EAST WEST UNIVERSITY

CSE479
Web Programming

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

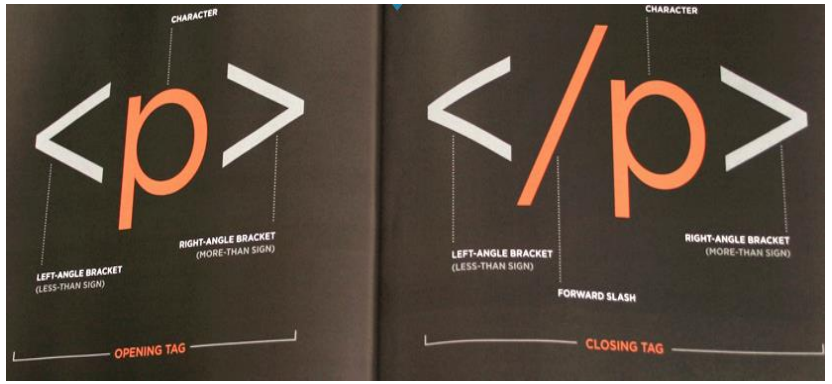
Faculty of Science and Engineering

Topic 5

The Document Object Model (DOM)

By the end of this unit you should be able to...

- ❑ Discuss why the DOM is an API
- ❑ Describe the DOM tree for an HTML page
- ❑ Locate nodes that represent the elements you want to work with
- ❑ Traverse the DOM tree for an HTML page
- ❑ Use an element's text content, child elements, and attributes
- ❑ Add and remove elements from the DOM tree of an HTML page
- ❑ Work with attribute nodes
- ❑ Describe what happens when an event occurs
- ❑ Discuss different event types
- ❑ Discuss the different steps of event handling
- ❑ Bind an event to an element
- ❑ Use parameters with event listeners
- ❑ Use properties of the event object
- ❑ Use methods of the event object



JavaScript Built-in objects fall into 3 categories

Browser object model (BOM)

- ★ Contains objects that represent the current **window** or **tab**
- ★ Contains objects that model the browser **history** & device **screen**

Document object model (DOM)

- ★ Uses objects to create a representation of the current web page
- ★ It creates a new object for each element within the page

Global JavaScript objects

- ★ Represents things that the JavaScript language needs to create a model of it
- ★ e.g., an object to represent dates and times

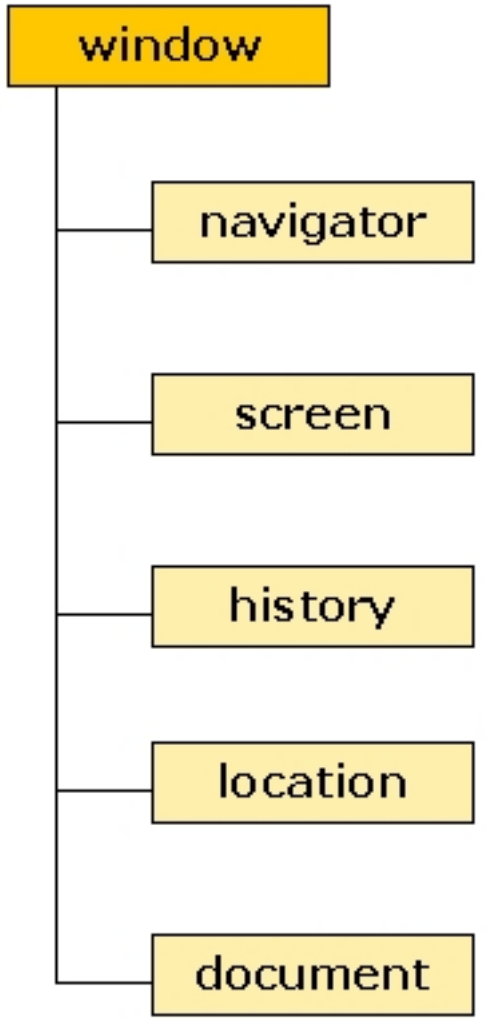
Group of objects that together model something Larger is called an *object model*

Browser object model

Creates a model of the browser window or tab

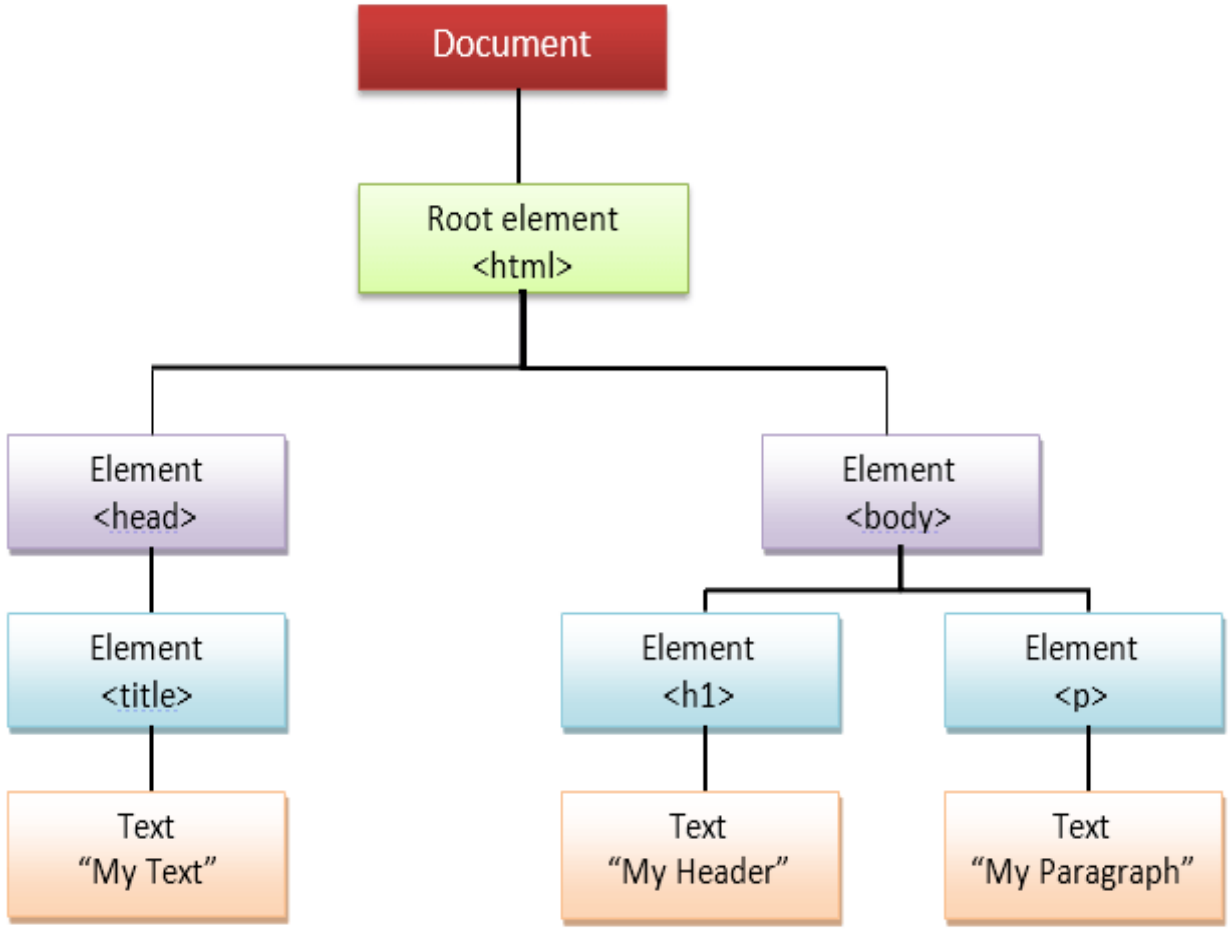
The topmost object is the ***window*** object

Its child objects represent other browser features



Document object model

- Creates a model of the current page
- The topmost object is the ***document*** object, which represents the page as a whole
- Its child objects represent other items on the page



Global JavaScript Objects

These objects do **not** form a single model

They are individual objects that relate to different parts of the JavaScript language

The names of these objects usually start with a **Capital** letter

Objects that represent basic **data types**

- ★ String
- ★ Number
- ★ Boolean

These help when working with values from these data types

Global JavaScript Objects

Objects that deal with real world concepts

- Date
  - To represent and handle dates and times
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
  - http://www.w3schools.com/jsref/jsref_obj_date.asp

- Math
  - For working with numbers and calculations
  - http://www.w3schools.com/js/js_math.asp

- Regex
  - For matching patterns within Strings of text
  - http://www.w3schools.com/jsref/jsref_obj_regexp.asp

The DOM

The Document Object Model specifies

- ★ How browsers should create a model of an HTML page
- ★ How JavaScript can access and update the contents of a web page while it is in the browser window

The DOM:

- ★ Is neither part of HTML
- ★ Nor part of JavaScript
- ★ Is a separate set of rules
- ★ Is implemented in all major browsers

Accessing and changing the HTML page

The DOM defines properties and methods to access and change each object in this model

The effect is that what the user sees in the browser window is updated

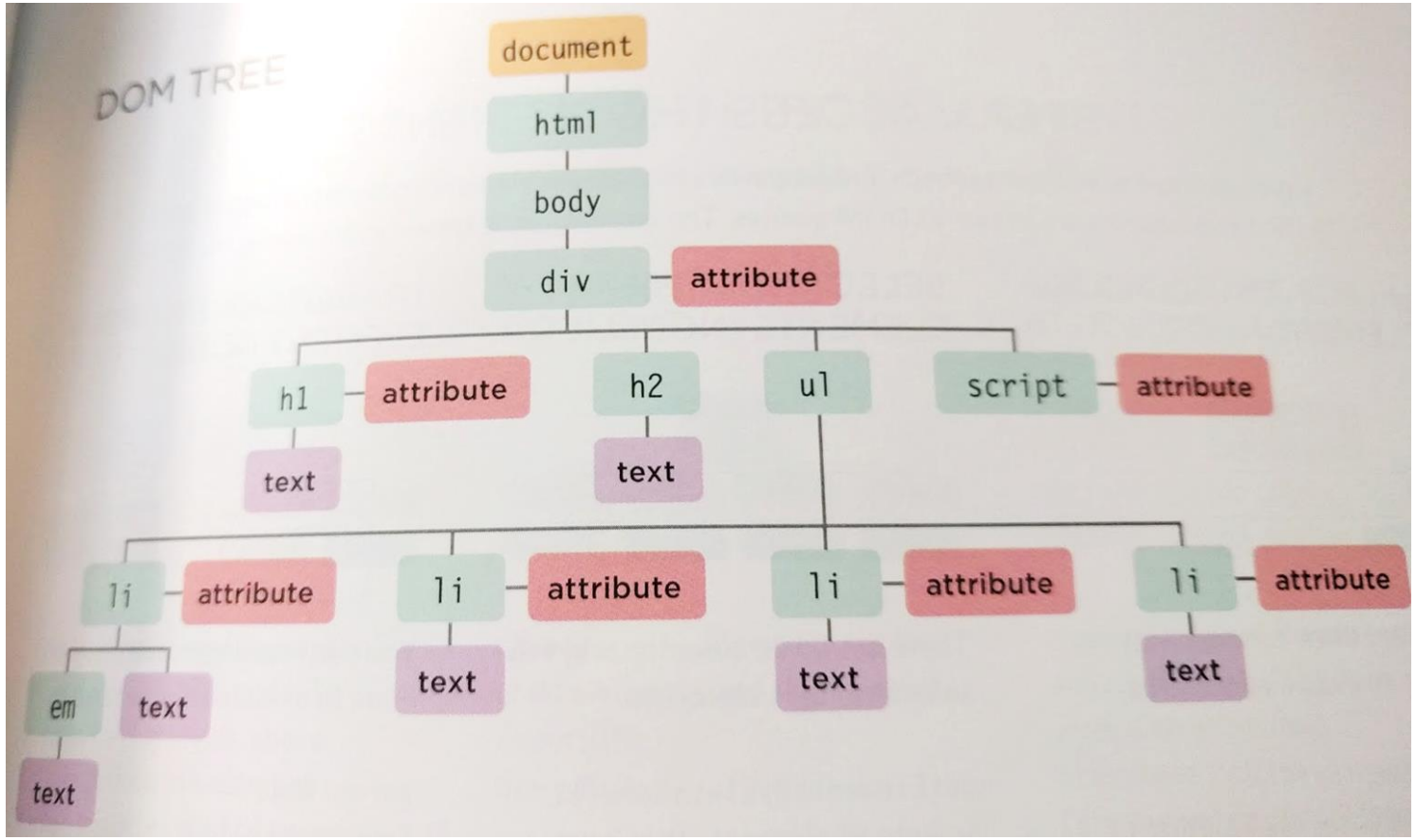
The **DOM is an API** (Application Programming Interface) - it lets the browser and your JavaScript program talk to each other

- ★ It states what your script can ask the browser about the current page
- ★ It states how your script can tell the browser to update what is being shown to the user

DOM tree is a model of a web page

Body of HTML page for shopping list

```
<!DOCTYPE html>
<html>
  <body>
  <div id="page">
    <h1 id="header">List</h1>
    <h2>Buy Groceries</h2>
    <ul>
      <li id="one" class="hot">
        <em>fresh</em> avocadoes
      </li>
      <li id="two" class="hot">
        cashew nuts
      </li>
      <li id="three" class="hot">honey</li>
      <li id="four">balsamic vinegar</li>
    </ul>
    <script scr="js/list.js"></script>
  </div>
  </body>
</html>
```



The DOM tree

When the browser loads a web page it creates a model of that page

The model is called a DOM tree

The DOM tree consists of 4 types of nodes

- ★ The document node
  - ○ Added at top and represents the entire page
  - ○ Navigate to any node via the this node
- ★ Element nodes - needed to access its text and attribute nodes
- ★ Attribute nodes - represent attributes in opening tags of HTML elements
- ★ Text nodes - used to store text within elements

Each HTML element, attribute, and piece of text in the HTML is represented by its own DOM node.

Working with the DOM tree

Accessing and updating the DOM involves two steps

- ★ Locate node that represents the element you want to work with
- ★ Use its text content, child elements, and attributes

We will use the terms elements and element nodes interchangeably.

The DOM is working with an element == the DOM is working with a node that represents that HTML element.

DOM Queries: Select an individual element node

Search entire HTML document for an element and returns its node

const el = document.getElementById("one");
Uses the value of the element's id attribute.

Oldest and best supported method.

Quickest and most efficient way to search for an element. Why?

const elFirst = document.querySelector("li.hot");
A recent addition to the DOM, so *not supported by older browsers*

Very flexible because its parameter is a CSS selector

Only returns the first match

DOM Queries: Selecting multiple elements

These methods all return a **NodeList**, a.k.a. **array of nodes**

const elements = document.getElementsByClassName("hot");
Selects all elements that have a specific value for their class attribute

Not supported in older browsers

const elements = document.getElementsByTagName("li");
Selects all elements that have a specific tag name

Safe to use in any script because widely supported.

const elements = document.querySelectorAll("li.hot");
Selects all elements that match a CSS selector

Not supported by older browsers

Traversing the DOM following 5 properties

Moving from one element node to a relating element node

When you have an element node, you can select another element in relation to it using one of the following **five properties**

- ***parentNode*** - finds the element node for the parent element in the HTML
- ***previousSibling*** - finds the previous sibling of a node if one exists
- ***nextSibling*** - finds the next sibling of a node if one exists
- ***firstChild*** - finds the first child of a node if one exists
- ***lastChild*** - finds the last child of a node if one exists

Traversing the DOM following 5 properties

```
let first = document.getElementById("one");
let next=first;

while ( next ) {
    console.log( next.innerText );
    next = next.nextElementSibling;
}
```

http://javascriptbook.com/code/c05/node-list.html

Whitespace nodes -- Problem!!

Some browsers **add a text node** whenever they come across whitespace (spaces and carriage returns) between elements. See https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Whitespace_in_the_DOM

Chrome, Firefox, Safari, and Opera

These properties return different elements in different browsers

- ***previousSibling***
- ***nextSibling***
- ***firstChild***
- ***lastChild***

Solution: jQuery or another JavaScript library

Working with selected elements

```
<li id="one" class="hot">
   <em>fresh</em> avocados
</li>
```

document.getElementById("one").textContent;

- To collect the text from the <li> element, use ***textContent***.
- Will return ***fresh avocados***
- Can also use ***textContent*** property to update the content of the element. Will replace the entire content including markup.
- Supported in all major browsers and IE9 and up

Adding or removing HTML content with innerHTML

There are two different approaches to adding and removing content from the DOM tree.

- ★ innerHTML
- ★ DOM manipulation

There are security **risks associated** with using innerHTML

innerHTML can be used on any element node.

It is used both to retrieve and to replace content (**text and markup**).

New content is provided as a string. Note: the string can contain markup.

```
let one = document.getElementById('one');
one.innerHTML='<img src="bbb" onerror=alert("hello") />';
```

Adding or removing HTML content with innerHTML

To remove the content of an element, set ***innerHTML*** to the empty string.

```
<li id="one" class="hot">
   <em>fresh</em> avocados
</li>
```

const *elContent = document.getElementById(“one”).innerHTML;*
The value of ***elContent*** would be “*fresh avocados*”

document.getElementById(“one”).innerHTML = “”;
This would remove the content of the list item, including the markup

document.getElementById(“one”).innerHTML = elContent;
This would add the content of the ***elContent*** variable, including the markup to the list item.

Adding elements using DOM manipulation

DOM manipulation offers an **alternative to innerHTML** to add new content to a page.
http://javascriptbook.com/code/c05/example.html

// Create a new element and store it in a variable. (Not in DOM tree yet)
const newEl = document.createElement('li');

// Create a text node and store it in a variable.
const newText = document.createTextNode('quinoa');

// Attach the new text node to the new element.
newEl.appendChild(newText);

// Find the position where (<ul> element in which) the new element should be added.
const position = document.getElementsByTagName('ul')[0];

// Insert the new element into its position. (Attached to DOM tree)
position.appendChild(newEl); ***// insertBefore()*** *another option*

Removing elements using DOM manipulation

DOM manipulation can be used to remove elements from the DOM tree.

- ★ Store the element to be removed in a variable
- ★ Store the parent of that element in a variable
- ★ Remove that element from its containing element

// Store the element to be removed in a variable.
const removeEl = document.getElementsByTagName('li')[3];

// Find the element which contains the element to be removed.
const containerEl = document.getElementsByTagName('ul')[0];

// Remove the element.
containerEl.removeChild(removeEl);

Attribute nodes

Once you have an element node, you can use other **properties** and **methods** on it to access and change its attributes

- ★ First, select the element node that carries the attribute
- ★ Use a method or property below to work with the element's attributes

Methods
element.getAttribute("id") *// gets the value of the specified attribute*
element.hasAttribute("class") *// checks if the element node has the specified attribute*
element.setAttribute("class", "cool") *// sets the value of an attribute*
element.removeAttribute("id") *// removes the specified attribute from an element node*

Properties
element.className *// gets or sets the value of the class attribute. See jQuery.addClass()*
element.id *// gets or sets the value of the id attribute*
Others - http://www.w3schools.com/jsref/dom_obj_all.asp

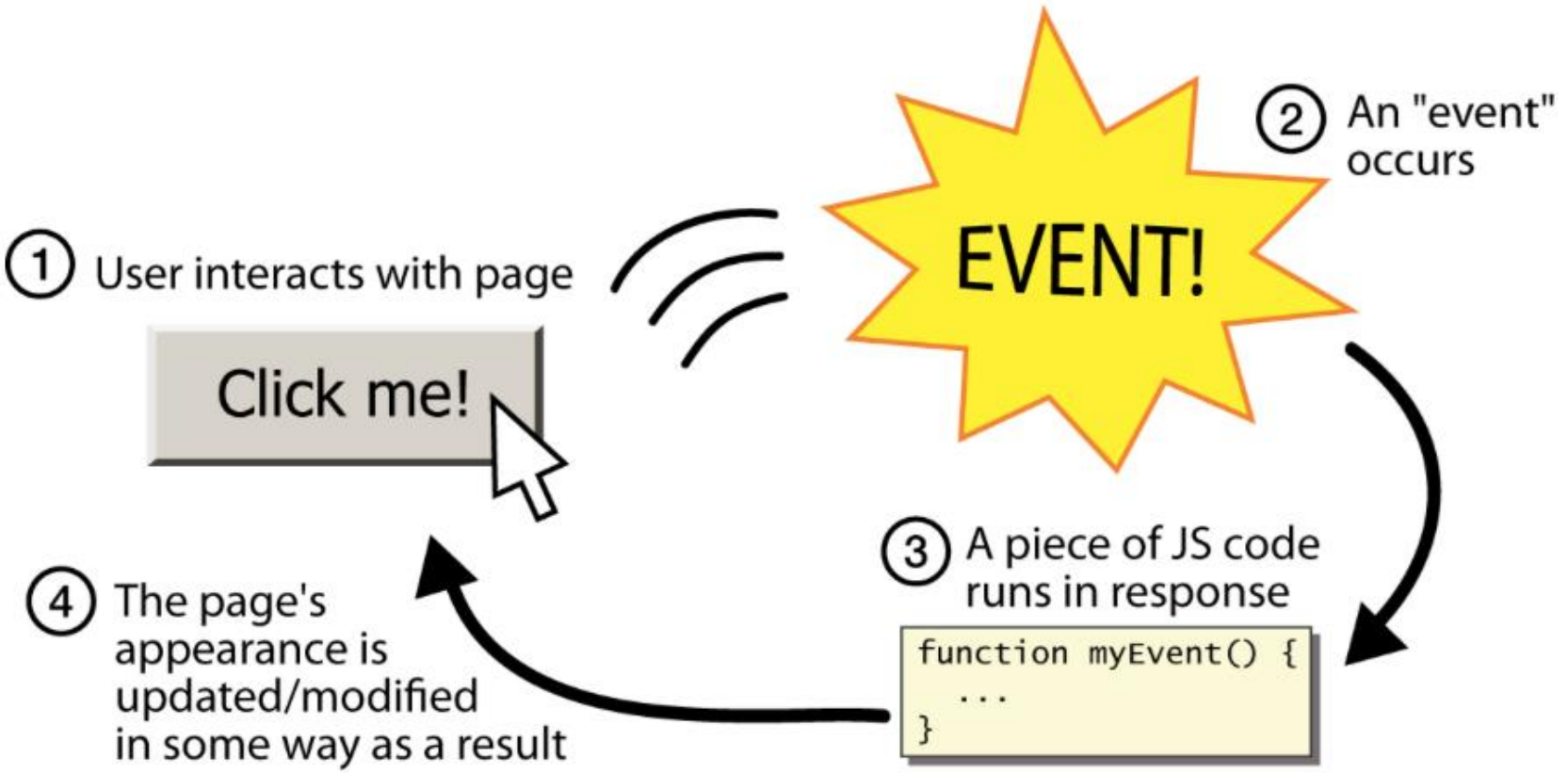
DOM events

The browser's way of saying "**Something just happened**"

Now your script can respond to these events

- ★ Scripts do so by updating the content of the page via the DOM
- ★ This makes the page more interactive

What happens when an event occurs?



Different event types

- ★ **UI Events**: occurs when user interacts with browser UI
  - ***load***, ***unload***, ***error***, ***resize***, ***scroll***
- ★ **Keyboard Events**: occurs when the user interacts with keyboard
  - ***Keydown***, ***keyup***, ***keypress***
- ★ **Mouse Events**:occurs when user interacts with a mouse, trackpad or touchscreen
  - ***click***, ***dblclick***, ***mousedown***, ***mouseup***, ***mousemove***, ***mouseover***, ***mouseout***

Different event types

- **Focus Events**: occurs when an element gains or loses focus
  - ***focus***, ***blur***
- **Form Events**: occurs when a user interacts with form element
  - ***input***, ***change***, ***submit***, ***reset***, ***cut***, ***copy***, ***paste***, ***select***
- **Mutation Events**: occurs when DOM structure has been changed by a script
  - ***DOMSubtreeModified***, ***DOMNodeInserted***, ***DOMNodeRemoved***, ***DOMNodeInsertedIntoDocument***, ***DOMNodeRemovedFromDocument***

Terminology

Events ***fire*** or are ***raised***

- ★ When an event has occurred, it is often described as having ***fired*** or ***been raised***

Events ***trigger*** scripts

- ★ Events are said to ***trigger*** a function or script

Event handling

When a user interacts with the HTML on a page, three steps are involved in running some JavaScript code

1. Select the element you want the script to respond to (element user interacted with)

1. **Bind the event to a DOM node**: indicate which event on the selected node(s) will trigger the response

1. State the code you want to run when the event occurs

Binding an event to an element

HTML event handlers: **Bad practice! Avoid!!**

- Early versions of HTML introduced a set of attributes that could respond to events on the elements they were added to.
- The attribute names matched the event names
- e.g. ***<a onclick="hide()">***

Traditional DOM event handlers: **Better than HTML event handlers**

- They let you ***separate HTML and JavaScript***
- Supported in all major browsers
- **Drawback**: can only attach a single function to any event
- ***element.onevent = functionName;***
- e.g., *tileElement.onclick = moveTile;*

Binding an event to an element - best approach

DOM Level 2 Event Listeners: **Best approach**

- Event listeners were added in an update to the DOM specs in 2000
- Allow one event to trigger multiple functions
- **Not supported in IE8 and earlier**
- ***element.addEventListener('event', functionName [, useCapture]);***
- e.g., *tileElement.addEventListener('click', moveTile, false);*

Can also call
element.removeEventListener*('event', functionName [, useCapture]);*
to remove an event listener

Using parameters with event listeners

Cannot have parameters after the ***functionName*** in an event listener

- ★ Workaround
  - ○ Wrap the function call in an anonymous function

```
el.addEventListener('blur',() => {
   checkUsername(5);
}, false);
```

Note: *checkUsername(length)* needs value for length parameter

The anonymous function runs when the event is raised

Event flow: 3rd argument to addEventListener()

Event flow matters when your code has event handlers

- on an element **AND**
- on one of its ancestor or descendent element

The event **always** starts at the ***root*** (window object), goes down until it hits the ***target***, and then goes back up to the ***root***

The phase where you initiate the event and the event **barrels down** the DOM from the root is known as the **Event Capturing Phase**

Up next is **Event Bubbling Phase** where your event bubbles back up to the root

https://www.kirupa.com/html5/event_capturing_bubbling_javascript.htm

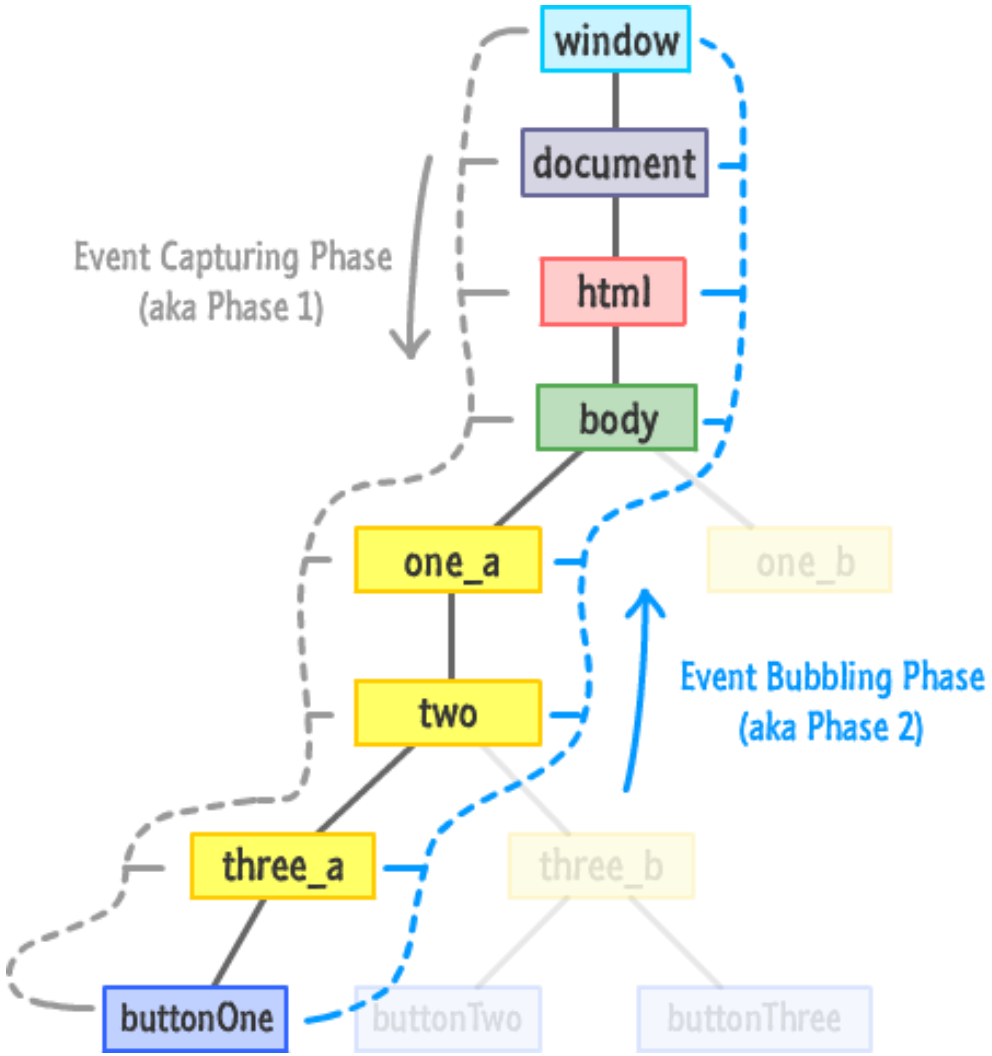
Event flow: matters only in a few cases

```
<body id="theBody" class="item">
    <div id="one_a" class="item">
        <div id="two" class="item">
            <div id="three_a" class="item">
                <button id="buttonOne" class="item">one</button>
            </div>
            <div id="three_b" class="item">
                <button id="buttonTwo" class="item">two</button>
                <button id="buttonThree" class="item">three</button>
            </div>
        </div>
    </div>
    <div id="one_b" class="item">

    </div>
</body>
```

In which phase do you handle events?

Event flow: every element is informed & notified of event



Event flow

Do you listen/respond to your event as it is goes down in the **capture phase**?

Do you listen/respond to your event as it climbs back up in the **bubbling phase**?

element.addEventListener("click", functionName, true);

- An argument of **true** means that you want to listen to the event during the **capture phase**.
- If you specify **false**, this means you want to listen for the event during the **bubbling phase**. (**this is the default**)

https://www.kirupa.com/html5/event_capturing_bubbling_javascript.htm

The event object: its properties

When an event occurs, the ***event object*** tells you information about the event, including the element it happened on

It helps answer the questions:

- ★ What is the ***type*** of the event?
- ★ Which HTML element is the ***target*** of the event?
- ★ Which ***key*** was pressed during the event?
- ★ Which ***mouse button*** was pressed during the event?
- ★ What was the ***mouse position*** during the event?

See http://www.quirksmode.org/js/events_properties.html for more details

Changing default behavior

The event object has methods that change:

- ★ **The default behavior of an element**
- ★ How an element's ancestors/descendents respond to the event

Some events (e.g., clicking on a link or submitting a form) take the user to another page. w3schools preventDefault()

To prevent the default behavior and keep the user on the same page, use the event object's ***preventDefault()*** method

```
if(event.preventDefault){
    event.preventDefault();
} else { // for IE8
    event.returnValue = false:
}
```

Changing how ancestor/descendent responds to an event

The event object has methods that change:

- ★ The default behavior of an element
- ★ **How an element's ancestor/descendent respond to the event**

Once you have handled an event using one element, you may want to stop the event from fumbling down to a descendent or bubbling up to its ancestors .
w3schools stopPropagation()

To stop the event from fumbling down or bubbling up, use the event object's ***stopPropagation()*** method

```
if(event.stopPropagation){
    event.stopPropagation();
} else { // for IE8
    event.cancelBubble = false:
}
```

Study the Examples