Welcome to the CoGrammar Tutorial: Node.js and React.js

The session will start shortly...

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

CoGrammar

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**

CoGrammar

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

CoGrammar

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

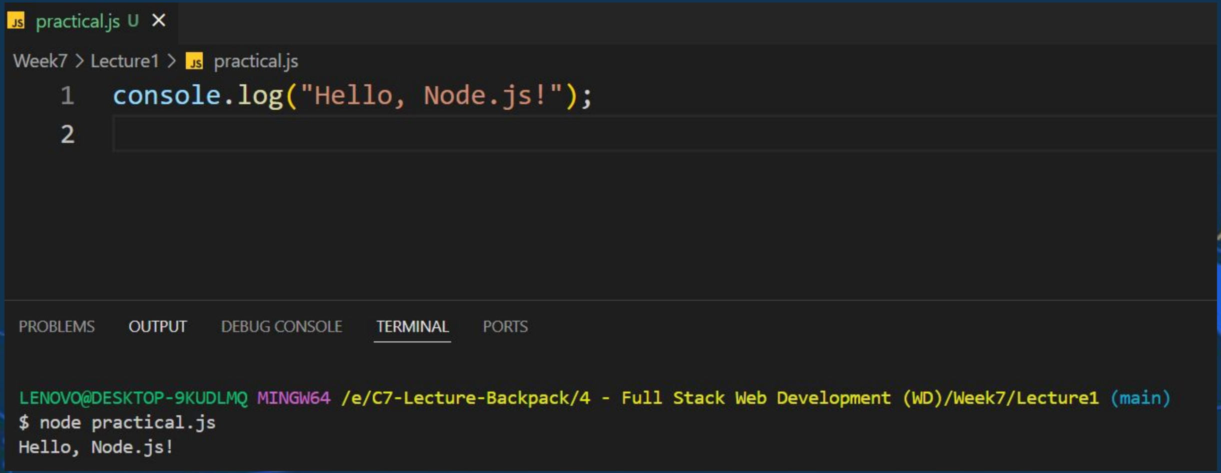
Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

What is Node.js?

❖ Node.js is a runtime environment that allows you to run JavaScript code on the server-side.

❖ It uses an event-driven, non-blocking I/O model, making it efficient for handling asynchronous operations.



```javascript
console.log("Hello, Node.js!");
```

```
LENOVO@DESKTOP-9KUDLMQ MINGW64 /e/C7-Lecture-Backpack/4 - Full Stack Web Development (WD)/Week7/Lecture1 (main)
$ node practical.js
Hello, Node.js!
```

CoGrammar

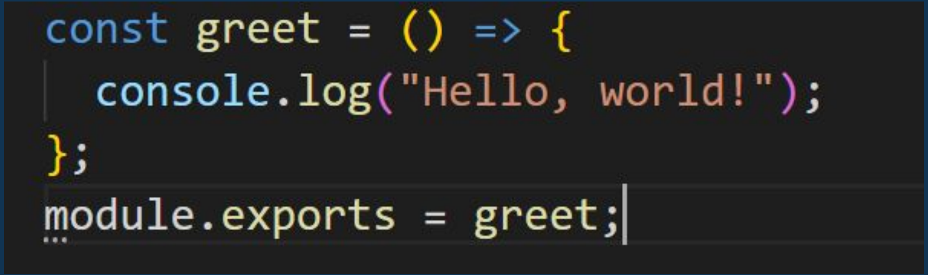
What are Modules?

- ❖ Modules in Node.js are encapsulated units of functionality that can be reused throughout your application.
- ❖ They promote code organization, maintainability, and reusability.
- ❖ Node.js implements the CommonJS module system, allowing modules to be defined using require() and exported using module.exports.

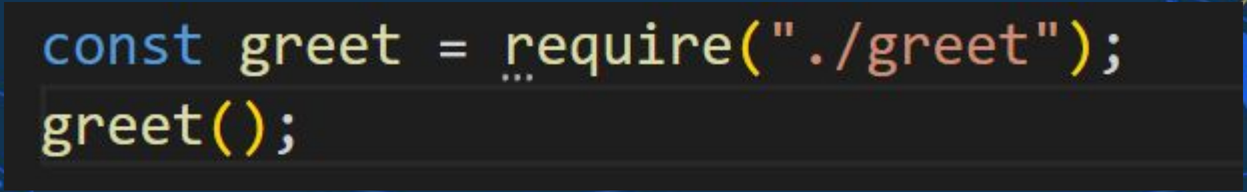
CoGrammar

Creating and Using Modules

- ❖ Modules in Node.js are encapsulated units of functionality that can be reused throughout your application.
- ❖ They promote code organization, maintainability, and reusability.



```
const greet = () => {
  console.log("Hello, world!");
};
module.exports = greet;
```



```
const greet = require("./greet");
greet();
```

CoGrammar

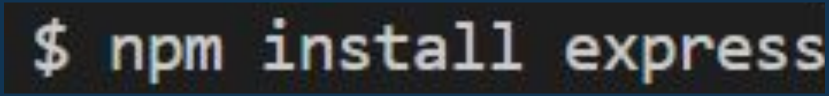
NPM (Node Package Manager)

- ❖ NPM is the default package manager for Node.js, used for installing, managing, and sharing packages of JavaScript code.
- ❖ It provides access to a vast repository of open-source packages and tools for Node.js development.

CoGrammar

Managing Dependencies with NPM

- Define project dependencies in the package.json file.
- Use npm install to install dependencies listed in package.json.



```
$ npm install express
```

CoGrammar

Creating a package.json File

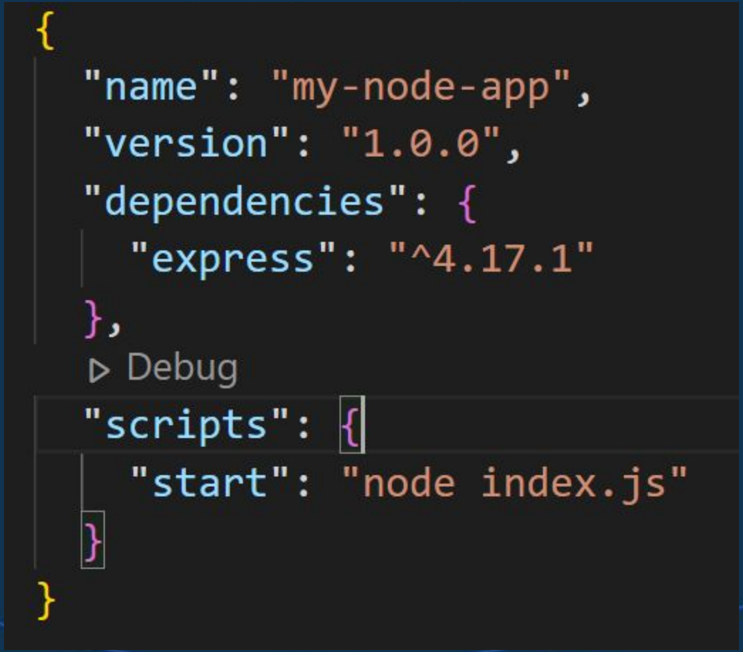
- Use **npm init** to generate a package.json file interactively or with default values.
- **package.json** serves as the manifest for your project, documenting project metadata, dependencies, and scripts.

CoGrammar

Understanding package.json Structure

- **name:** The name of the project.
- **version:** The version of the project.
- **dependencies:** List of project dependencies and their version specifications.
- **scripts:** Custom scripts for tasks like testing, building, and deployment.

Understanding package.json Structure

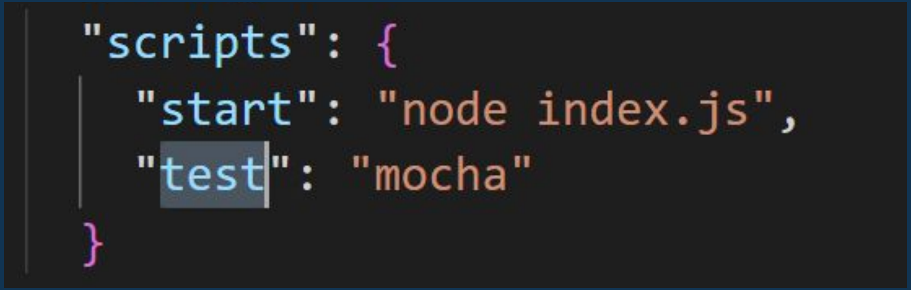


```
{
  "name": "my-node-app",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.17.1"
  },
  ▷ Debug
  "scripts": {
    "start": "node index.js"
  }
}
```

CoGrammar

Managing Scripts in package.json

- Use the **scripts** field in **package.json** to define custom scripts.
- Scripts can be executed using **npm run <script-name>**.



```
"scripts": {
  "start": "node index.js",
  "test": "mocha"
}
```

CoGrammar

React Introduction

- ❖ ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components.
- ❖ It is an open-source, component-based front end library responsible only for the view layer of the application.
- ❖ A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of code.
- ❖ The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks.

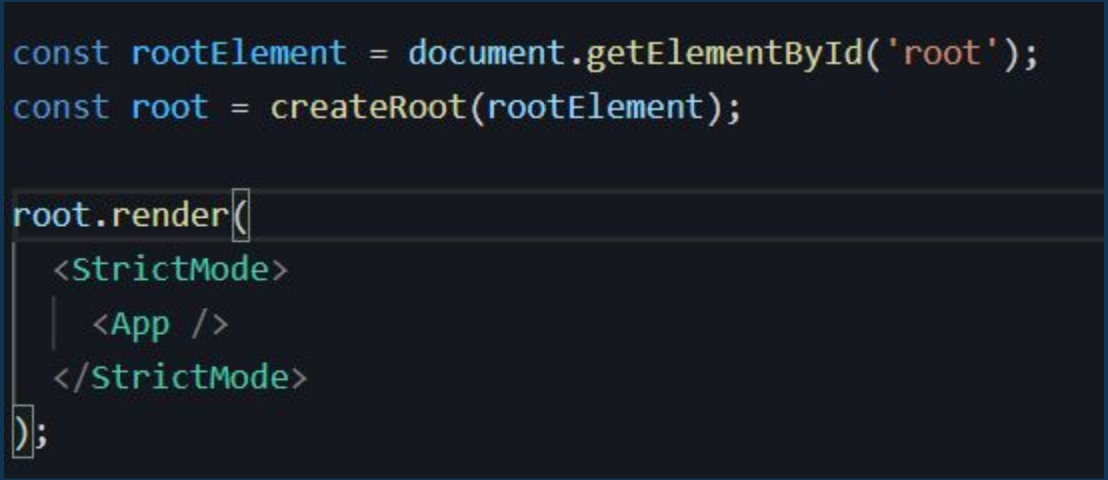
CoGrammar

Rendering in React

- ❖ React renders HTML to the web page by using a function called **createRoot()** and its method **render()**.
- ❖ The **createRoot()** function takes one argument, an HTML element. The purpose of the function is to define the HTML element where a React component should be displayed.
- ❖ The **render()** method is then called to define the React component that should be rendered.

CoGrammar

Rendering in React



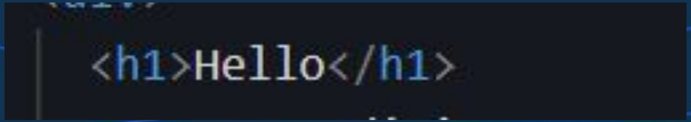
```
const rootElement = document.getElementById('root');
const root = createRoot(rootElement);

root.render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

CoGrammar

React JSX

- JSX(JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML.
- In other words, JSX is an HTML-like syntax used by React that extends ECMAScript so that HTML-like syntax can co-exist with JavaScript/React code.
- JSX allows you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code.

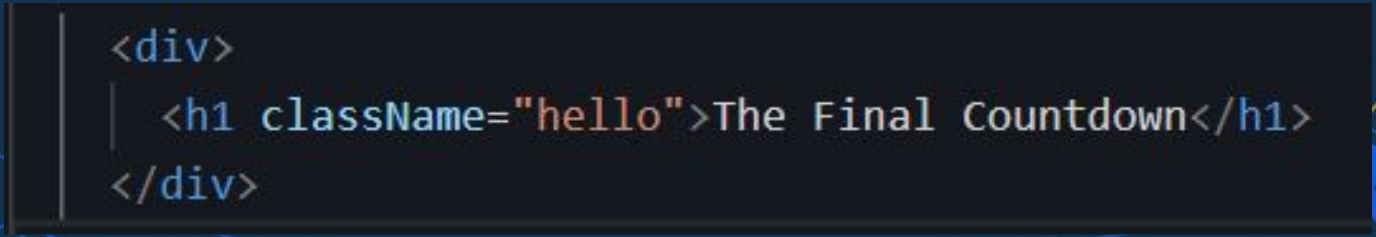


```
<h1>Hello</h1>
```

CoGrammar

JSX Attributes

- ❖ JSX use attributes with the HTML elements same as regular HTML.
- ❖ JSX uses camelcase naming convention for attributes rather than standard naming convention of HTML such as a class in HTML becomes className in JSX because the class is the reserved keyword in JavaScript.

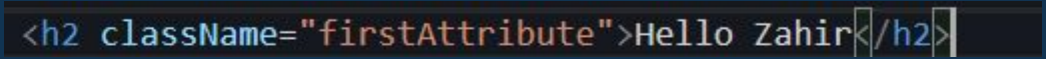


```
<div>
  <h1 className="hello">The Final Countdown</h1>
</div>
```

CoGrammar

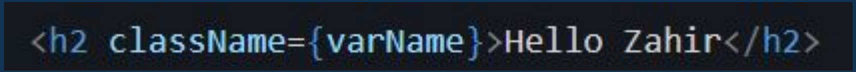
JSX Attributes

- ❖ In JSX, we can specify attribute values in two ways:
  - ➢ As String Literals: We can specify the values of attributes in double quotes.



```
<h2 className="firstAttribute">Hello Zahir</h2>
```

  - ➢ As Expressions: We can specify the values of attributes as expressions using curly braces **{}**

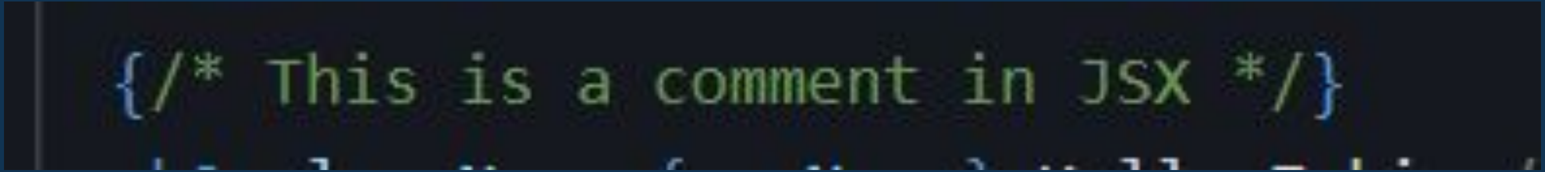


```
<h2 className={varName}>Hello Zahir</h2>
```

CoGrammar

JSX Comments

❖ JSX allows us to use comments that begin with /* and ends with */ and wrapping them in curly braces {}.

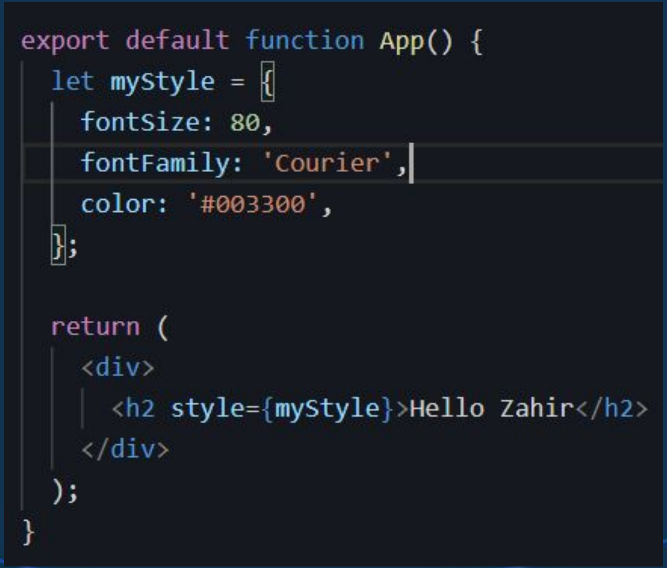


```
{/* This is a comment in JSX */}
```

CoGrammar

JSX Styling

❖ To set inline styles, you need to use camelCase syntax.



```
export default function App() {
  let myStyle = {
    fontSize: 80,
    fontFamily: 'Courier',
    color: '#003300',
  };

  return (
    <div>
      <h2 style={myStyle}>Hello Zahir</h2>
    </div>
  );
}
```

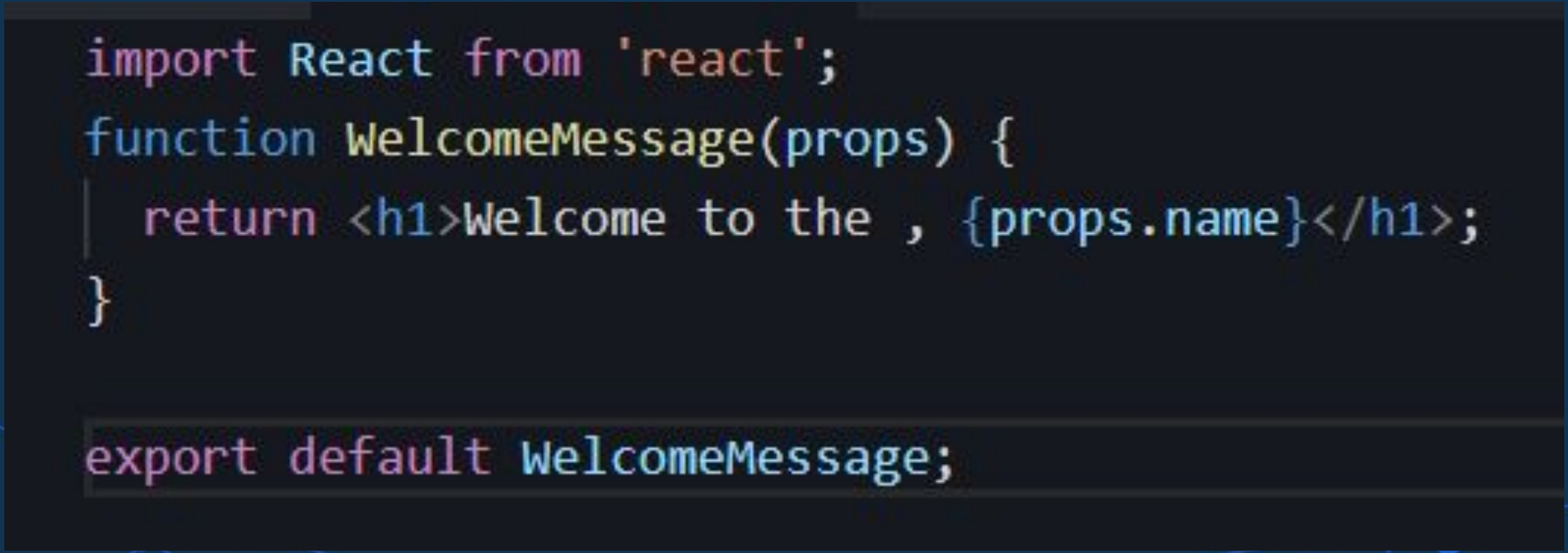
CoGrammar

React Components

- A Component is considered as the core building blocks of a React application.
- It makes the task of building UIs much easier.
- Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.
- All React components have their own structure, methods as well as APIs. They can be reusable as per your need.

CoGrammar

Functional Components



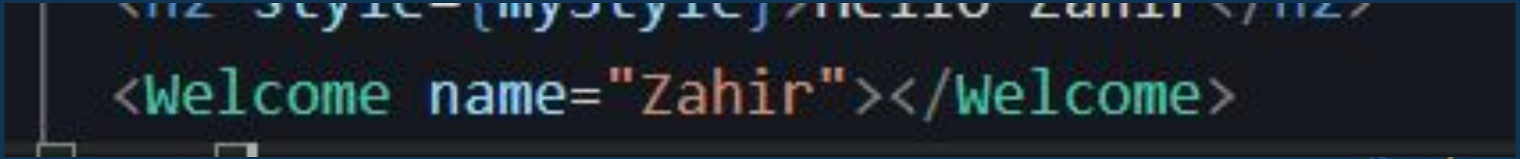
```
import React from 'react';
function WelcomeMessage(props) {
  return <h1>Welcome to the , {props.name}</h1>;
}

export default WelcomeMessage;
```

CoGrammar

Props

- ❖ Components can be passed props, which stands for properties.
- ❖ Props are like function arguments, and you send them into the component as attributes.



```
<Welcome name="Zahir"></Welcome>
```

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar