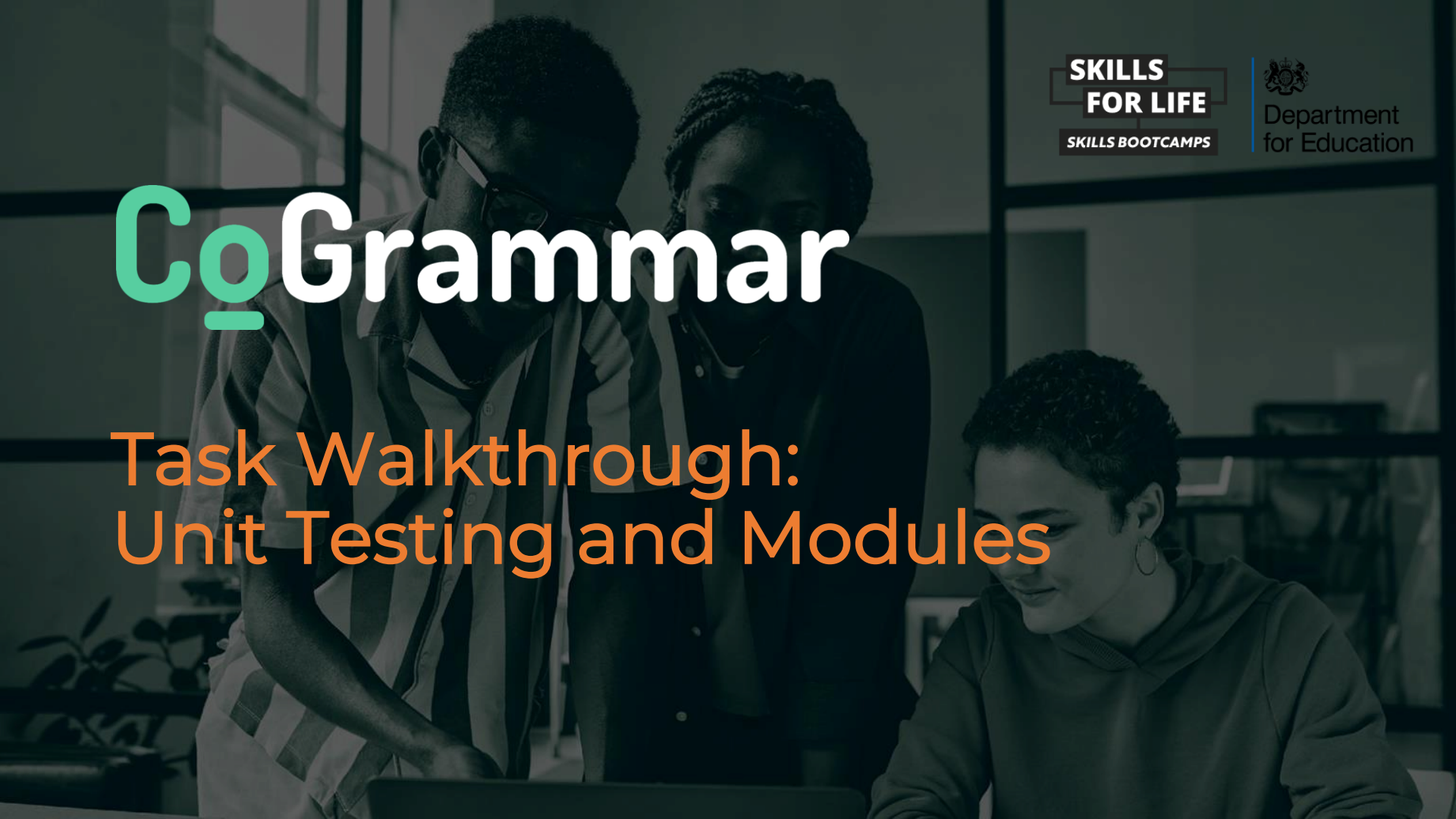
Welcome to this CoGrammar

Task Walkthrough: Task 17 and 18

The session will start shortly...

Questions? Drop them in the chat.



Task Walkthrough: Unit Testing and Modules

Software Engineering 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** throughout this session, should you wish to ask any follow-up questions.
- 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

Software Engineering 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

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

Learning Outcomes

- Identify and explain the purpose of unit testing and its importance in software development.
- Apply unit testing tools (unittest) to write and execute test cases for different modules, verifying the functionality of individual methods and class interactions.
- Describe the structure and role of Object-Oriented Programming (OOP) modules in building modular and reusable code.

Task Walkthrough: Unit Testing

CoGrammar

What is unit testing?

- Software testing method where individual units or components of a software application are tested in isolation to ensure they work as intended.
- The goal is to verify that each unit of the software performs as designed and that all components are working together correctly.
- Unit tests help developers catch bugs early in the development process, when they are easier and less expensive to fix.

Advantages of Unit Testing

- Catch errors early
- Improve code quality
- Refactor with confidence
- Document code behaviour
- Facilitate collaboration

Arrange, Act, Assert

- The AAA pattern is a common pattern used in unit testing to structure test cases. It stands for Arrange, Act, Assert.
  - Arrange: Set up any necessary preconditions or test data for the unit being tested.
  - Act: Invoke the method or code being tested.
  - Assert: Verify that the expected behaviour occurred.

Arrange, Act, Assert

Let's have a look at an example of how to write a unit test in Python using the AAA pattern.

- Consider a simple function that adds two numbers:

```
def add_numbers(a, b):
    return a + b
```

Arrange, Act, Assert

- To test this function, we would create a new function called test_add_numbers (note that the name must start with test_ for the Python test runner to find it).

```
def test_add_numbers(self):
    # Arrange
    a = 2
    b = 3

    # Act
    result = add_numbers(a, b)

    # Assert
    self.assertEqual(result, 5)
```

We've set up the test data (Arrange) by creating two variables a and b with the values 2 and 3.

We then invoke the function being tested (Act) and store the result in a variable called result.

Finally, we assert that the result is equal to the expected value of 5 (Assert).

Auto-graded task

In this task, you are going to write unit tests for one of your previous practical tasks. Follow the instructions below.

- Select one of the recent practical programming tasks you completed. This will be the focus of your unit tests. Consider the selected task your "implementation".
- Identify at least three different scenarios (use cases) in which your implementation will be used. These use cases will guide the creation of your unit tests to ensure your implementation works correctly in different situations.
- Create tests for each use case. Focus on testing your implementation directly without external dependencies.
    - Single file: If your code is in one Python file, add the unit tests at the end of that same file.
    - Multiple files: If your code is spread across multiple files, create a separate Python file for the tests and import the files you need to test.
- If your code is difficult to test, then it means you likely need to refactor it so that it's easier to test. This is a normal and necessary part of writing good code and precise tests.

Task Walkthrough: OOP - Modules

CoGrammar

What are Modules?

- Modules in Python are files containing Python code that can define variables, functions, classes, or other Python constructs.
- The primary purpose of modules is to organise code into reusable and manageable units, facilitating better code organisation, maintenance, and reuse.

Practical task

In this practical task, you are going to implement and test the task manager application you designed in the **Software Design Task**.

Follow the steps below to complete the task. You do not have to provide written answers to the questions, as they are meant to guide your thinking.

1. Set up a virtual environment for your project by following these steps:
    - Determine the steps that are required to create and activate a virtual environment for your project.
    - Select a tool for managing the virtual environment (such as **venv**).
2. Integrate PEP 8 linting into your project:
    - Decide how you will integrate PEP 8 linting.
    - Specify any particular PEP 8 rules or configurations you want to enforce or ignore through the use of configuration files such as **.flake8**.
3. PEP 8 compliance:
    - Ensure that your code adheres to PEP 8 standards.
    - Make any necessary adjustments to achieve compliance.

4. Implement your task manager application that follows your design:
    - Develop the necessary classes based on your design. Feel free to update the designs for the application to suit your implementation needs if you have discovered design improvements during your implementation phase.
    - Implement a file-based data access layer to read from and write to files for storing data. This ensures that the application's data is safely stored and easily accessible, even after the application is closed.
    - Remember to split your code into multiple modules. A common practice is to have one class or set of related functions or constants per Python module.

5. Write unit tests:
    - Identify **at least four different use cases** based on your designs for the task management application. You may need to redesign the application to include at least four use cases.
    - Create unit tests for each identified use case. These tests should verify that your code functions as expected under different scenarios:
        - Write unit tests that focus on the core logic and data structures of your application.
        - Avoid testing parts of the application that depend on external systems, such as text files. This can simplify your tests while making them more reliable.

6. Use a tool like `pip freeze` to generate a **requirements.txt** file.
    - Depending on your project's dependencies and whether you've installed any additional packages via `pip`, the **requirements.txt** file may be empty if no external packages are required for your project.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar