Welcome to this CoGrammar tutorial: Unit Testing and Modules

The session will start shortly...

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

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

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

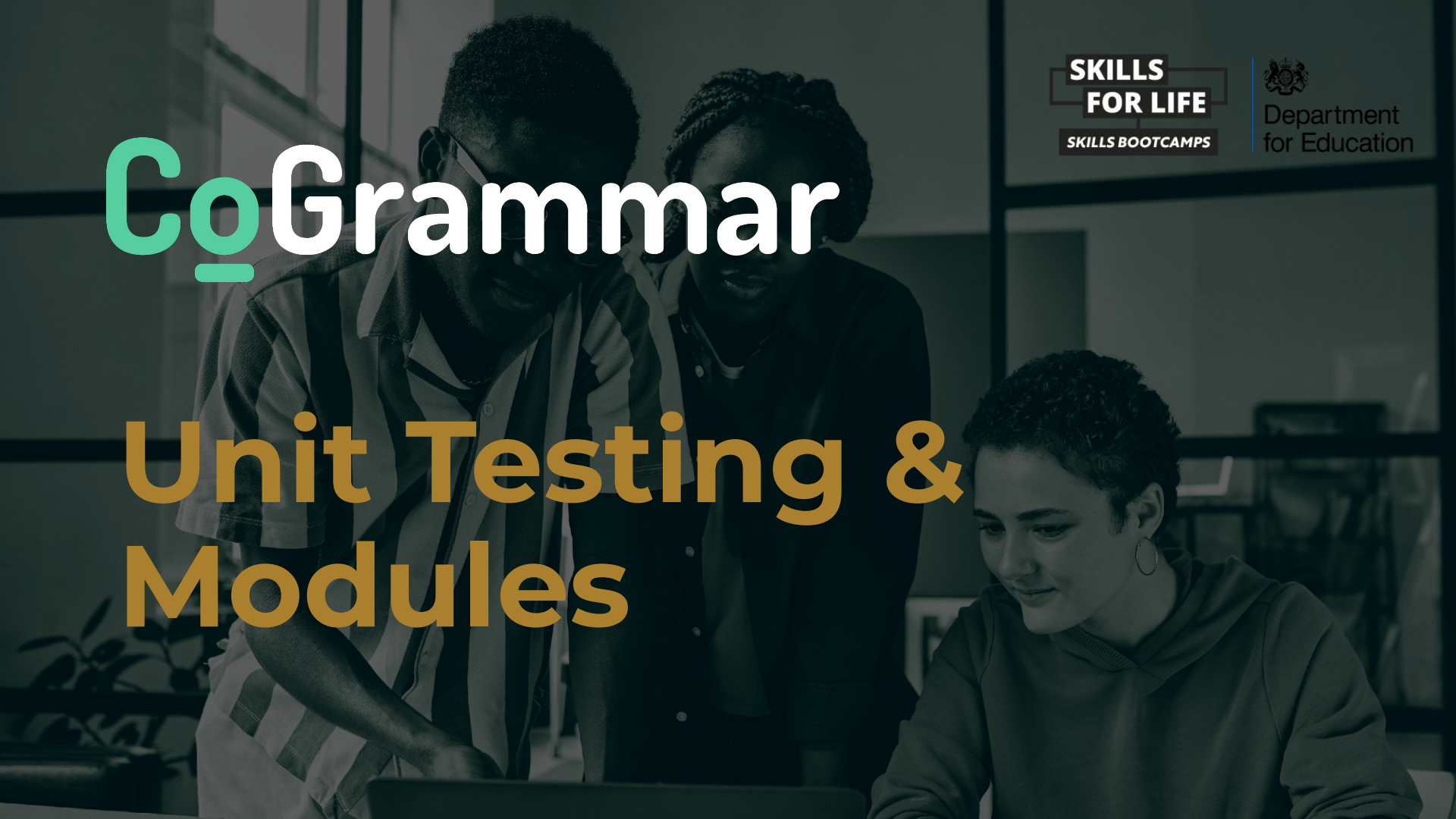
Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

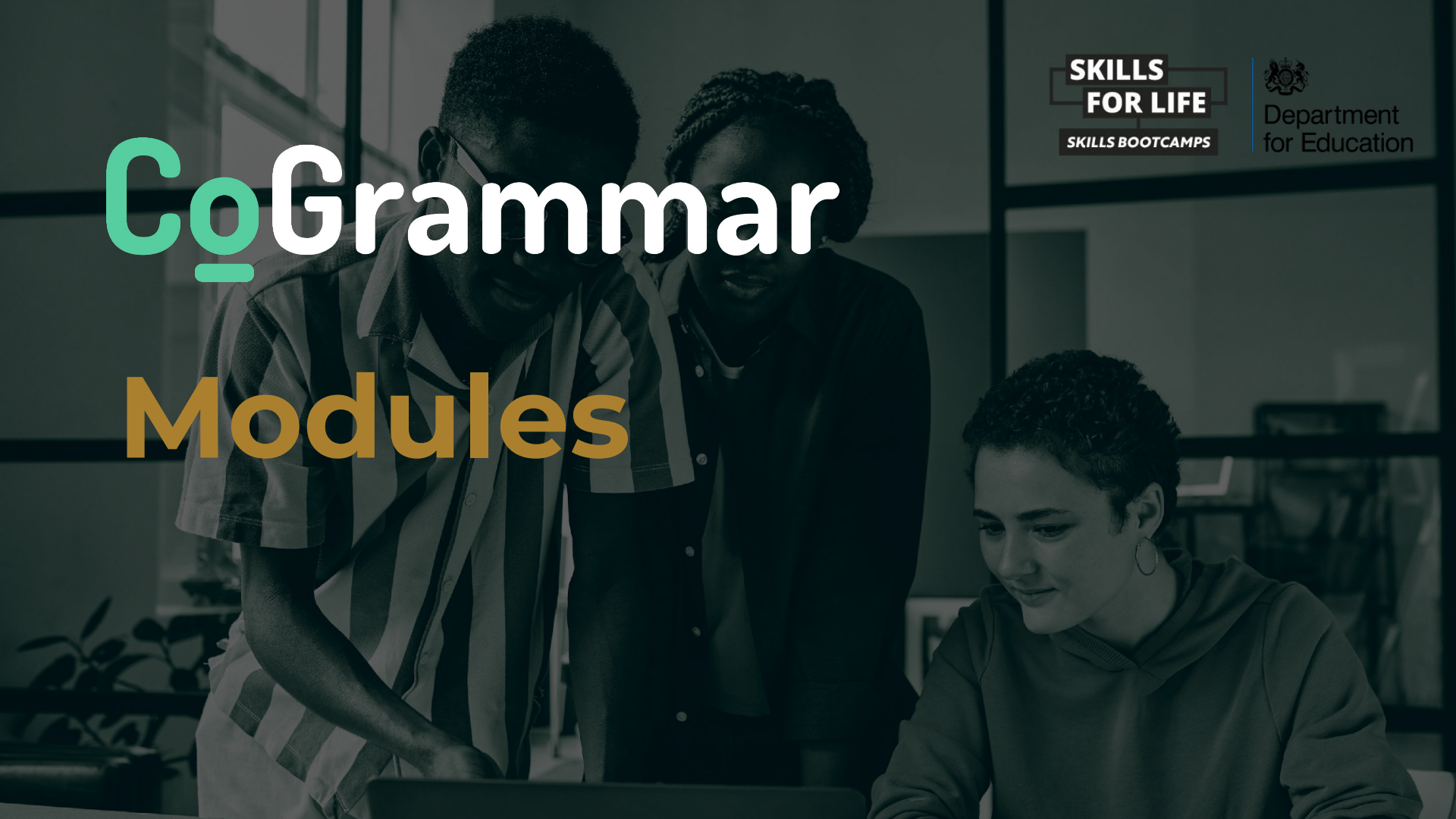

SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Unit Testing & Modules

Why Modular Programming Matters

- **Better Code Organization**: Divides large programs into smaller, manageable parts.
- **Reusability**: Write once, use multiple times.
- **Maintainability**: Easier to debug and update.
- **Collaboration**: Different teams can work on different modules.

How Modules & Unit Testing Are Connected

- **Modular design facilitates testing**: Each module can be tested independently.
- **Unit tests validate module functionality**: Ensures correct behavior of each part.
- **Encapsulation helps in isolation**: Testing one module doesn't interfere with others.


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Modules

Modules

CoGrammar

What are Modules, Packages, and Libraries?

- **Script**: A standalone Python file meant to be executed directly.
- **Module**: A single Python file (.py) containing functions and classes.
- **Package**: A directory with an __init__.py file, containing multiple modules.
- **Library**: A collection of packages/modules, often installed via pip.
- **Example**: math (module), numpy (library with multiple modules).

Creating & Using Modules

- **Steps**:
  - Create a Python file `my_module.py`.
  - Define functions/classes in it.
  - Import it into another script using `import my_module`.
- **Example**:

```
# my_module.py
def greet(name):
    return f"Hello, {name}!"

# main.py
import my_module
print(my_module.greet("Alice"
```

Working with Virtual Environments (venv)

- Why use **venv**?:
  - Isolates dependencies for different projects.
  - Prevents conflicts between package versions.
- Creating a Virtual Environment:
  - python -m venv myenv
- Activating the Environment:
  - **Windows**: myenv\Scripts\activate
  - **macOS/Linux**: source myenv/bin/activate
- Deactivating:
  - `deactivate`

Managing Dependencies with `pip` & `requirements.txt`

- Installing packages: `pip install package_name`
- Freezing dependencies: `pip freeze > requirements.txt`
- Installing from **`requirements.txt`**: `pip install -r requirements.txt`

Code Quality: Linting & Code Style (PEP 8 & PEP 484)

- **PEP 8:** Python's official style guide.
- **PEP 484:** Introduces type hinting.
- **Linting tools:**
  - `flake8`: General code style checks.
  - `pylint`: Static code analysis.
- **Example:**

```
def add(num1: int, num2: int) -> int:
    """Return the sum of two integers."""
    return num1 + num2
```


SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education
CoGrammar
Unit Testing

What is Unit Testing?

- What is Unit Testing?
  - Testing individual components of code (functions, methods, classes).
  - Ensures correctness and prevents bugs.
- Why Unit Test?
  - Helps catch errors early.
  - Improves maintainability.
  - Supports refactoring with confidence.

AAA (Arrange-Act-Assert) Pattern

- Test Structure:
  - Arrange: Set up test data and environment.
  - Act: Execute the function being tested.
  - Assert: Verify the output matches expectations.

```
class TestAddFunction(unittest.TestCase):

  def test_positive_numbers(self):
    # Arrange (Set up the necessary data)
    x = 2
    y = 3

    # Act (Call the function under test)
    result = add(x, y)

    # Assert (Check if the result is what we expect)
    self.assertEqual(result, 5)  # Check if the result is 5
```

FIRST Principles in Testing

- **Fast:** Tests should run quickly.
- **Independent:** Tests should not depend on each other.
- **Repeatable:** Tests should yield the same results every time.
- **Self-validating:** Each test should have clear pass/fail criteria.
- **Thorough:** Cover various edge cases and normal cases.

Resolving Failing Tests

- **Common causes of test failures:**
  - Incorrect assertions.
  - Unexpected function behavior.
  - Environment or dependency issues.
- **Debugging failing tests:**
  - Print output to debug errors.
  - Use breakpoints for deeper inspection.
  - Refactor code if needed.

Writing Unit Tests for Modularized Code

- Test individual functions within modules.
- Mock dependencies when needed.
- Example:

```
def test_total_price(self):
    item = Item("Laptop", 1000, 2)
    self.assertEqual(item.total_price(), 2000)
```

Conclusion and Recap

CoGrammar

Writing Unit Tests for Modularized Code

- Use **modules** to structure code effectively.
- **Virtual environments** help manage dependencies.
- **Linting** improves code quality.
- **Unit tests** ensure reliability.
- Follow **AAA pattern & FIRST principles** for better testing.

Questions and Answers

CoGrammar

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar