



Welcome to this session: Fetching and displaying of APIs

The session will start shortly...

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



Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:



Ian Wyles
Designated Safeguarding
Lead



Simone Botes



Nurhaan Snyman



Rafiq Manan



Ronald Munodawafa



Tevin Pitts

Scan to report a
safeguarding concern



or email the Designated
Safeguarding Lead:
Ian Wyles

safeguarding@hyperiondev.com

Skills Bootcamp Cloud Web Development

- 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**

Skills Bootcamp Cloud Web Development


- 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
- If you are hearing impaired, please kindly use your computer's function through Google chrome to enable captions.

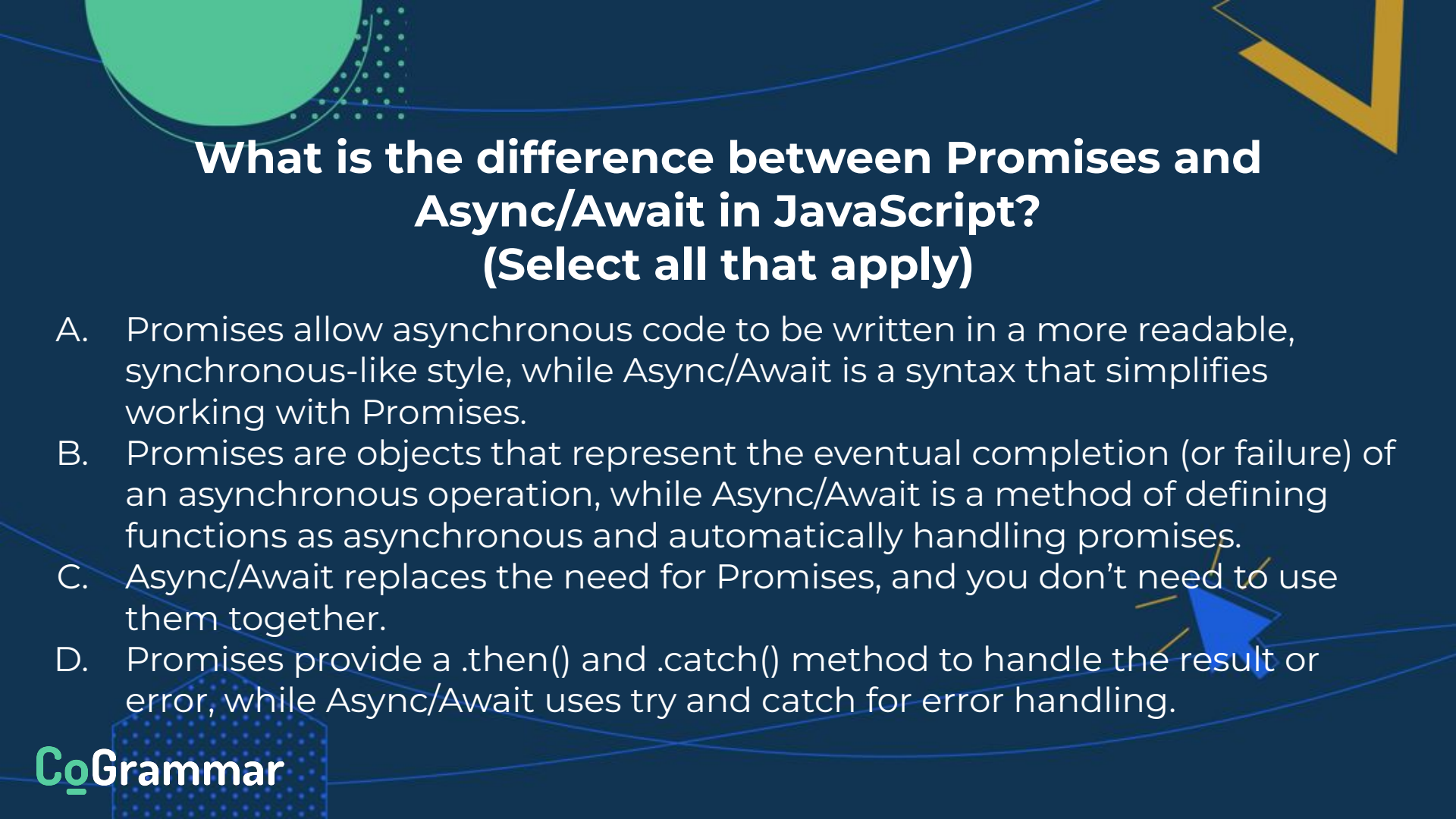
Tutorial Outcomes

- Recap on The Document Object Model (DOM), Events, APIs and Asynchronous VS Synchronous.
- Interact with an API to retrieve data which will be used to dynamically update a Website.



Why is the DOM important in Web Development, particularly for JavaScript?

- A. It allows JavaScript to manipulate and update the HTML and CSS of a webpage dynamically.
 - B. It stores JavaScript variables and functions.
 - C. It automatically formats JavaScript code to make it more readable.
 - D. It prevents JavaScript from causing errors in a webpage.
- 



What is the difference between Promises and Async/Await in JavaScript? (Select all that apply)

- A. Promises allow asynchronous code to be written in a more readable, synchronous-like style, while Async/Await is a syntax that simplifies working with Promises.
- B. Promises are objects that represent the eventual completion (or failure) of an asynchronous operation, while Async/Await is a method of defining functions as asynchronous and automatically handling promises.
- C. Async/Await replaces the need for Promises, and you don't need to use them together.
- D. Promises provide a `.then()` and `.catch()` method to handle the result or error, while Async/Await uses `try` and `catch` for error handling.

Tutorial Overview

- Recap on The Document Object Model (DOM)
- Recap on Events
- Recap on APIs
- Recap on Asynchronous VS Synchronous

The Document Object Model (DOM)

- ❖ What is the Document Object Model (DOM)?
 - **The Document Object Model (DOM)** is a programming interface for web documents.
 - It represents the **structure** of HTML documents as a hierarchical **tree** of **objects**.
 - Each **node** in the tree corresponds to a **part** of the document, such as elements, attributes, or text content.
 - The DOM **provides** a structured representation of the document, allowing **scripts** to **dynamically** access, modify, and manipulate its content and structure.

DOM Manipulation

- ❖ What is **DOM** Manipulation?
 - DOM manipulation refers to the process of **dynamically** altering the structure, content, or style of web documents using **scripting languages** like JavaScript.
 - It allows developers to create **interactive** and **dynamic** web pages by **accessing** and **modifying** elements in the Document Object Model (DOM).

DOM Manipulation

- ❖ Why is **DOM** Manipulation significant?
 - It enables developers to respond to user actions, update content dynamically, and create engaging user experiences without page reloads.
 - It forms the foundation for modern web applications, facilitating tasks such as form validation, content updates, and animation effects.
 - It empowers developers to create responsive and interactive interfaces, enhancing usability and user engagement.

Document Object

- ❖ What is the **document** object?
 - The **document** object represents the entire HTML document as a tree structure.
 - It serves as the entry point to the web page's content, allowing **manipulation** and **interaction** with its elements.
 - The **document** object serves as the root node of the Document Object Model (DOM) tree.
 - It offers various **properties** and **methods** for interacting with the document's structure and content.
 - It serves as a fundamental component for **dynamic** web development, enabling developers to create **responsive** and **interactive** user interfaces.

DOM Traversal

- ❖ DOM **traversal** involves navigating through the DOM tree to access or manipulate elements.

```
// Retrieve the element with the ID "myDiv"
let element = document.getElementById("myDiv");

// Retrieve all elements with the CSS class "container"
let containers = document.getElementsByClassName("containers");

// Retrieve all list (li) item elements
let listItems = document.getElementsByTagName("li");

// Retrieve the first list element where the parent is an ordered list
let listOrderedParent = document.querySelector("ol > li");

// Retrieve a specific list element where the parent is an ordered list
let thirdItem = document.querySelector("ol > li:nth-child(3)");

// Retrieve the first paragraph element within a container
let paragraph = document.querySelector(".container p");

// Retrieve all paragraph elements within a container
let paragraphs = document.querySelectorAll(".container p");
```

DOM Manipulation

- ❖ Creating new elements:
 - It is possible to create entirely new elements using the createElement method in JavaScript
 - `document.createElement("p");`
- ❖ Appending to elements:
 - We can add both text and new elements to existing elements in our HTML file.
 - `document.body.appendChild(myItem);`

DOM Manipulation

❖ Modifying elements:

```
// Change inner HTML content of an element
document.getElementById("myElement").innerHTML = "<strong>New content</strong>"

// Set text content of an element
document.getElementById("myElement").textContent = "Updated text content"

// Set attribute value of an element
document.getElementById("myElement").setAttribute("class", "new-class");
```


DOM Manipulation

❖ Removing elements:

```
// Get the element to remove
let elementToRemove = document.getElementById("toRemove");

// Remove the element from the DOM
elementToRemove.remove();
```

Events

- ❖ JavaScript is often used to handle events that occur on a website.
- ❖ An **event** is an **action** that occurs that your program responds to.
- ❖ DOM **events** are either things that a user does, such as clicking a button or entering text into a text box, or actions caused by the browser, such as when a web page has been completely loaded.
- ❖ **Event handling** is the process of **capturing**, **processing**, and **responding** to events.

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.
- ❖ The method has the following syntax:
 - *element*.**addEventListener**(*event*, *handler*, *options*);
 - **event**: A string representing the type of event to listen for (e.g., "click", "keydown").
 - **handler**: A function that is called when the event occurs.
 - **options** (optional): An object that can specify properties like *capture*, *once*, and *passive*.

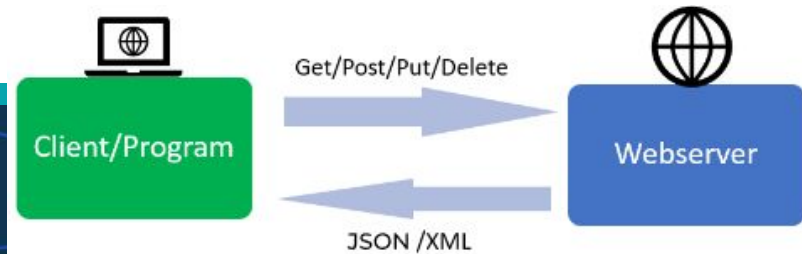
An Introduction to APIs

- ❖ **API** stands for **A**pplication **P**rogramming **I**nterface.
- ❖ An **“application”** refers to any software that interacts with other software or external services.
- ❖ An **“interface”** is the point where these interactions occur, allowing different programs to communicate with each other.



An Introduction to APIs

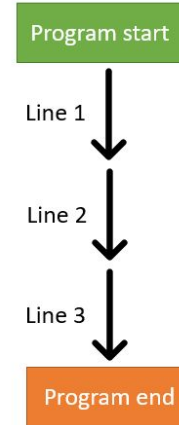
- ❖ Requests are used to send your information to the server, asking it to perform a set of tasks.
- ❖ Examples of requests of CRUD (Create, Read, Update, and Delete) operations:
 - **GET** - Request or **read** data that **exists** within the database.
 - **POST** - We may want to **create** new (add new) items to the database.
 - **PUT** - This method allows you to **update existing data** on the server, such as changing the image of an item or updating the entire item's details.
 - **PATCH** - Unlike PUT, which updates the entire resource, PATCH is used to make **partial updates** to **existing data**.
 - **DELETE** - As the name suggests, this **deletes** an item from a resource.



Asynchronous VS Synchronous

❖ Synchronous processing:

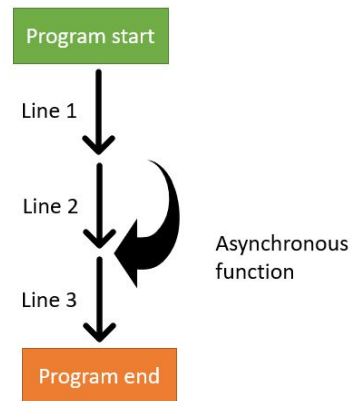
- Code runs in order, line by line.
- Each line of code waits for the one before it to finish before running.
- The line below the current line of code won't run until the current line of code is finished running.
- This approach is safer because the flow of control moves predictably through the code.
- Each line runs one after the other from start to finish.



Asynchronous VS Synchronous

❖ Asynchronous processing:

- Asynchronous processing is slightly more complex as it involves multiple sets of code being run at the same time.
- If this process is not used properly, it can lead to more errors than synchronous programming because different parts of the code might run at the same time and also depend on each other, causing issues.
- The general rule of thumb is to only run an asynchronous function if no other code is dependent on it.
- Promises are asynchronous and run separately from your normal code.



PROMISES VS ASYNC/AWAIT

- ❖ The key differences between the two concepts.

	Promise	Async
Scope	Only the original promise itself is asynchronous.	The entire function itself is asynchronous.
Logic	Any synchronous work can be performed inside the same callback that defines the promise.	Any synchronous work can be performed inside the async function along with asynchronous operations.
Error Handling	Promises can handle errors using a combination of <code>.then()</code> , <code>.catch()</code> , and <code>.finally()</code> .	Error handling in async functions is done using a combination of <code>try/catch/finally</code> .

What is the purpose of `fetch()` in JavaScript?

- A. It sends an HTTP request to a server and returns a Promise that resolves to the response.
- B. It fetches data from the local storage and returns a string.
- C. It sends an HTTP request to a server and automatically updates the DOM with the response.
- D. It fetches the latest version of a webpage from the server without reloading the page.



In JavaScript, why is it important to wait for a Promise before continuing?

- A. To speed up the execution of the program by forcing it to run synchronously.
- B. To avoid blocking the event loop, allowing multiple asynchronous operations to be processed in parallel.
- C. There is no need to wait.
- D. To ensure that asynchronous operations complete before the program continues to the next task, preventing potential errors.

Questions and Answers



Thank you for attending



CoGrammar



Department
for Education