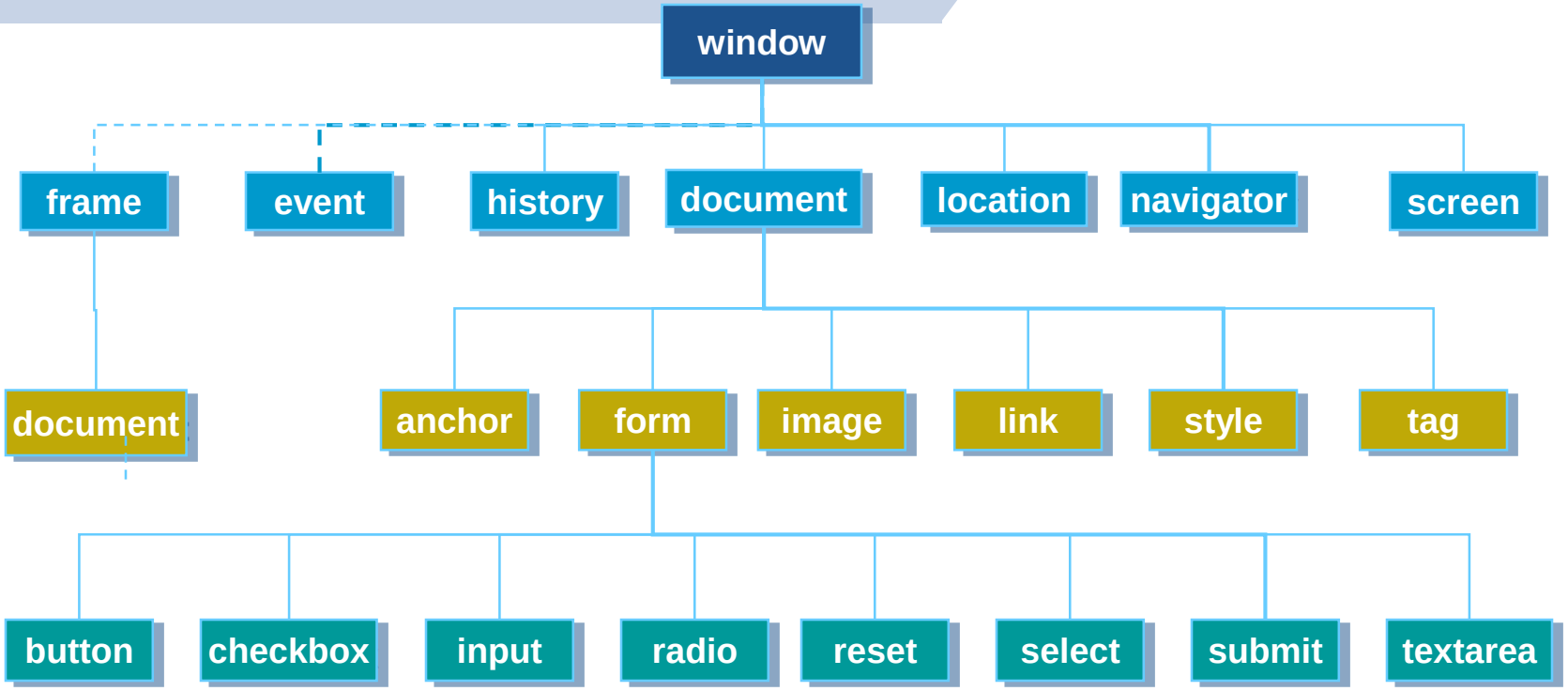
Eng. Niveen Nasr El-Den
SD & Gaming CoE
iTi

Day 5

Browser Object Model cont.

BOM

Model Hierarchy



BOM is a larger representation of everything provided by the browser including any other functionality the browser may expose to JavaScript.

Document

- The *document* object represents the entire HTML document and can be used to access all elements in a page.
- A *page* is what appears within the browser window.
- So, every window is associated with a document object.
- Document Object has its own set of Properties, Collections, Methods & Event handlers.

Document Collection

- links, anchors, images, forms are collection/array containing all occurrences of those objects within the document.
- Since they are treated as arrays, they have the *length* property which specifies the number of entries in the collection/array.
- To access a specific entry in any of these collections, we can use either their index or name.

Document Collection

Collection	Description
forms[ ]	An array containing an entry for each form in the document
images[ ]	An array containing an entry for each image in the document
anchors[ ]	An array containing an entry for each anchor in the document.
links[ ]	An array containing an entry for each link in the document.

Document Collection

- An item from an object collection can be referenced in one of the following ways:

    1. collection[i]
    2. collection[“name”]
    3. name
    4. id

- Example:

    document.forms[0]

    document.forms[“myForm”]

    document.myForm
    document.formID

Document Event Handler

onclick
ondblclick
onkeydown
onkeypress
onkeyup
onmousedown
onmouseup

Image

- The Image object is an image on an HTML form, created by using the HTML 'IMG' tag.
- Any images created in a document are then stored in an array in the document.images property.
- Image Object has its own set of Properties, Methods & Event handlers.

Image

- **Object Model Reference:**

  [window.]document.imageName
  [window.]document.imageID
  [window.]document.images[i]
  [window.]document.images[imageName]

```
<html>
<body>
 …..
<img name="img1" src="img1.jpg">
<img name="img2" src="img2.jpg">
…
</body>
</html>
```

document.img1.src="img1.jpg"
document.images[0].src="img1.jpg"

document.img2.src="img2.jpg"
document.images[1].src="img2.jpg"

Image Properties & Event handlers

Properties

name	id	src	height	width

Event handlers

onabort	onload	onerror	onclick	ondblclick	onmouseover

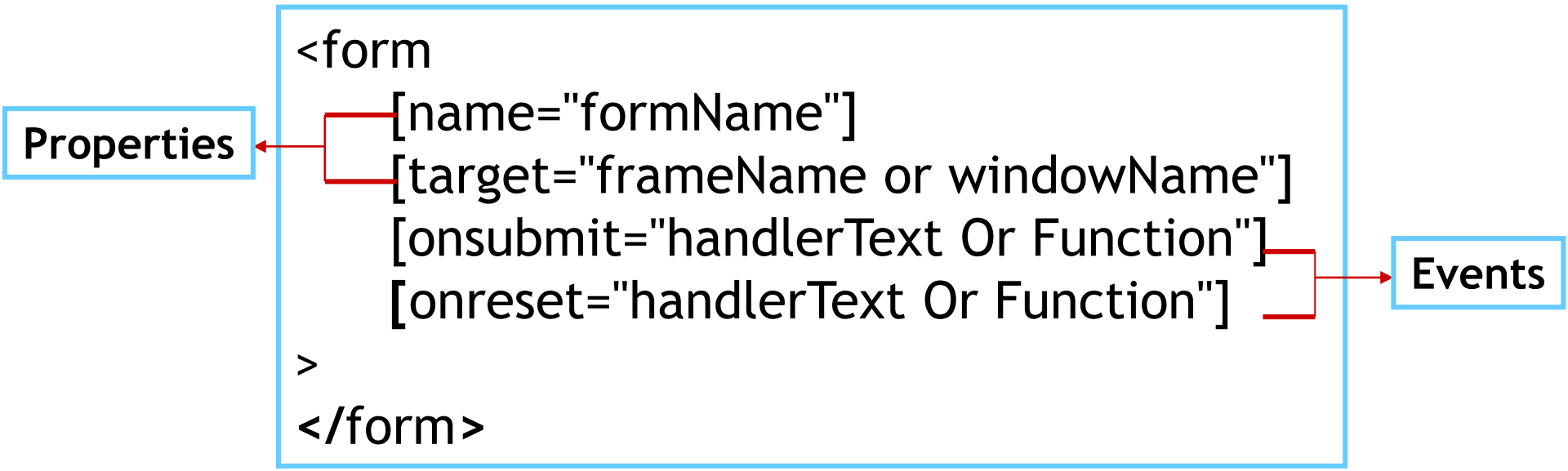
Example!

Form

- By using the form you have at your disposal; information about the elements in a form and their values.
- A separate instance of the form object is created for each form in a document.
- Objects within a form can be referred to by a numeric index or be referred to by name.
- Object Model Reference:

  [window.]document.formname
  [window.]document.forms[i]
  [window.]document.forms[“formNAME”]
  [window.]document.forms[“formID”]

Form


<form
[name="formName"]
[target="frameName or windowName"]
[onsubmit="handlerText Or Function"]
[onreset="handlerText Or Function"]
>
</form>
Properties
Events

Form Properties

Property	Description
elements[ ]	An array containing all of the elements of the form. Use it to loop through form easily.
length	The number of elements in the form.
name	The name of the form.
id	The id of the form.
target	The name of the target frame or window form is to be submitted to.

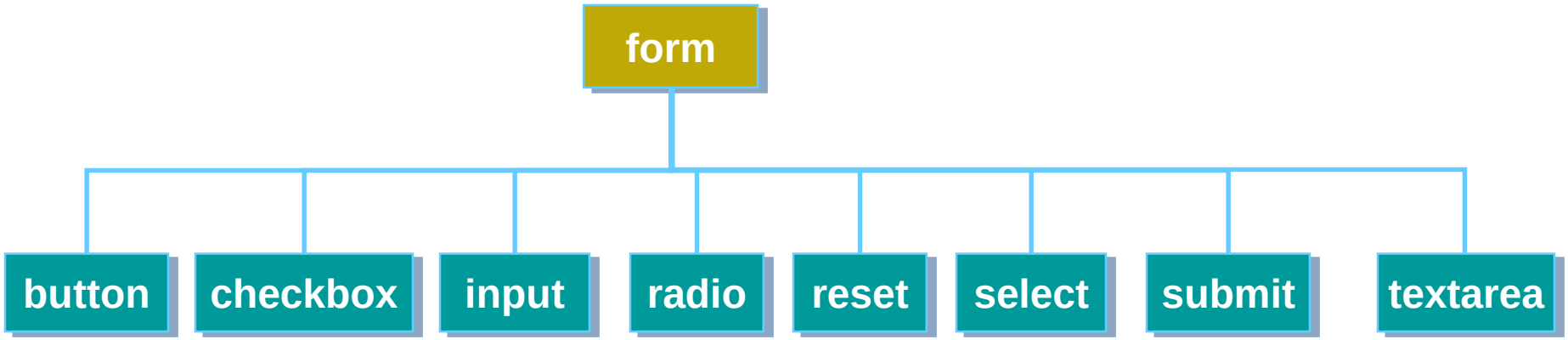
Form Methods

Method	Description
reset()	Resets the form. Clicking the reset button clears all contents that the user has made..
submit()	Submits a form. Clicking the submit button submits the content of the form to the server

Form Event Handler

Event	Description
onreset	Code is executed when the form is reset (by clicking on "reset" button)
onsubmit	Code is executed when form is submitted

Form


form
button
checkbox
input
radio
reset
select
submit
textarea

Text

```
<input type="text"
       id="id"
       value="string"
      maxlength="n"
      size=“x"
/>
```

Text Event Handler

Event	Event Handler	Description
focus	onfocus	The field gains focus when the user tabs into or clicks inside the control.
blur	onblur	The field loses focus when the user tabs from or clicks outside the control.
change	onchange	The field loses focus after the contents of the control have changed.

Text Methods

Method	Description
blur( )	**Removes focus from the field.**
focus( )	**Assigns focus to the field; places the cursor in the control.**
select( )	**Selects, or highlights, the content of the control.**

Example!

Drop-down lists

```
<select
    id="id"
    multiple
    size="n"
>
  <option value="string" selected > label </option>
  <option value="string2" > label2 </option>
  ...
</select>
```

Drop-down lists Properties

Property	Description
length	The number of options in the list.
selectedIndex	The index number, beginning with 0, of the selected option.
options[ ]	An array of the options in the list. Used to reference properties associated with the options; e.g., options[1].value or options[2].text.
selected	A true or false value indicating whether an option is chosen.
value	The value associated with an option
text	The text label associated with an option.

Drop-down lists Event Handler

Event Handler	Description
onfocus	The control gains focus.
onblur	The control loses focus.
onchange	A different option from the one currently displayed is chosen.

Example!

Radio Button

```
<input type="radio"
       id="id"
        name="name"
        value="string"
        checked
/>
```

Radio Button Properties

Property	Description
length	The number of radio buttons with the same name.
checked	A true or false value indicating whether a button is checked.
value	The value attribute coded for a button. A checked button with no assigned value is given a value of "on".

Radio Button Event Handler

Event Handler	Description
onfocus	The control gains focus.
onblur	The control loses focus.
onclick	The button is clicked.

Example!

Checkbox

```
<input type="checkbox"
      id="id"
      name="name"
      value="string"
      checked
/>
```

Button

```
<input type="button“
      id="id“
      value="string"
/>
```

Button Event Handler

Event Handler	Description
onclick	The mouse is clicked and released with the cursor positioned over the button.
ondblclick	The mouse is double-clicked with the cursor positioned over the button.
onmousedown	The mouse is pressed down with the cursor positioned over the button.
onmouseout	The mouse cursor is moved off the button.
onmouseover	The mouse cursor is moved on top of the button.
onmouseup	The mouse button is released with the cursor positioned over the button.

Reminder DOM References

- Use this keyword is used to refer to the current object.
  - e.g. the calling object in a method.
- Self reference to the object is used :

```
<input   type="text"
         onfocus = "this.value='You are in!'"/>
```

- Passing current Object as a function parameter:

```
function myFunction(myObject)
{
        myObject.value = "In the function!!"
}
<input type="text" onclick="myFunction(this)"/>
```

JavaScript Cookies

Cookies

- Cookies are small text strings that you can store on the computers of people that visit your Web site.
- Cookies were originally invented by Netscape to give ‘memory‘ to web servers and browsers.
- Normally, cookies are simple variables set by the server in the browser and returned to the server every time the browser accesses a page on the same server.
- A cookie is not a script, it is a mechanism of the HTTP server accessible by both the client and the server.

Need Of Cookies

- HTTP is a state-less protocol; which means that once the server has sent a page to a browser requesting it, it doesn't remember any thing about it.

- The HTTP protocol, is responsible for arranging:
  - Browser requests for pages to servers.
  - The transfer of web pages to your browser.

Need Of Cookies

- *Stateless protocols* have the advantage that they require fewer resources on the server -- the resources are pushed into the client.
- But the disadvantage is that the client needs to tell the server enough information on each request to be able to get the proper answer.
- As soon as personalization was invented, this became a major problem.
- **Cookies** were invented to solve this problem.

Cookies

- ***Cookies*** are a method for a server to ask the client to store arbitrary data for use in future connections.
- They are typically used to carry persistent information from page to page through a user session or to remember data between user sessions.
- With JavaScript, you can create and read cookies in the client-side without resorting to any server-side programming.
- A cookie may be written and accessed by a script but the cookies themselves are simply passive text strings.

Types Of Cookies

- Cookies has two types:
  - Session Cookies/ Non-persistent : These cookies reside on the Web browser and have *no expiry date*. They expire as soon as the visitor closes the Web browser.
  - Persistent Cookies: These cookies have an *expiry date*, are stored on a visitor's hard drive and are read by the visitor's browser each time the visitor visits the Web site that sent the cookie.

Benefits of Cookies

- **Authentication**
  - no longer need to enter password
  - Greeting people by name.
- **Maintaining state**
  - Adventure games that use cookies to keep track of pertinent character data and the current state of the game.
- **Saving time for returning visitors**
  - The user does not have to re-enter information
- **Shopping carts**
  - By storing data as you move from one page (or frame) to another.
- **Research websites.**

Cookies Limitations

- All Browsers are preprogrammed to allow a total of 300 Cookies, after which automatic deletion based on expiry date and usage.
- Each individual domain name (site) can store 20 cookies.
- Each cookie having a maximum size of 4KB.

Cookies Facts

- A server can set, or deposit, a cookie only if a user visits that particular site.
  - i.e. one domain cannot deposit a cookie for another, and cross-domain posting is not possible.
- A server can retrieve only those cookies it has deposited.
  - i.e. one server cannot retrieve a cookie set by another.
- Cookies can be retrieved only by the Web site that created them. Therefore any cookie you create is safe from view of other Web sites.

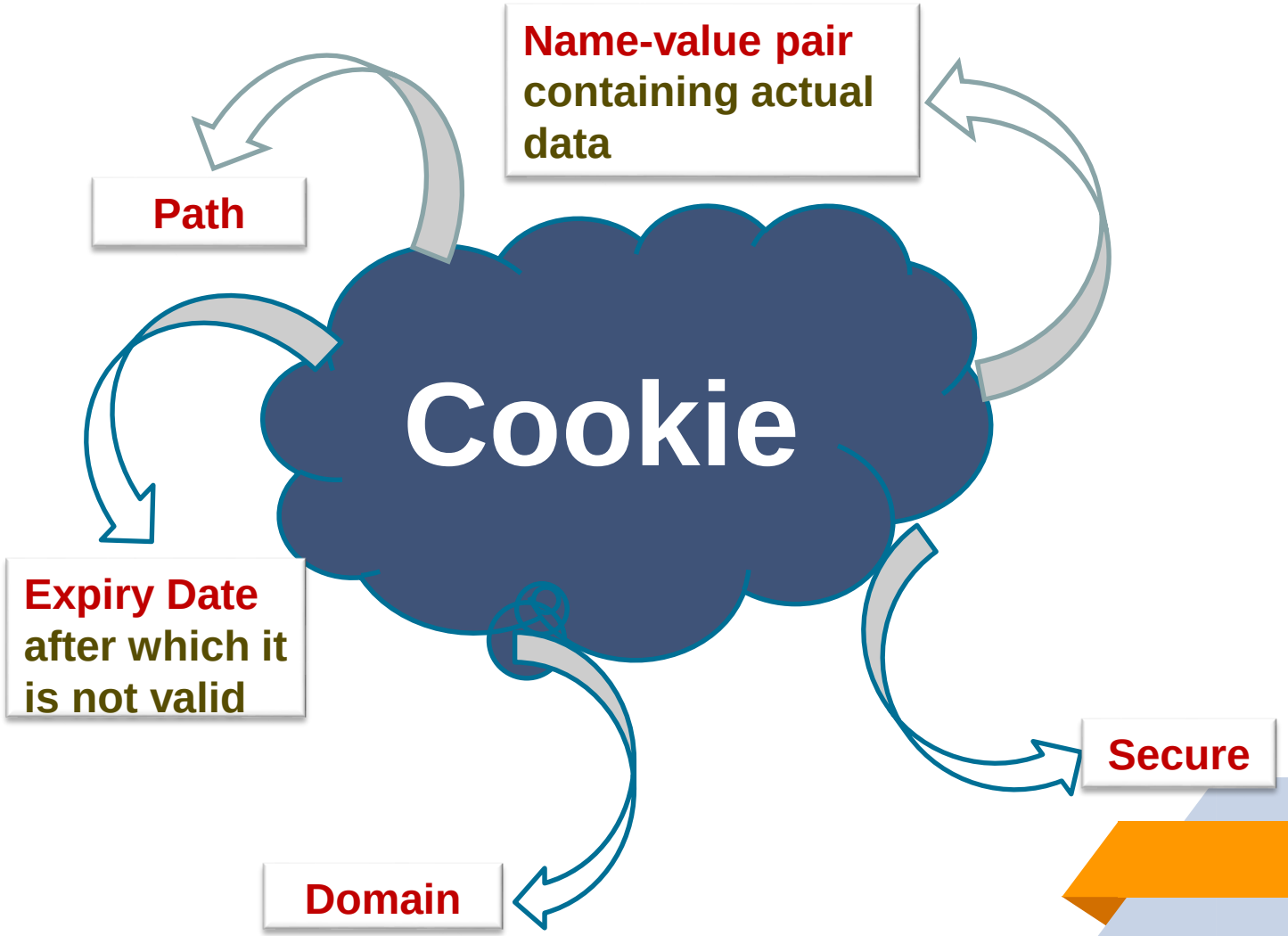
Cookies Securing Facts

- Highly unreliable, from a programming perspective.
  - It's like having your data stored on a hard drive that sometimes will be missing, corrupt, or missing the data you expected.
- Cookie security is such that only the originating domain can ever use the contents of your cookie "*Same-origin policy*".
- Cookies just identify the computer being used, not the individual using the computer.
- Cookie files stored on the client computer are easily read by any word processing program, text editor or web browsing software unless an encryption mechanism is applied.

Cookies False Claims

- Cookies are like worms and viruses in that they can erase data from the user's hard disks
- Cookies generate popups
- Cookies are used for spamming
- Cookies are only used for advertising

Cookies Parameters



Cookies Parameters

Parameter	Description	Example
name=value	This sets both cookies name and its value	username=JavaScript
expires=date	This optional value set the date that the cookie will expire on. The date should be in the format returned by toGMTString() method of the Date object. If the expires value is not given, the cookie will be destroyed the moment the browser is closed.	expires=25/3/2009 00:00:00
path=path	This optional value specifies a path within the site to which the cookies applies. Only documents in this path will be able to retrieve the cookies. Usually this is left blank, meaning that only the path that set the cookies can retrieve it.	path=/tutorials/
domain=domain	This optional value specifies a domain within which the cookie applies. Only websites in this domain will be able to retrieve the cookie. Usually this is left blank, meaning that only the domain that set the cookie can retrieve it.	domain=xyz.com
secure	This optional flag indicates that the browser should use SSL when sending the cookies to the server. This flag is rarely use.	secure

Working with Cookies

- Cookies can be *created*, *read* and *deleted* by JavaScript, under these conditions:

  1. The user's navigator must be cookie-enabled. This can be checked using "*navigator.cookieEnabled*" property .
  2. The cookie(s) that you set or accept are only accessible at pages with a *matching domain name*, *matching path*.
  3. The cookies must not have reached or passed their expiry date.

- When these criteria are met the cookies become available to JavaScript via the *document.cookie* property.

Creating a Cookie

- Assigning a value to the *document.cookie* property

document.cookie="name=value";

document.cookie="name=value;expires=date";

```
<head>
 <script language=”JavaScript”>
  document.cookie =  ”myCookie =“ +
                  encodeURIComponent(“This is my Cookie”);
  window.alert(“myCookie=” +
                encodeURIComponent(“This is my Cookie”));
 </script>
</head>
```

Creating a Cookie

- Assigning a value to the *document.cookie* property

document.cookie="name=value;expires=date";

```
<head>
 <script language=”JavaScript”>
   var myDate = new Date();
   document.cookie = “myCookie=” +
                        encodeURIComponent(“This is my Cookie”) +
                              “;expires=” + myDate.toGMTString();
 </script>
</head>
```

Displaying a Cookie

- **Retrieve created Cookie value**
  - Extract the name and value of the cookie to two variables.
  - The document.cookie will keep a list of name=value pairs separated by semicolons, where name is the name of a cookie and value is its string value
  - We use strings' *split()* function to break the string into key and values.

```
<head>
    <script language=”JavaScript”>
        var newCookie = document.cookie;
        var cookieParts = newCookie.split(“=”);
        var cookieName = cookieParts[0];
        var cookieValue = decodeURIComponent(cookieParts[1]);
        window.alert(cookieName);
        window.alert(cookieValue);
    </script>
</head>
```

Clearing a Cookie

- If the user logs out or explicitly asks not to save his or her username in a cookie, hence, you need to delete a cookie to remove a username cookie.

- Simply reassign the cookie, but set the expiration date to a time has already passed.

```
<head>
 <script language=”JavaScript”>
  var newDate = new Date();
  newDate.setTime(newDate.getTime() - 86400000);
  document.cookie = “myCookie=;expires=“+ newDate.toUTCString();
 </script>
</head>
```

Multiple Cookies

- Most Web browsers set limits on the number of cookies or the total number of bytes that can be consumed by the cookies from one site.
- **Creating Multiple cookies**
  - Assign each cookie in turn to the document.cookie object and ensure that each cookie has a different name, and may have a different expiration date and time.
- **Accessing Multiple Cookies**
  - more complicated since accessing document.cookie,there will be a series of cookies separated by semicolons;

CookieName=firstCookieValue;secondCookieName=secondCookieValue;etc.

Creating a Cookie Function Library

- Working with cookies requires a lot of string and date manipulation, especially when accessing existing cookies when multiple cookies have been set.
- To address this, you should create a small cookie function library for yourself that can:
  - create
  - access
  - delete

  } cookies

  without needing to rewrite the code to do this every time.

Creating a Cookie Function Library

- **getCookie (cookieName)**

  Retrieves a cookie value based on a cookie name.
- **setCookie (cookieName,cookieValue[,expiryDate])**

  Sets a cookie based on a cookie name, cookie value, and expiration date.
- **deleteCookie (cookieName):**

  Deletes a cookie based on a cookie name.
- **allCookieList ():**
  returns a list of all stored cookies
- **hasCookie (cookieName)**
  Check whether a cookie exists or not

Assignment