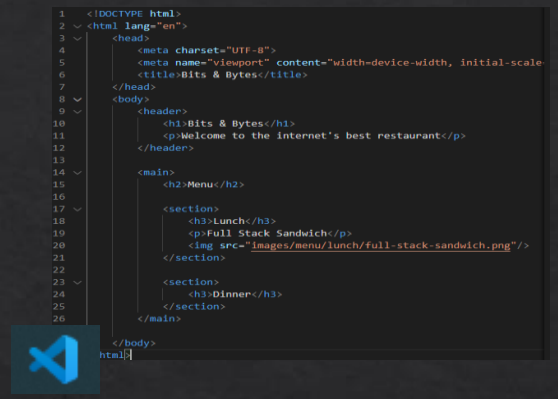
Module 3

JavaScript – Event Handling



Session Objectives:

- Select DOM objects and attach anonymous functions with addEventListener()
- Describe event bubbling and how it works
- Describe default browser behavior and what elements default behavior needs to be handled for a form
- List possible event types and what elements get those events
- Describe how to add listeners to newly created DOM elements
- Remove an event listener with removeEventListener()

Module 3
JavaScript – Event Handling

Important Definitions

Event Handling
A style of user interaction that browsers use to allow developers to react and interact with a user's use of their site.

Event handling is managed all in JavaScript and it not written into CSS or into the HTML. It is how UIs are managed on the web.

Event Handler
A function that can be attached to a DOM element and run when a defined type of event happens on that DOM element.

The function can be an anonymous function or a named function.

Event Objects
An object given to every Event Handler that defines properties about that event, like location, which DOM element the event happened on, and key presses or mouse click information.

Event object are always passed in as the first parameter to an Event Handler function.

Event Propagation
The process of an event firing on a DOM element and every parent of the DOM element up to the window object.

You can stop propagation at a certain level by calling event.stopPropagation() at that DOM element.

Default Action
The default process that a browser will perform if no Event Handler prevents it from happening.

You can stop default behavior by calling event.preventDefault().

Default behavior is most important to watch out for on buttons, forms, and a elements.

Module 3
JavaScript – Event Handling

The browser event model

Browser events work in what is commonly called a Publish and Subscribe manner. Publish and Subscribe is a programmatic way to pass messages between different parts of a system while keeping those different parts decoupled from each other, meaning that the parts don't have to know about each other, they just need to know which messages to watch out for.

Publishing means that the parts of the system can send messages out for other parts to act on and Subscribe means that a part can listen for certain messages to be published and perform logic in response to it.

Module 3

JavaScript – Event Handling

What are the events?

When an event is triggered, that event might have some information that gets sent along with it. If it's a mouse event, you'll get the X and Y coordinates of where it happened. If it's a keyboard event, you'll get the key that was pressed. This information can allow you to create very powerful user interface interactions with your JavaScript.

Where do events happen?

In the browser, events are always attached to DOM elements. You can listen to events on links, buttons, input elements, tables, table rows and any other DOM element on the page.

When you want to listen for an event in your JavaScript, you first need to select the DOM element that you want to listen for events on. Then, you'll attach a function to that DOM element that you want triggered when the event happens.

Module 3
JavaScript – Event Handling

Listening for events in JavaScript
Reacting to events in JavaScript requires three things:

1. A DOM element that you want to listen to events on
2. A specific event that you want to listen to
3. A function that holds the logic that you want to execute

All DOM elements can receive the following events:

1. Mouse Events
    1. `click` - a user has clicked on the DOM element
    2. `dblclick` - a user has double clicked on the DOM element
    3. `mouseover` - a user has moved their mouse over the DOM element
    4. `mouseout` - a user has moved their mouse out of the DOM element

Input elements, like `<input>`, `<select>`, and `<textarea>`, also trigger these events:

1. Input Events
    1. `keydown` - a user pressed down a key (including shift, alt, etc.) while on this DOM element
    2. `keyup` - a user released a key (including shift, alt, etc.) while on this DOM element
    3. `change` - a user has finished changing the value of this input element
    4. `focus` - a user has selected this input element for editing
    5. `blur` - a user has unselected this input element for editing

Form elements have these events:

1. Form Events
    1. `submit` - a user has submitted this form using a submit button or by hitting Enter on a text input element
    2. `reset` - a user has reset this form using a reset button

There are many more events that can be listened for, but these are the ones you'll use most of the time. You can find more at the MDN documentation for events.

Module 3
JavaScript – Event Handling

Adding event handlers to DOM *

```
function changeGreeting() {
  let greetingHeader = document.getElementById('greeting');
  greetingHeader.innerText = 'Goodbye';
}


let changeButton = document.getElementById('change-greeting');

changeButton.addEventListener('click', (event) => {
  changeGreeting();
});
```

*Best practice

Event handling using anonymous functions

You could also get the same functionality by attaching an anonymous function as the event listener instead of calling a named function:

```
changeButton.addEventListener('click', (event) => {
  let greetingHeader = document.getElementById('greeting');
  greetingHeader.innerText = 'Goodbye';
});
```

Module 3

JavaScript – Event Handling

Bubbling and propagation

A single event doesn't just trigger on one element. It actually triggers on many elements, if you let it.

When an event triggers, it's triggered on the element that has been clicked or changed first. The browser runs any event listeners on that element, but then it doesn't stop there.

The browser then goes to that element's parent and triggers the event there too. This process is called event propagation or event bubbling. The browser continues to do this, triggering the event on up the parent tree until it gets to the window object that's the super parent of all the elements.

Module 3
JavaScript – Event Handling

Where to add event listeners

As you may have seen in this chapter, the DOM doesn't get created until the HTML has been read in by the browser. The timing of this can't be guaranteed, so your JavaScript could, in theory, load and run before the DOM is fully ready.

But to attach event listeners to DOM elements, you need to be able to select elements from the DOM. Trying to get DOM elements from the DOM before the DOM is ready will cause errors at run time.

To make sure that the DOM is fully ready before you attach your event listeners, you can listen for an event.

When the DOM is fully loaded into a browser, the browser itself triggers an event called DOMContentLoaded on the document object. What you need to do is add all of your event listeners inside of an anonymous function that only runs once the DOMContentLoaded event is fired:

```
document.addEventListener("DOMContentLoaded", () => {
  // Register all of your event listeners here
});
```

Module 3
JavaScript – Event Handling

What about elements that are created and aren't yet on the DOM?

Often times you'll create a new DOM element yourself or from a template and want some of its elements to have event handlers on them. Those elements don't exist when the page is loaded in the browser, so any registering of events in DOMContentLoaded won't happen for those new elements.

In that case, you'll need to attach those events after creating the DOM elements. This is typically done by writing a new function that takes the new DOM element and attaches the event handlers that are needed. Then, after creating the new elements, pass that to the new function and then attach that element to the living DOM

Module 3

JavaScript – Event Handling

New Tools!

JavaScript
Event Handling

Dev Pack