CoGrammar

Welcome to this session:

Node.js

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

CoGrammar HyperionDev

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.

Learning Outcomes

By the end of this lecture, students should be able to:

- Identify the concept of modules in Node.js and learn how to create, import, and use modules effectively.
- Gain familiarity with NPM (Node Package Manager) and its role in managing dependencies, versioning, and scripts.
- Develop basic Node.js modules.

Do you know how to make use of code from another JavaScript file?

A. Yes, I know how to import/export code between JavaScript files.
B. I’ve heard of it, but I’m not sure how it works.
C. No, I don’t know how to do that.

Have you ever built a Node.js application from scratch?

A. Yes, I’ve built multiple applications.
B. Yes, I’ve built one or two applications.
C. No, but I have contributed to a Node.js project.
D. No, I haven’t built an application yet.
E. I’m not sure.

CoGrammar

Lecture Overview

→ Introduction to Node.js?
→ Managing Node packages
→ Understanding modules
→ Importing/Exporting modules
→ Creating your own package
→ Set up your own scripts

What is Node.js?

- Node.js is a **runtime environment** that allows you to run JavaScript on servers, making it possible to write server-side code.
- It uses an event-driven, non-blocking model, allowing it to handle many tasks simultaneously, which is ideal for apps that need to be fast and support many users.
- Node.js includes built-in tools (modules) to simplify tasks. It includes a few built-in modules, such as **fs** (file system), **http** (HTTP server), and **path** (file and directory paths), which can be used without additional installation.

Installing Node.js?

- To get started, you'll need to refer to the **“Additional Reading – Installation, Sharing, and Collaboration”** guide for detailed instructions on installing Node.js (pages 18 to 21).
- Alternatively, you can obtain it from https://nodejs.org/

CoGrammar

Managing Node packages

- ❖ **Node package manager** (NPM).
  - ➢ NPM is a tool that helps you manage packages, which are collections of reusable code that can add specific functionality to your Node.js applications.
  - ➢ A **package** can contain one or more **modules**, and it typically includes a **package.json** file that provides metadata about the package, such as its name, version, and dependencies.
  - ➢ NPM allows you to easily install, update, and manage these packages, making it simple to incorporate external libraries and tools into your projects.

CoGrammar

Understanding modules

- ❖ A **module** is a reusable piece of code that has been organised into a separate file. Modules can help break applications into smaller, more manageable parts, making them easier to develop and maintain.
- ❖ Key benefits of using modules:
  - ➢ Encapsulation.
  - ➢ Reusability.
  - ➢ Maintainability.

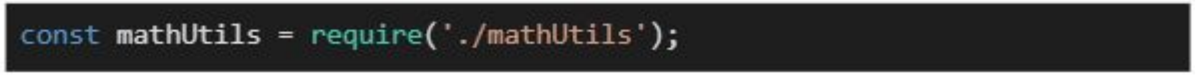
CoGrammar

Core Modules vs. User-defined Modules

- ❖ Node.js provides several core modules like http, fs, and path, which can be used without installation.
- ❖ User-defined modules are created by developers to encapsulate specific functionality.

Importing modules CommonJS

- ❖ Common JavaScript syntax (traditional way):
  - ➢ This method uses the **require()** function to import modules. For example, to import a module named mathUtils.js, you would write:

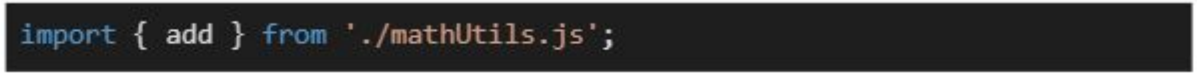


```
const mathUtils = require('./mathUtils');
```

Importing modules ES6

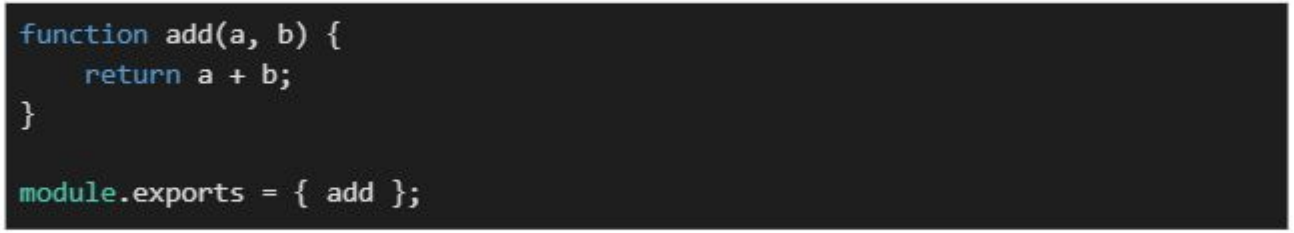
❖ ES6 syntax:

➢ ES6, also known as ECMAScript 2015, introduced a more modern approach to importing modules. Using ES6 syntax, it's possible for us to use the **import** statement to bring in modules. For instance, to import the same mathUtils.js module, you would write:



```
import { add } from './mathUtils.js';
```

Exporting from a module



```
function add(a, b) {
    return a + b;
}

module.exports = { add };
```

mathUtils.js

CoGrammar

Creating your own package

1. Create your package directory
2. Navigate into your package directory
3. Initialise a new Node package
   - ➢ npm init

CoGrammar

Creating your own package
npm init

❖ When you run **npm init**, you're creating a new Node package.

❖ This command sets up a **package.json** file in your module's directory.

❖ The package.json file is **essential** because it contains important information about your module, such as its name, version, description, entry point (the main file), scripts, dependencies, and more.

CoGrammar

Adding external packages with NPM

- Navigate into your package directory
  - Make sure that you are in the directory that contains your package package.json
- Use **npm install** to install external packages.
  - i.e. npm install lodash
- A **node_modules** directory will be created within the main directory of your package. It contains all the modules for any additional packages that you installed.

CoGrammar

Installed packages CommonJS vs ES6

- ❖ **CommonJS** is the most commonly used module system in Node.js. It uses the require() method to import modules dynamically at runtime.
  - ➢ This means that modules are loaded as the code runs, allowing for flexibility in determining which module to import. You can even use it within conditionals or loops.
- ❖ **ES6** is the module system supported by modern web browsers. It uses the import statement for static imports, meaning that the modules are loaded at compile-time, before the code runs. Because of this static nature, you cannot use variables or place import statements inside conditionals or loops.
  - ➢ If you make use of ES6, add the following key-value pair to your package.json file **"type": "module"**

CoGrammar

Set up your own scripts in Node.js

- ❖ Sometimes, navigating file names can be confusing, especially when needing to input different arguments. NPM provides a handy tool to navigate this: **scripting**.
- ❖ Use the **scripts** object in **package.json** to define custom scripts.
- ❖ Scripts can be executed using **npm run *script-name***.

CoGrammar

What is the primary purpose of running npm init in a Node.js project?

A. To create a new Node.js project and generate a package.json file.
B. To install all dependencies listed in a package.json file.
C. To start a server and run the project in production mode.
D. To create a new JavaScript file for the project.

What is the purpose of the package.json file in a Node.js project?

A. To store metadata about the project, like its name, version, and description.
B. To list all the dependencies (packages) the project needs to run.
C. To define project scripts (e.g., start, test, build) that can be run via npm.
D. All of the above.

Let's take a break

CoGrammar

Questions and Answers

CoGrammar

Thank you for attending

CoGrammar

SKILLS FOR LIFE SKILLS BOOTCAMPS

Department for Education