Client-side Technologies

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

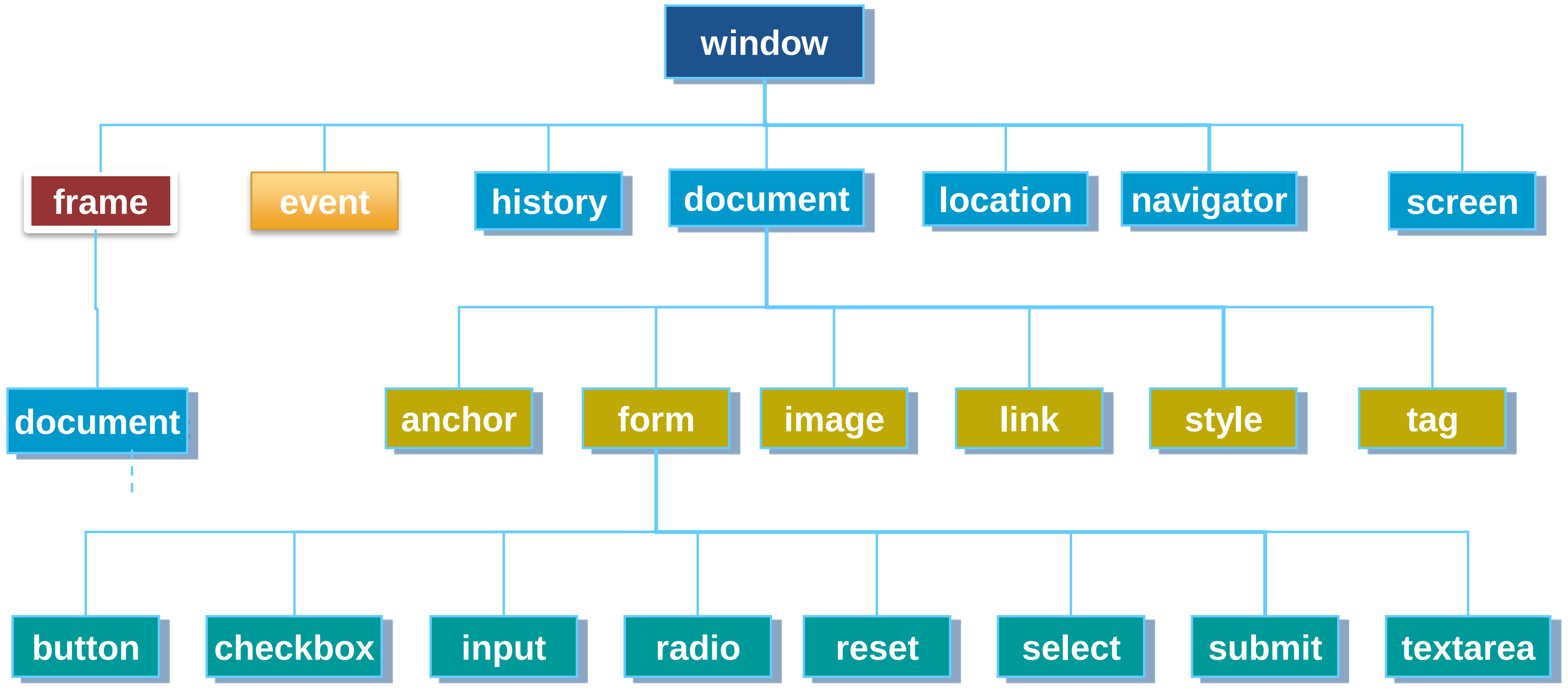
Day 6

Basics of JavaScript

Browser Object Model

BOM *cont.*

Browser Object Model Hierarchy



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

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.item(i)
  3. collection.namedItem(id)
  4. collection[“name”]
  5. collection[“id”]
  6. collection.name

- Example:

```
document.forms[0]
document.forms[“myForm”]
document.forms[“frmid”]
document.myForm
```

Document Event Handler

onclick
ondblclick
onkeydown
onkeypress
onkeyup
onmousedown
onmouseup

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.

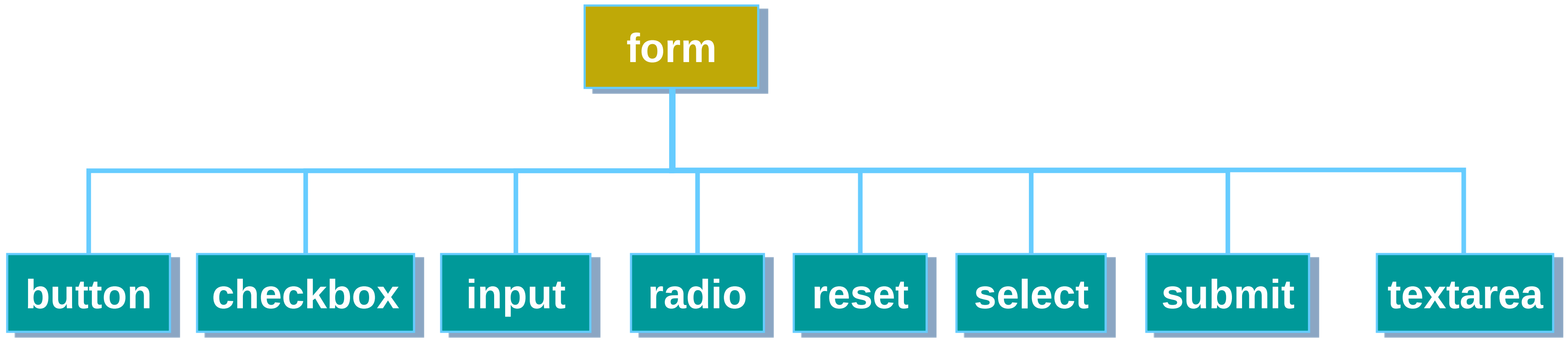
Form Methods

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	Fires when the field gains focus when the user tabs into or clicks inside the control.
blur	onblur	Fires when the field loses focus when the user tabs from or clicks outside the control.
change	onchange	Fires after changing the field value and losing being focused.
input	oninput	Fires every time the field value changes.
select	onselect	Fires when the content of the field being selected.

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)”/>
```

Events & Event Handlers

Events

- We have the ability to create dynamic web pages by using *events*.
- Events are actions that respond to user’s specific actions.
- Events are controlled in JavaScript using event handlers that indicate what actions the browser takes in response to an event.
- Examples for different events:
  - A mouse click
  - A web page loading
  - Taking mouse over an element
  - Submitting an HTML form
  - A keystroke on your keyboard

Events

- Event handlers are created as attributes added to the HTML tags in which the event is triggered. (first way of binding an event handler)
- An Event handler adopts the event name and appends the word "*on*" in front of it.

  < tag onevent = "JavaScript commands;">
- Thus the "click" event becomes the *onclick* event handler.

Mouse Events

Event handler	Description
onmousedown	when pressing any of the mouse buttons.
onmousemove	when the user moves the mouse pointer within an element.
onmouseout	when moving the mouse pointer out of an element.
onmouseup	when the user releases any mouse button pressed
onmouseover	when the user moves the mouse pointer over an element.
onclick	when clicking the left mouse button on an element.
ondblclick	when Double-clicking the left mouse button on an element.
ondragstart	When the user has begun to select an element

Keyboard Events

Event handler	Description
onkeydown	When User presses a key
onkeypress	When User presses a key other than Modifiers (ctrl, shft, ..etc.)
onkeyup	When User releases the pressed a key

Other Events

Event handler	Description
onabort	The User interrupted the transfer of an image
onblur	when loosing focus
onfocus	when setting focus
onchange	when the element has lost the focus and the content of the element has changed
onload	a document or other external element has completed downloading all the data into the browser
onunload	a document is about to be unloaded from the window
onerror	When an error has occurred in a script.
onmove	when moving the browser window

Other Events

Event handler	Description
onreset	When the user clicks the form reset button
onsubmit	When the user clicks the form submit button
onscroll	When the user adjusts an element’s scrollbar
onresize	When the user resizes a browser window
onhelp	When the user presses the F1 key
onselect	When selecting text in an input or a textarea element
onstart	When a marquee element loop begins
onfinish	When a marquee object finishes looping
onselectstart	When the user is beginning to select an element

Binding Events

- Binding Event Handlers to Elements can be:
    1. Event handlers as tag attribute
    2. Event handlers as object property

Event handlers as tag attribute

```
<input type=button value="click me" name=b1
   onclick="alert('you have made a click')">
OR
   <script>
   function showmsg()
   {
       alert("you have made a click")
   }
   </script>

<input type=button value="click me"
   onclick="showmsg()" />
```

Example!

Event handlers as object property

```
<body>
    <form>
        <input type=button name='b1' value="Click ME" />
    </form>
</body>
```

```
<script>
function showAlert ()
{
alert(“you have clicked me”)
}
document.forms[0].b1.onclick=showAlert
</script>
```

Note: there are no parentheses

Event handlers as object property

```
<body>
    <form>
        <input type=button name='b1' value="Click ME" />
    </form>
</body>
```

```
<script>
document.forms[0].b1.onclick=function ~~showAlert~~ (){
                                alert(“you have clicked me”)
                            }
</script>
```

Event handlers return value

```
<a href=“1.htm” onclick=“myFunc(); return false”>
```

This will make the browser ignore the action of href
valid only inline

- Another way that can also make the browser ignore the action of href is:

```
<a href="javascript:void(0)" onclick="alert('hi')" >
    click me
</a>
```

To avoid refresh action
if href is empty

Example!

Reminder: void Operator

- void Operator
  - A unary operator used to explicitly return undefined.
  - It can be used as shown

```
void expression;
void(expression);
```

- Example:

```
var  val= void “javascript”;
typeof val;       //undefined
```

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.

HTTP cookie

=

Web cookie

=

Browser cookie

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.

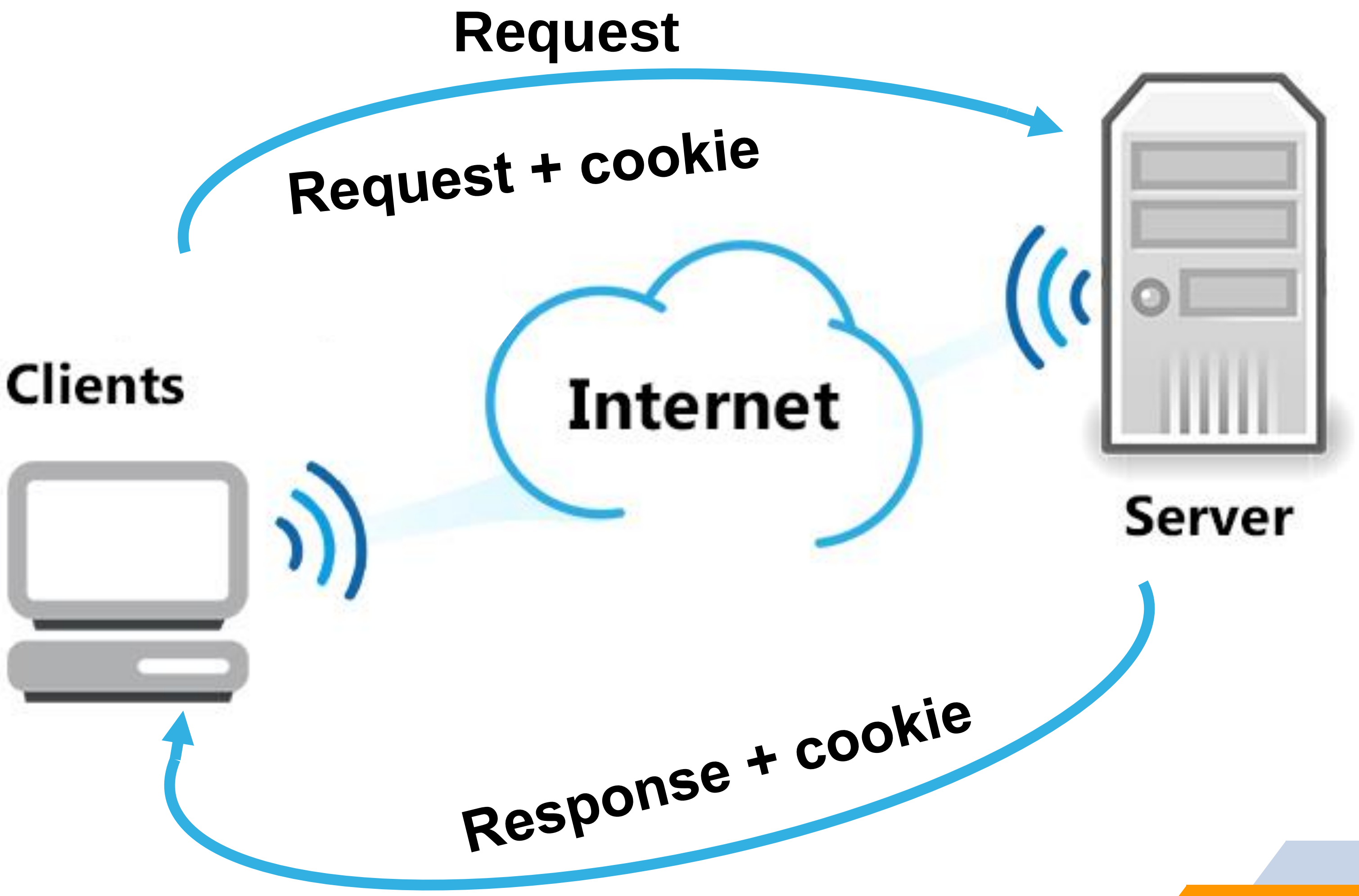
Types Of Cookies

- 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

Session Management

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


Request
Request + cookie
Clients
Internet
Server
Response + cookie

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 are sent with every request, so they can worsen performance (especially for mobile data connections).

Cookies Securing Facts

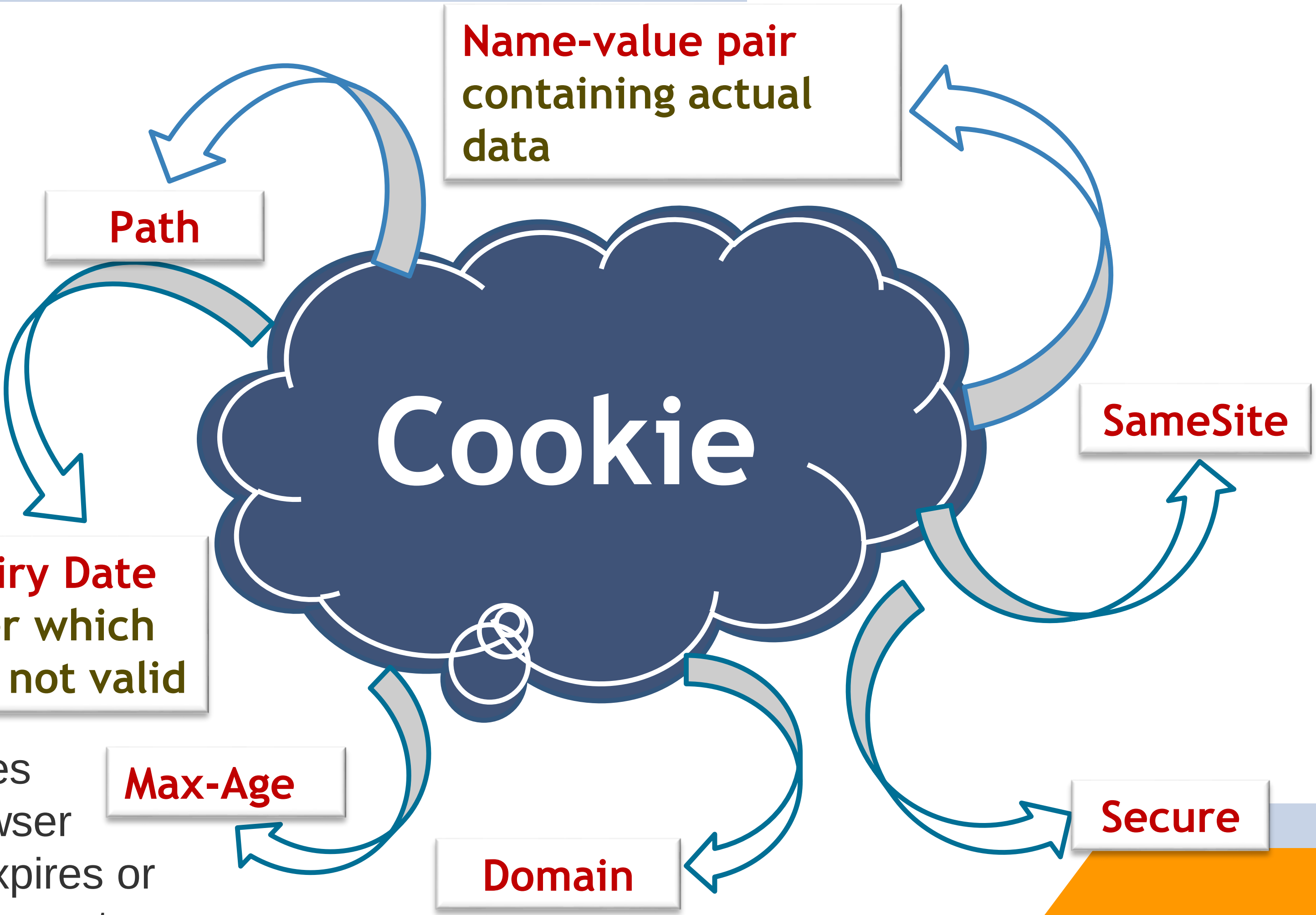
- Highly unreliable, from a programming perspective.
  - It's like having your data stored on a hard drive that sometimes will be missing, corrupt, or missing the data you expected.
- Cookie security is such that only the originating domain can ever use the contents of your cookie "*Same-origin policy*".
- 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

https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie


Name-value pair containing actual data
Path
SameSite
Cookie
Expiry Date after which it is not valid
Max-Age
Domain
Secure

To let cookies survive browser close use expires or max-age parameter

Cookies Parameters

https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

Parameter	Description	Example
name=value	This sets both cookies name and its value. The cookie value string can use encodeURIComponent() to ensure that the string is in a valid format and does not contain any disallowed char in cookie values e.g. commas, semicolons, or whitespace.	username=JavaScript
expires=date	This optional value set the date that the cookie will expire on. The date should be in the GMT format. If the expires value is not given, the cookie will be destroyed the moment the browser is closed	expires= today.toUTCString()
max-age=sec	Similar to expires but is a number of seconds till the cookie disappears. It has priority over expires If neither expires nor max-age specified it will expire at the end of session.	max-age=60*60*60*5 //5 hours

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.

When we **update** or **delete** a cookie, we should use exactly the same **path** and **domain** options as when we set it.

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:

$$\left.\begin{array}{l}\triangleright\ \text{create}\\ \triangleright\ \text{access}\\ \triangleright\ \text{delete}\end{array}\right\}\quad \text{cookies}$$

  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

Cookies

were once used for general client-side storage.

Now it is recommended to use Modern Storage APIs.

Web Storage API

&

IndexedDB.

Cookie

is a small piece of data that a server sends to the user's web browser.
The browser may store it and send it back with later requests to the same server.

Cookie

is used to tell if two requests came from the same browser keeping a user logged-in etc..

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop
- https://www.javascripttutorial.net/javascript-event-loop/
- https://vaibhavgupta.me/2018/01/20/understanding-event-loop/
- https://www.programiz.com/javascript/setInterval
- https://thisthat.dev/encode-uri-vs-encode-uri-component/

Assignment