




Welcome to the CoGrammar Lecture: Event Handling

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.



Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(Fundamental British Values: Mutual Respect and Tolerance)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Skills Bootcamp

8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✓ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks

Guided Learning Hours (GLH):

Minimum of 15 hours

Task Completion: First four tasks

Due Date: 24 March 2024

✓ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**

Software Engineering - **13 tasks**

Web Development - **13 tasks**

Due Date: 28 April 2024

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

Completion: All mandatory tasks,
including Build Your Brand and
resubmissions by study period end
Interview Invitation: Within 4 weeks
post-course
Guided Learning Hours: Minimum of
112 hours by support end date
(10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship
Outcome: Document within 12
weeks post-graduation
Relevance: Progression to
employment or related
opportunity

Agenda

- ❖ Understand the concept of event handling and its significance in web development.
- ❖ Learn about the event loop and its role in managing events in JavaScript applications.
- ❖ Explore different types of events, including UI events, system events, and custom events.
- ❖ Gain practical experience in handling UI events such as mouse clicks and keyboard inputs using JavaScript.

Event Handling

- ❖ Events are actions or occurrences that happen in the system you are programming.
- ❖ Event handling is the process of capturing, processing, and responding to events.
- ❖ Without event handling, applications would not be able to respond to user input or system events promptly.



Event Handling

- ❖ Event handling makes applications interactive by allowing them to respond to user actions in real-time.
- ❖ Different parts of the program can communicate with each other through events, enabling better coordination and functionality.



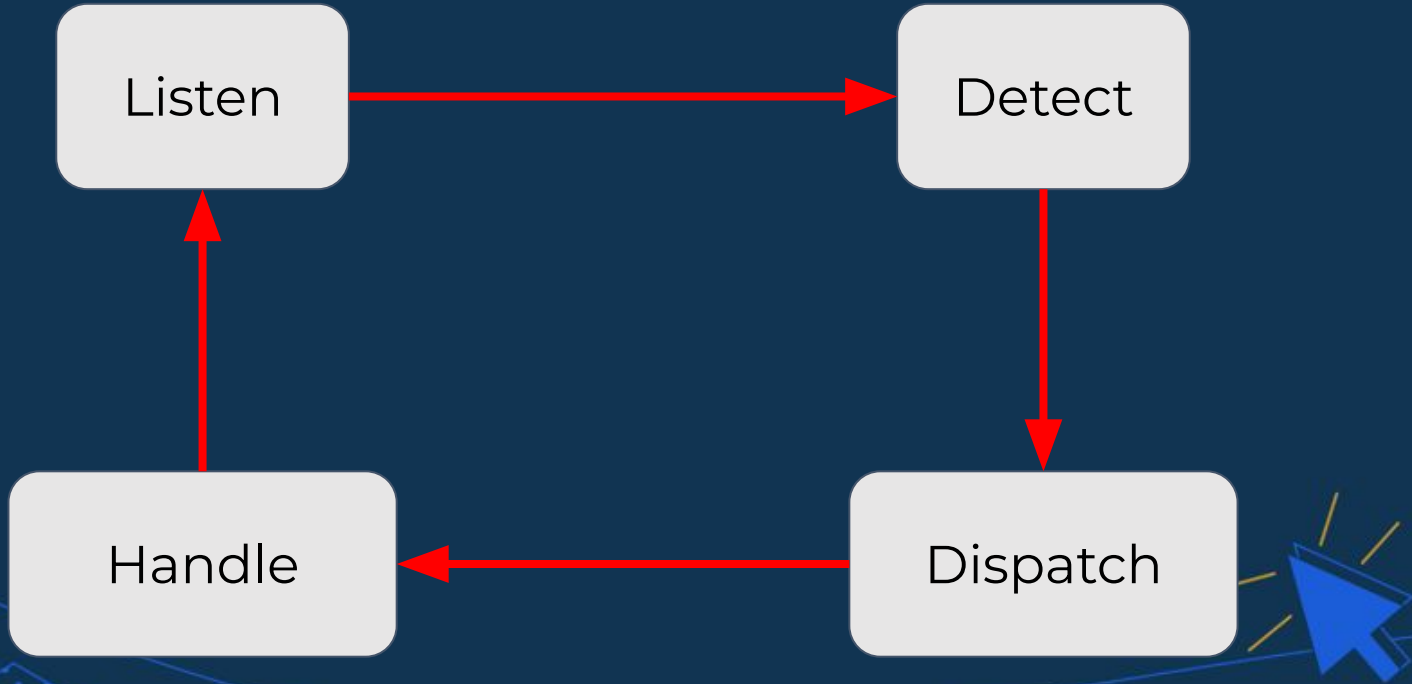
Event Loop

- ❖ Mechanism that controls the flow of events in a program, ensuring that events are processed in a timely and efficient manner.
- ❖ It allows programs to be event-driven, responding to user input and system events dynamically.
- ❖ Ensures that the program remains responsive and can handle multiple events concurrently without blocking.

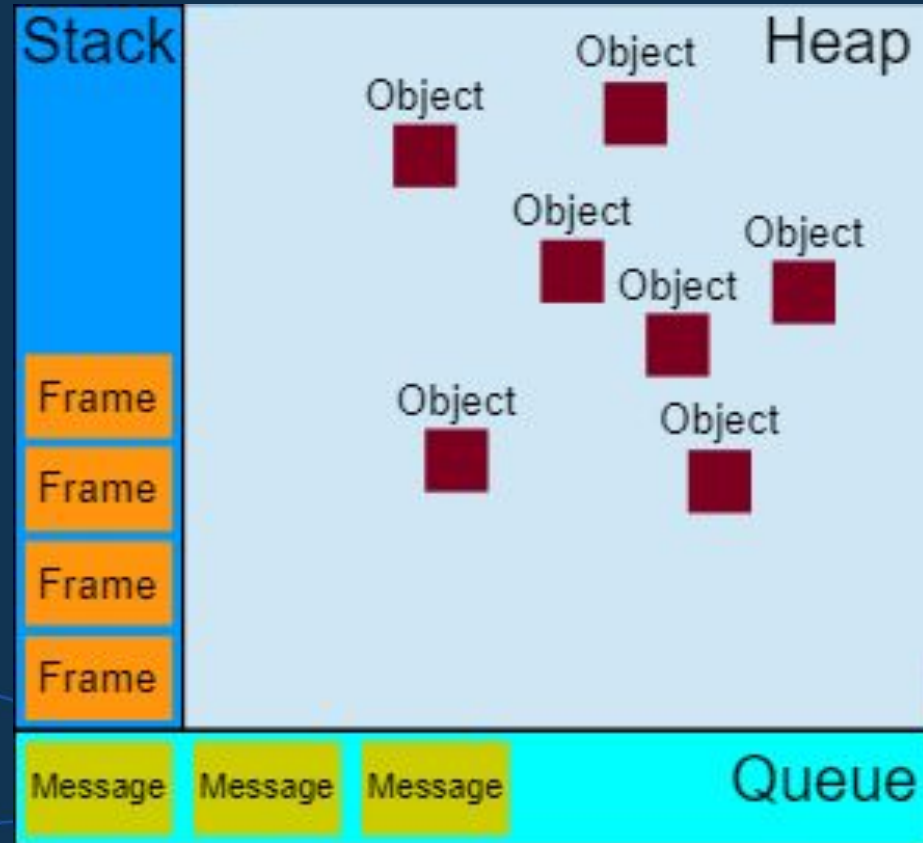
How the event loop works

1. **Listen for events:** The event loop continuously monitors for events occurring within the program or system.
2. **Event Detection:** When an event occurs, such as a mouse click or keyboard press, the event loop detects it.
3. **Event Dispatching:** The event loop then dispatches the event to the appropriate event handler or callback function.
4. **Event Handling:** The event handler associated with the event executes its code to respond to the event.
5. **Resume Listening:** After handling the event, the event loop resumes listening for more events, ensuring the program remains responsive.

How the event loop works



Event loop: Queue



Event loop: Queue

- ❖ A JavaScript runtime uses a **message queue**, which is a list of messages to be processed. Each message has an associated function that gets called to handle the message.
- ❖ At some point during the event loop, the runtime starts handling the messages on the queue, starting with the oldest one.
- ❖ To do so, the message is removed from the queue and its corresponding function is called with the message as an input parameter.

Event loop: Queue

- ❖ Calling a function creates a new stack frame for that function's use.
- ❖ The processing of functions continues until the stack is once again empty. Then, the event loop will process the next message in the queue.
- ❖ In web browsers, messages are added anytime an event occurs and there is an event listener attached to it e.g. a click on an element with a click event handler will add a message.

Event Handling in Action

- ❖ Event handling in JavaScript involves capturing and responding to various events that occur within a web page or application.

```
let button = document.getElementById("myButton");  
button.addEventListener("click", function () {  
    alert("Button clicked!");  
});
```


The event Object

- ❖ In JavaScript, when an event occurs, an event object is automatically created by the browser.
- ❖ The event object contains information about the event that occurred, such as its type, target element, and additional event-specific data.

```
document.addEventListener("click", function (event) {  
    console.log("Event Type:", event.type);  
    console.log("Target Element:", event.target);  
    console.log("Mouse Coordinates:", event.clientX, event.clientY);  
});
```


Types of Events

- ❖ **User Interface (UI) events:** Generated by user interactions with the application's interface (e.g., mouse clicks, keyboard input).
- ❖ **System events:** Generated by the system or browser (e.g., loading, error events).
- ❖ **Custom events:** Events that are defined by the programmer for specific purposes.

Let's Breathe!

Let's take a small break
before moving on to
the next topic.



Handling UI Events

- ❖ One common approach to handle UI events is by using the **addEventListener** method. This method allows developers to attach event listeners to DOM elements and specify callback functions to be executed when the events occur.

```
document.getElementById("myButton").addEventListener("click", function ()  
    alert("Button clicked!");  
});
```

Handling UI Events

- ❖ Click Event: Initiates an action when a user clicks on a button, link, or any interactive element.

```
document.getElementById("myButton").addEventListener("click", function ()  
    alert("Button clicked!");  
});
```

Handling UI Events

- ❖ Keydown Event: Captures keystrokes, allowing for keyboard-driven interactions within the application.

```
document.addEventListener("keydown", function (event) {  
  console.log("Key pressed:", event.key);  
});
```


Handling UI Events

- ❖ **Mouseover Event:** Provides feedback when the mouse cursor enters a specific area or element.

```
let element = document.getElementById("myElement");  
  
element.addEventListener("mouseover", function () {  
    console.log("Mouse over element!");  
});
```

Handling UI Events

- ❖ Mouseout Event: Triggers actions when the mouse cursor leaves a designated area or element.

```
let fetchedElement = document.getElementById("myElement");
fetchedElement.addEventListener("mouseout", function () {
  console.log("Mouse out of element!");
});
```

Handling System Events

- ❖ System events are events triggered by changes in the system or browser environment, rather than direct user interactions. These events provide valuable information about the application's state or the browser's behavior, allowing developers to respond accordingly.

```
window.addEventListener("resize", function () {  
  console.log("Window resized");  
});
```


Handling System Events

- ❖ Resize Event: Adjusts layout or UI elements dynamically in response to changes in the browser window size.

```
window.addEventListener("resize", function () {  
  console.log("Window resized");  
});
```

Handling System Events

- ❖ Load Event: Executes scripts or initializes components once the entire page and its dependencies have been loaded.

```
window.addEventListener("load", function () {  
  console.log("Page loaded!");  
});
```

Custom Events

- ❖ Events defined by the programmer for specific application requirements.
- ❖ Can be used to extend the functionality of event handling systems.
- ❖ The **Event** class provides a straightforward way to create custom events and trigger them within the application.
- ❖ The **Event** class is a built-in constructor function available in the JavaScript environment.
- ❖ It allows developers to create custom events by instantiating new event objects with specified event types and optional configuration settings.

Custom Events

- ❖ When called on a DOM element, the **dispatchEvent()** method triggers the specified custom event, causing it to propagate through the DOM tree.
- ❖ This propagation follows the same event flow as native browser events, allowing event listeners attached to the target element or its ancestors to capture and respond to the custom event.

```
// Define a custom event
var customEvent = new Event("customEvent");

// Dispatch the custom event
document.dispatchEvent(customEvent);

// Handle the custom event
document.addEventListener("customEvent", function () {
  console.log("Custom event occurred!");
});
```

Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

