CSS222 WEB PROGRAMMING

LECTURE 06 – JAVASCRIPT 2

Dr. Wittawin Susutti
wittawin.sus@kmutt.ac.th

Outline

- Javascript in HTML
- async and deferred
- Event driven programming
  - Event
  - Categories
- DOM
  - Getting DOM objects
  - Adding event listeners
  - Node properties
- Event in JS
- Multiple event listeners
- Event bubbling

JavaScript in HTML `<script>` Tags

```
<!DOCTYPE html>                <!-- This is an HTML5 file -->
<html>                         <!-- The root element -->
<head>                         <!-- Title, scripts & styles can go here -->
<title>Digital Clock</title>
<style>                        /* A CSS stylesheet for the clock */
#clock {                       /* Styles apply to element with id="clock" */
  font: bold 24px sans-serif;  /* Use a big bold font */
  background: #ddf;            /* on a light bluish-gray background. */
  padding: 15px;               /* Surround it with some space */
  border: solid black 2px;     /* and a solid black border */
  border-radius: 10px;         /* with rounded corners. */
}
</style>
</head>
<body>                   <!-- The body holds the content of the document. -->
<h1>Digital Clock</h1>   <!-- Display a title. -->
<span id="clock"></span> <!-- We will insert the time into this element. -->
<script>
// Define a function to display the current time
function displayTime() {
    let clock = document.querySelector("#clock"); // Get element with id="clock"
    let now = new Date();                         // Get current time
    clock.textContent = now.toLocaleTimeString(); // Display time in the clock
}
displayTime()                   // Display the time right away
setInterval(displayTime, 1000); // And then update it every second.
</script>
</body>
</html>
```

```
<script src="scripts/digital_clock.js"></script>
```

Advantages to using the `src` attribute:

- Simplifies HTML files.
- A single copy of code.
- Downloaded once, by the first page that uses it—subsequent pages can retrieve it from the browser cache.

When scripts run: `async` and `deferred`

```
<script defer src=“deferred.js”></script>
<script async src=“async.js”></script>
```

- The **defer** attribute
  - execution of the script until after the document has been fully loaded.
  - **run in the order** in which they appear in the document

- The **async** attribute
  - run the script as soon as possible but does not block document parsing while the script is being downloaded
  - **run as they load**, which means that they may execute out of order

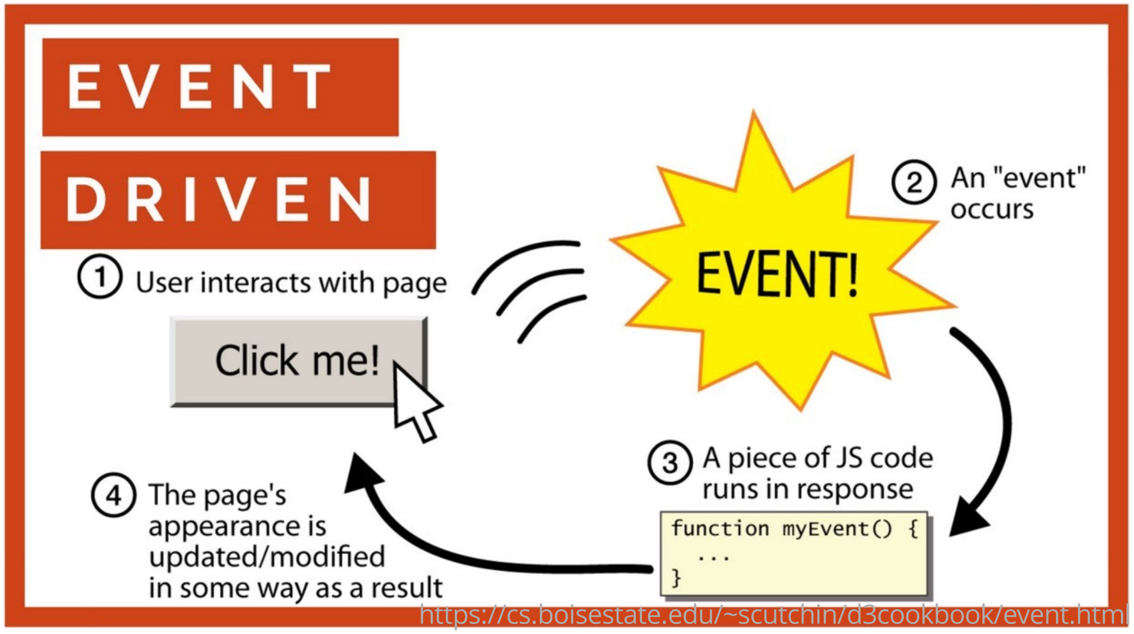
- If a `<script>` tag has both attributes, the async attribute takes precedence.

Event-driven programming

- Most JavaScript written in the browser is **event-driven.**
- The code doesn't run right away, but it executes after some event fires.
- Any function listening to that event now executes.
- This function is called an "**event handler**".

Event-driven programming

- Client-side JavaScript programs use an asynchronous event-driven programming model.


EVENT
DRIVEN
1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

https://cs.boisestate.edu/~scutchin/d3cookbook/event.html

Events

- Events can occur on any element within an HTML document
- Event model:
  - ***event type:*** specifies what kind of event occurred.
  - ***event target:*** the object on which the event occurred or which the event is associated.
  - ***event handler,*** or ***event listener:*** the function handles or responds to an event.
  - ***event propagation:*** the process which the browser decides which objects to trigger event handlers on.

Event Categories

- ***Device-dependent input events***
  - These events are directly tied to a specific input device e.g., "`mousedown`," "`mousemove`," "`mouseup`," "`touchstart`," "`touchmove`," "`touchend`," "`keydown`," and "`keyup`."

- ***Device-independent input events***
  - The "`click`" event, for example, indicates that a link or button has been activated.
  - This is often done via a mouse click, but it could also be done by keyboard or (on touch-sensitive devices) with a tap.

- ***User interface events***
  - UI events are higher-level events, often on HTML form elements that define a user interface for a web application e.g., "`focus`", "`change`", and "`submit`".

Event Categories

- ***State-change events***
  - Some events are not triggered directly by user activity, but by network or browser activity, and indicate some kind of life-cycle or state-related change e.g., `“load”` and `“DOMContentLoaded”.`

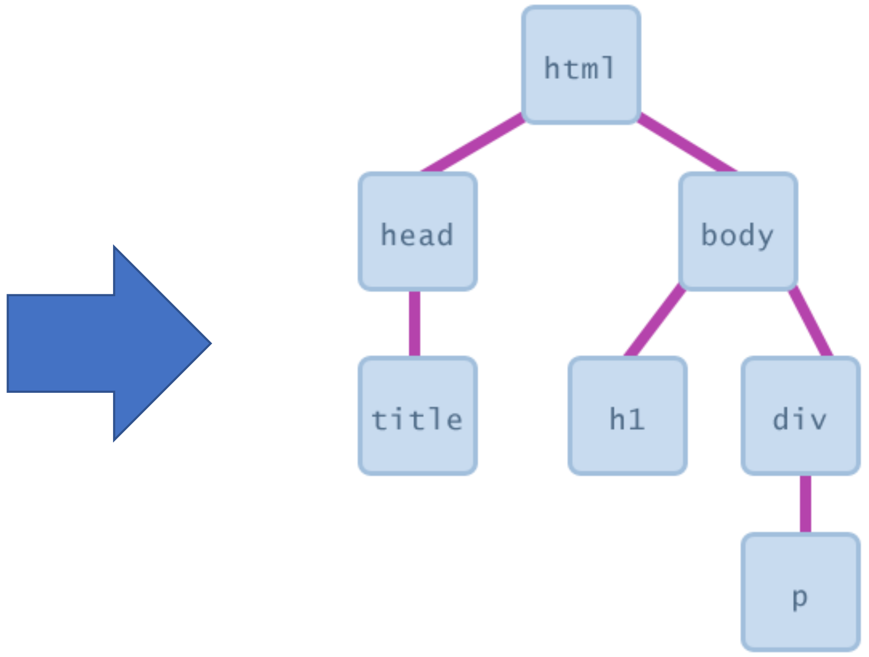
- ***API-specific events***
  - A number of web APIs defined by HTML and related specifications include their own event types e.g., `<video>` and `<audio>` elements define a long list of associated event types such as `“waiting”, “playing”, “seeking”, “volumechange”,` and so on.

The DOM

- Every element on a page is accessible in JavaScript through the **DOM**: **Document Object Model**
- The DOM is the tree of nodes corresponding to HTML elements on a page.
- Can modify, add and remove nodes on the DOM, which will modify, add, or remove the corresponding element on the page.

The DOM

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <h1></h1>
        <div>
            <p></p>
        </div>
    </body>
</html>
```



Getting DOM objects

- We can access an HTML element's corresponding DOM object in JavaScript via the `querySelector` function:

```
document.querySelector('css_selector');
```

This returns the **first** element that matches the given CSS selector

- The `querySelectorAll` function:

```
document.querySelectorAll('css_selector');
```

Returns **all** elements that match the given CSS selector.

Getting DOM objects

```
let element = document.querySelector('#button');
```

- Returns the DOM object for HTML element with `id='button'` or null.

```
document.querySelectorAll('.quote, .comment');
```

- Return a list of DOM objects containing all elements that have a "quote" class **AND** all elements that have a "comment" class.

```
document.querySelector("div.user-panel.main input[name='login']");
```

Adding event listeners

- Each DOM object has the addEventListener method defined:

```
addEventListener(event_name, function_name);
```

- To stop listening to an event, use `removeEventListener`:

```
removeEventListener(event_name, function_name);
```

- ***event_name*** is the string name of the JavaScript event you want to listen to
  - Common ones: click, focus, blur, etc.
- ***function_name*** is the name of the JavaScript function you want to execute when the event fires

index.html

```html
<html>
  <head>
    <meta charset=“utf-8”>
    <title>First JS Example</title>
    <script src=“script.js” defer></script>
  </head>
  <body>
    <button>Click Me!</button>
  </body>
</html>
```

script.js

```javascript
function onClick() {
  console.log(‘clicked’);
}
const button = document.querySelector(‘button’);
button.addEventListener(‘click’, onClick);
```

DOM object properties

- You can access **attributes** of an HTML element via a property (field) of the DOM object

```
const image = document.querySelector('img');
image.src = 'new-picture.png';
```

Adding and removing classes

- You can control **classes** applied to an HTML element via `classList.add` and `classList.remove`:

```
const image = document.querySelector('img');
```

// Adds a CSS class called "active".

```
image.classList.add('active');
```

// Removes a CSS class called "hidden".

```
image.classList.remove('hidden');
```

Some properties of Element objects

Property	Description
`id`	The value of the id attribute of the element, as a string
`innerHTML`	The raw HTML between the starting and ending tags of an element, as a string
`textContent`	The text content of a node and its descendants.
`classList`	An object containing the classes applied to the element

Add elements via DOM

- We can create elements dynamically and add them to the web page via `createElement` and `appendChild`:

```
document.createElement(tag_string)
element.appendChild(element);
```

- Technically you can also add elements to the webpage via `innerHTML`, but it poses a security risk.

// Try **not** to use innerHTML like this:

```
element.innerHTML = '<h1>I am IRON MAN</h1>';
```

Remove elements via DOM

- We can also call remove elements from the DOM by calling the `remove()` method on the DOM object:

```
element.remove();
```

- And actually, setting the `innerHTML` of an element to an **empty string** is a fine way of removing all children from a parent node.

```
// This is fine and poses no security risk.
element.innerHTML = '';
```

Node properties

Property	Description
`textContent`	The text content of a node and its descendants. (This property is writeable)
`childNodes`	An array of this node's children (empty if a leaf)
`parentNode`	A reference to this node's parent Node

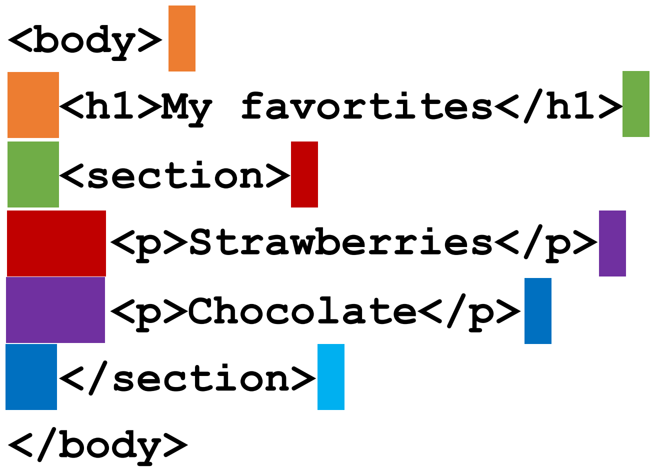
```
<body>
    <h1>My favorites</h1>
    <section>
        <p>Strawberries</p>
        <p>Chocolate</p>
    </section>
</body>
```

What's the **parentNode** of **`<section>`**?

What are the **childNodes** of **`<section>`**?

TextNode

- In addition to Element nodes, the DOM also contains Text nodes.
- All text present in the HTML, **including whitespace**, is contained in a text node:



```
<body>🟧
🟧<h1>My favortites</h1>🟩
🟩<section>🟥
🟥<p>Strawberries</p>🟪
🟪<p>Chocolate</p>🟦
🟦</section>🟦
</body>
```

DOM and Text nodes

- The DOM is composed of Nodes, and there are several subtypes of Node.
    - Element: HTML elements in the DOM
    - Text: Text content in the DOM, including whitespace
        - Text nodes cannot contain children
    - Comment: HTML comments
    - (more)
- The type of a node is stored in the nodeType property

Events in JavaScript

- If you put a `"click"` event listener on an element, what happens if the user clicks a *child* of that element?
- A click event set on an element will fire if you click on a child of that element

`Event.currentTarget` vs `target`

- You can access either the element clicked or the element to which the event listener was attached:
  - ***`event.`***`target`: the element that was clicked / "dispatched the event" (might be a child of the target)
  - ***`event.`***`currentTarget`: the element that the original event handler was attached to

Multiple event listeners

- What if you have event listeners set on both an element and a child of that element?
    - Do both fire?
    - Which fires first?

Event bubbling

- Both events fire if you click the inner element
- By default, the event listener on the inner-most element fires first
- This event ordering (inner-most to outer-most) is known as **bubbling**.