Lecture 8: Asynchronous Programming and DOM

Asynchronous Programming

- JavaScript is a single-threaded language, meaning it executed one task at a time
- Asynchronous programming allows JavaScript to perform tasks without blocking the main thread, which is crucial for a responsive we experience.

The Event Loop

Call Stack – Where the JavaScript engine keeps track of functions that need to be executed.

Callback Queue – Holds functions that are ready to be executed once the call stack is empty

The Event Loop – Checks the call stack and callback queue, pushing tasks from the queue to the stack when it’s empty.

Asynchronous Methods

setTimeout – Calls callback function, after specified delay time

```
setTimeout(() => {
    console.log('heyy')
}, 5000) // 5 seconds
```

setInterval – Executes a code snippet with a fixed time delay between each calls

```
setInterval(() => {
    console.log('hey again');
}, 5000) // every 5 second
```

Asynchronous Methods

Promises – Represent a value that will be available in the future, either resolved (success) or rejected (error)

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve('Data fetched'), 2000);
});

promise.then(result => console.log(result)).catch(
    error => console.log(error)
);
```

Asynchronous Methods

Async / Await – Simplifies asynchronous code, making it more readable

```
async function foo() {
    try {
        let data = await someAsyncFunction();
        console.log(data);
    } catch(error) {
        console.log(error);
    }
}
```

Promise.all()

Promise.all() – takes an array of promises and returns a single promise. This single promise fulfills when all of the promises in the array have fulfilled, or it rejects if any of the promises reject

```
Promise.all([promise1, promise2, promise3]).then(
    results => {
        let result1 = results[0];
        let result2 = results[1];
        let result3 = results[2];
    }
).catch(err => {
    console.log(err);
});
```

Promise.race()

Promise.race() – takes an array of promises and returns a single promise that fulfills or rejects as soon as the first promise in the array fulfills or rejects

```
Promise.race([promise1, promise2, promise3]).then(
    value => {
        console.log('First resolved:', value);
    }
).catch ((err) => {
    console.log("First rejected:", err);
});
```

What is DOM?

DOM (Document Object Model) – is an object-oriented representation of a webpage, which can be modified with a scripting language like JavaScript.

The DOM represents HTML elements in a hierarchical, tree-like structure

JavaScript uses the DOM to make web pages interactive by changing content, styles, and attributes dynamically.

Basic Methods for Selecting Elements

document.getElementById(id) – Selects an element by its unique **id** attribute.

```
document.getElementById('id');
```

document.getElementsByClassName(className) – Returns a live HTMLCollection of elements with the specified class

```
document.getElementsByClassName('class');
```

Basic Methods for Selecting Elements

document.querySelector(selector) – Selects the first element matching a CSS selector

```
document.querySelector('div>p>#id');
```

document.querySelectorAll(selector) – Selects all elements matching a CSS selector, returning a static NodeList

```
document.querySelectorAll('div>p>#id');
```

Event Handling in DOM

Events allow JavaScript to respond to user interactions, like clicks, keyboard input, and page load.

Element.addEventListener(“event”, function) – Attaches an event listener to an element

Common events include: **clock, mouseover, keydown, submit** and **load**.

Event.preventDefault() – Prevents the default action associated with an event