Welcome to the CoGrammar 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

Enhancing Accessibility: Activate Browser Captions

Why Enable Browser Captions?

- Captions provide **real-time text for spoken content**, ensuring inclusivity.
- Ideal for individuals in noisy or quiet environments or for those with **hearing impairments**.

How to Activate Captions:

1. YouTube or Video Players:
   - Look for the CC (Closed Captions) icon and click to enable.
2. Browser Settings:
   - Google Chrome: Go to *Settings > Accessibility > Live Captions* and toggle ON.
   - Edge: Enable captions in *Settings > Accessibility*.

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 Progression Overview

✓ Criterion 1 - Initial Requirements

Specific achievements within the first two weeks of the program.

To meet this criterion, students need to, by no later than 01 December 2024 (C11) or 22 December 2024 (C12):

- **Guided Learning Hours** (GLH): Attend a minimum of 7-8 GLH per week (lectures, workshops, or mentor calls) for a total minimum of 15 GLH.
- **Task Completion:** Successfully complete the first 4 of the assigned tasks.

✓ Criterion 2 - Mid-Course Progress

Progress through the successful completion of tasks within the first half of the program.

To meet this criterion, students should, by no later than 12 January 2025 (C11) or 02 February 2025 (C12):

- **Guided Learning Hours** (GLH): Complete at least 60 GLH.
- **Task Completion :** Successfully complete the first 13 of the assigned tasks.

CoGrammar

Skills Bootcamp Progression Overview

✓ Criterion 3 – End-Course Progress

Showcasing students' progress nearing the completion of the course.

To meet this criterion, students should:

- **Guided Learning Hours** (GLH): Complete the total minimum required GLH, by the support end date.
- **Task Completion :** Complete all mandatory tasks, including any necessary resubmissions, by the end of the bootcamp, 09 March 2025 (C11) or 30 March 2025 (C12).

✓ Criterion 4 - Employability

Demonstrating progress to find employment.

To meet this criterion, students should:

- **Record an Interview Invite:** Students are required to record proof of invitation to an interview by 30 March 2025 (C11) or 04 May 2025 (C12).
  - **South Holland Students** are required to proof and interview by 17 March 2025.
- **Record a Final Job Outcome :** Within 12 weeks post-graduation, students are required to record a job outcome.

CoGrammar

Stay Safe Series:

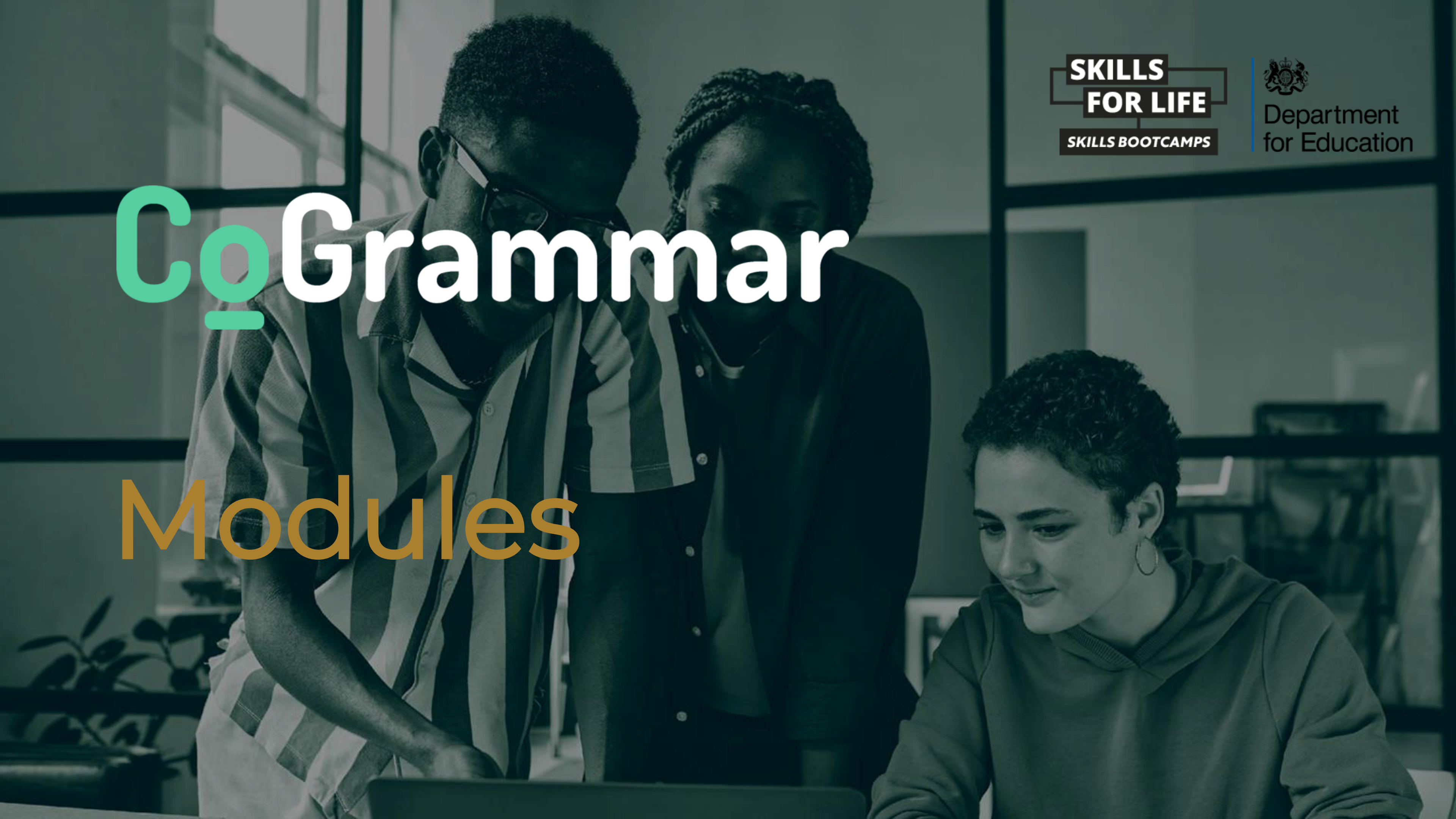
Mastering Online Safety One Week or Step at a Time

While the digital world can be a wonderful place to make education and learning accessible to all, it is unfortunately also a space where harmful threats like online radicalisation, extremist propaganda, phishing scams, online blackmail and hackers can flourish.

As a component of this BootCamp the *Stay Safe Series* is designed to guide you through essential measures in order to protect yourself & your community from online dangers, whether they target your privacy, personal information or even attempt to manipulate your beliefs.

Shop Smart: Staying Safe with Online Purchases

- Ensure you have a secure connection.
- Use familiar merchants.
- Use secure passwords.
- Don’t make purchases on public connections.
- Make sure the payment method is secure.
- Look at online reviews.



SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Modules

Polls

CoGrammar

Poll

1. **What is the purpose of the __init__.py file in a Python module?**

   A. It makes a directory a package.

   B. It prevents a module from being imported.

   C. It automatically executes when Python starts.

   D. It stores environment variables.

Poll

2. **Which Python module is used for working with dates and times?**

   A. calendar

   B. datetime

   C. time

   D. dateutil

Learning Outcomes

- Define the purpose and importance of Python **modules, requirements files and virtual environments**.
- Differentiate between **scripts, modules, packages, and libraries** in Python
- Import and use modules from the **Python Standard Library**
- Create **custom** Python **modules and import and use them into scripts**
- Apply object-oriented principles to modularisation by defining **classes, functions and other variables within modules**
- Implement **Python code style guidelines (PEP 8)**, type hinting **(PEP 484)**, and linting tools

Modules

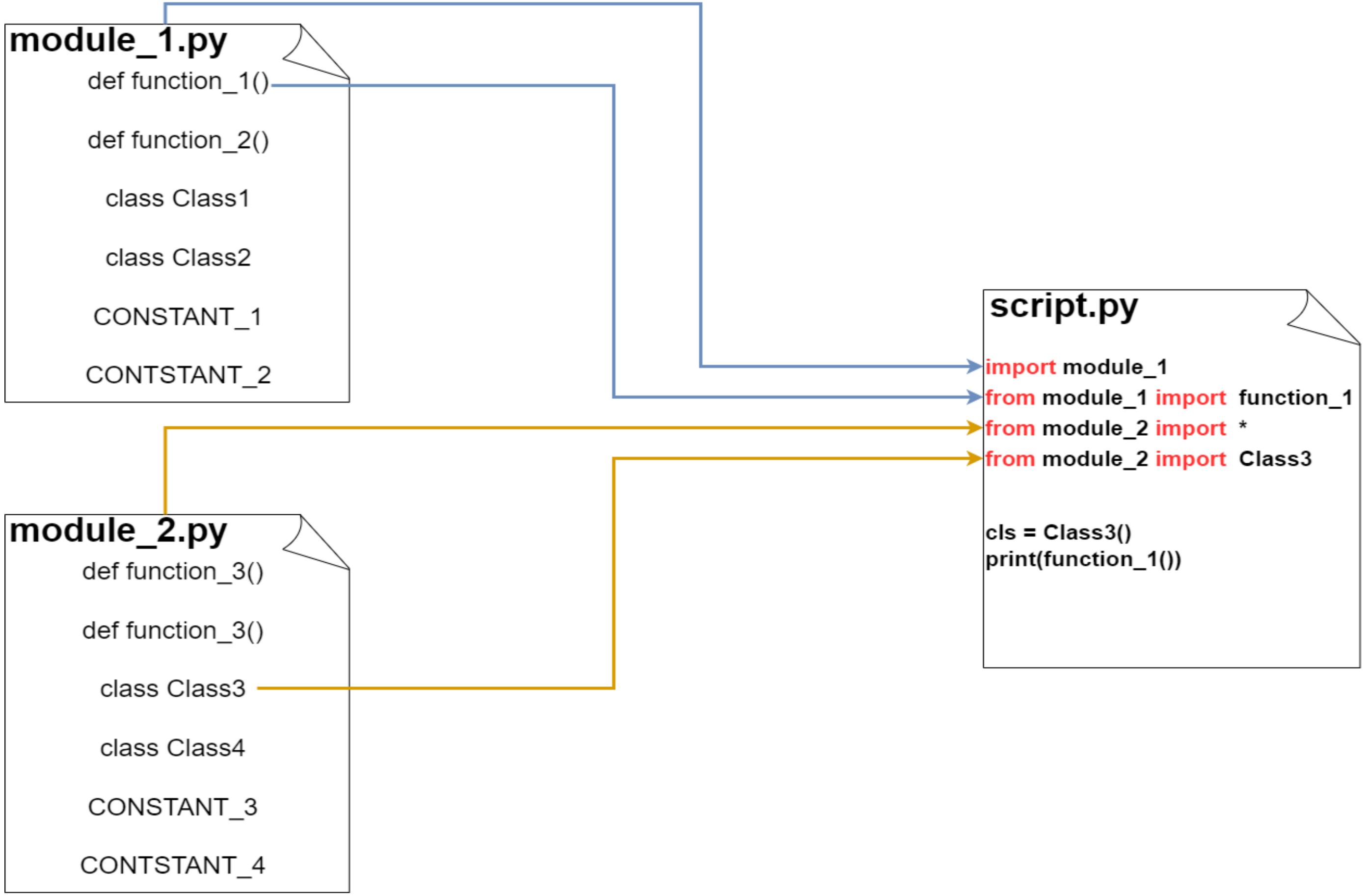
CoGrammar

Analogy

Just as a toolbox organises various tools into separate compartments, modules in programming organise related **functions**, **classes**, and **variables** into **separate "compartments"** or "**drawers**". Each module serves a specific purpose, like a drawer containing tools for a particular task.

Just as you wouldn't mix your screwdrivers with your hammers, modules keep related elements separate and organised. When you need a specific function or variable, you can "open the drawer" (import the module) and access the tools (functions and variables) inside.

CLASTES
VURDIDGLES
CLASSES
FNODNICITES
VARIAGBES
IMAGINED WITH AI IMAGINED WITH AI



module_1.py

def function_1()

def function_2()

class Class1

class Class2

CONSTANT_1

CONTSTANT_2

module_2.py

def function_3()

def function_3()

class Class3

class Class4

CONSTANT_3

CONTSTANT_4

script.py

```
import module_1
from module_1 import  function_1
from module_2 import  *
from module_2 import  Class3


cls = Class3()
print(function_1())
```

Differentiating Scripts, Modules, Packages, and Libraries

CoGrammar

Scripts

- A script is a standalone file containing executable Python code. It typically encapsulates a sequence of instructions to perform a specific task or set of tasks. It has the extension .py
- A Jupyter notebook is an interactive document that combines code, text, and visualisations in a browser-based environment, featuring cells for separate code execution and documentation, making automation challenging due to its interactive nature. It has the extension .ipynb

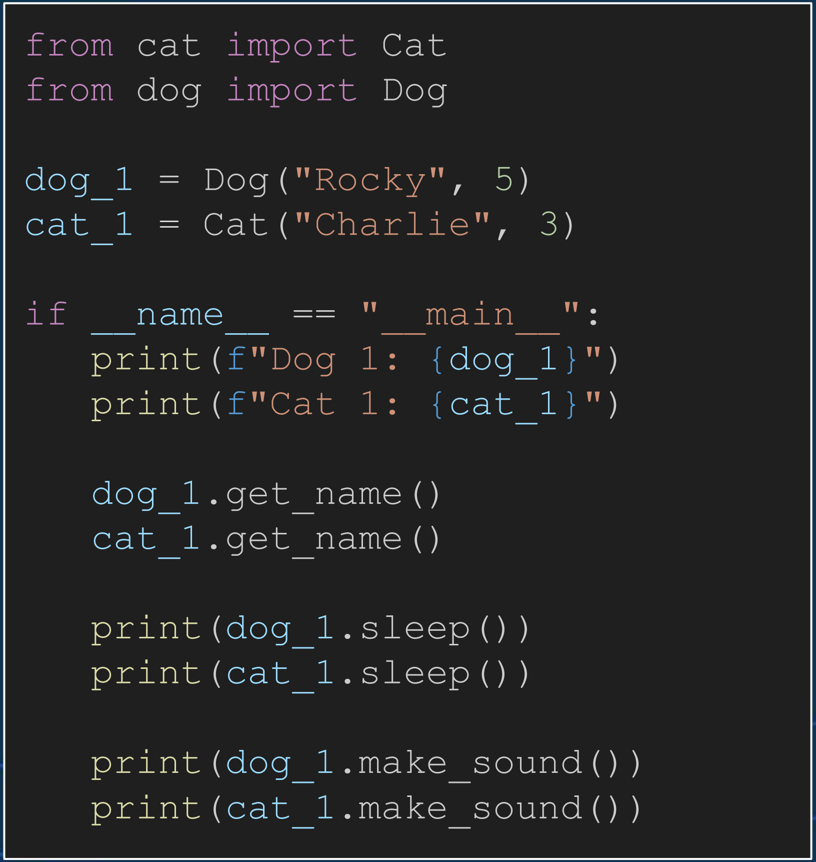
Scripts

- Scripts are designed to accomplish a particular goal or solve a specific problem
- They often automate repetitive tasks, process data
- Scripts can be standalone programs or part of a larger software system, focusing on a specific functionality or aspect of the application
- Shouldn’t be used to implement new classes or functions. Those are for modules.

Scripts

- "__name__" variable in Python:
  - Special variable managed by Python
  - Automatically set:
    - To "__main__" when script is run directly.
    - To module's name (filename) when executed as part of an import statement.

Scripts



```python
from cat import Cat
from dog import Dog

dog_1 = Dog("Rocky", 5)
cat_1 = Cat("Charlie", 3)

if __name__ == "__main__":
    print(f"Dog 1: {dog_1}")
    print(f"Cat 1: {cat_1}")

    dog_1.get_name()
    cat_1.get_name()

    print(dog_1.sleep())
    print(cat_1.sleep())

    print(dog_1.make_sound())
    print(cat_1.make_sound())
```

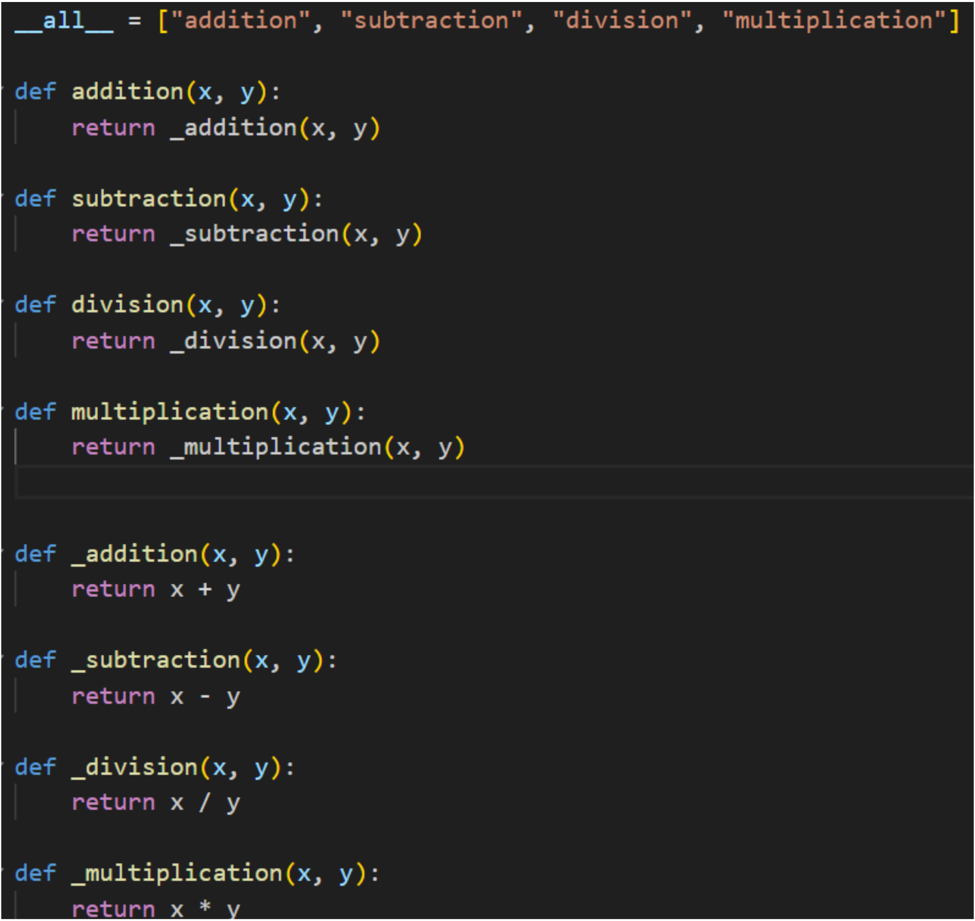
Module

- A module is a Python file (.py) that encapsulates reusable code elements such as functions, classes, and variables.
- They can be accessed by import-ing the module into other Python files (modules or scripts), enabling code reuse, maintenance and organisation
- Once imported, the functionalities defined in the module can be accessed and utilised in any script that imports it.

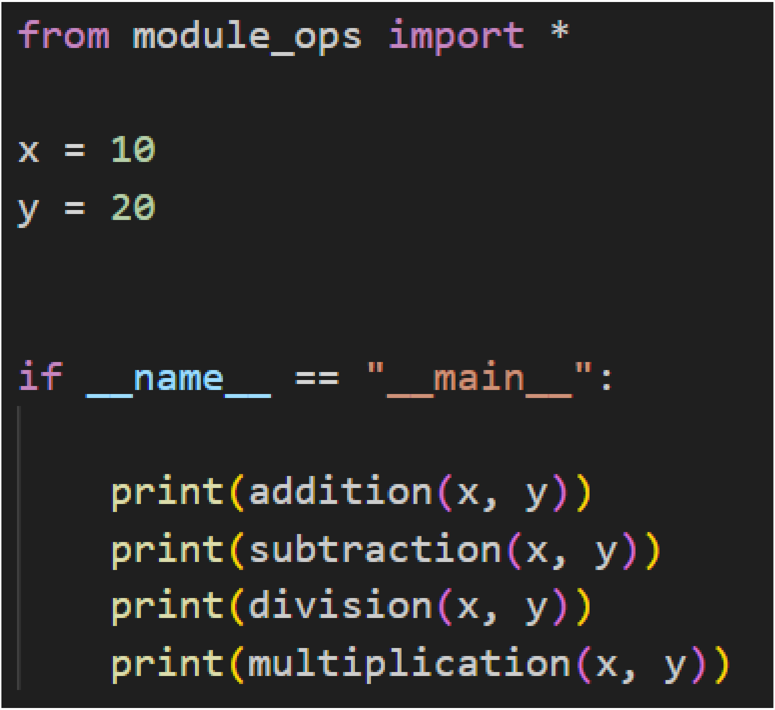
Module

Module-level names are global within the module, but they are not visible outside the module unless explicitly exported:

- Names defined at the module level, such as functions and variables, are accessible globally within the module
- However, these names are not visible to other scripts unless explicitly exported using techniques like the __all__ list or using the from module import * syntax
- This encapsulation ensures that module internals remain private unless explicitly exposed, promoting encapsulation and preventing namespace pollution.



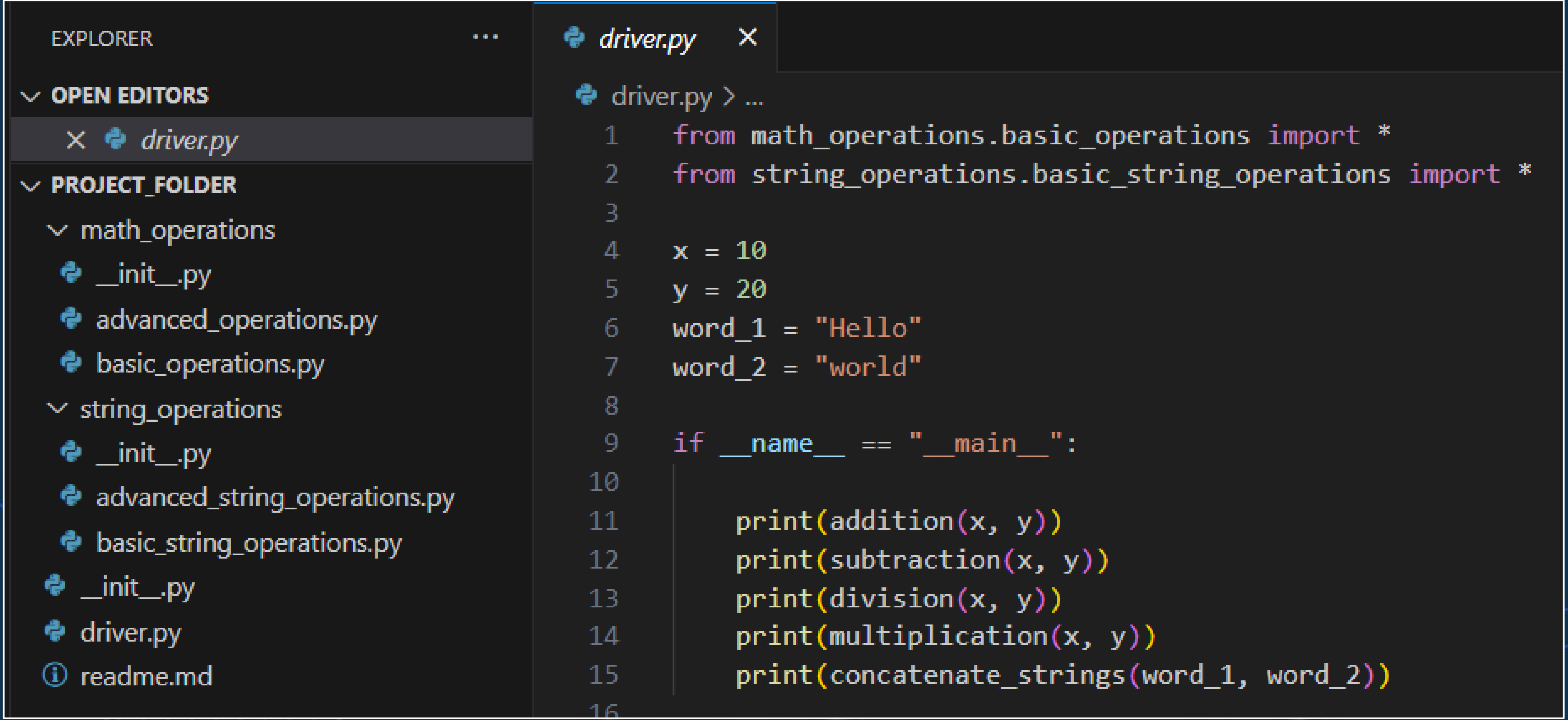
```python
__all__ = ["addition", "subtraction", "division", "multiplication"]

def addition(x, y):
    return _addition(x, y)

def subtraction(x, y):
    return _subtraction(x, y)

def division(x, y):
    return _division(x, y)

def multiplication(x, y):
    return _multiplication(x, y)


def _addition(x, y):
    return x + y

def _subtraction(x, y):
    return x - y

def _division(x, y):
    return x / y

def _multiplication(x, y):
    return x * y
```



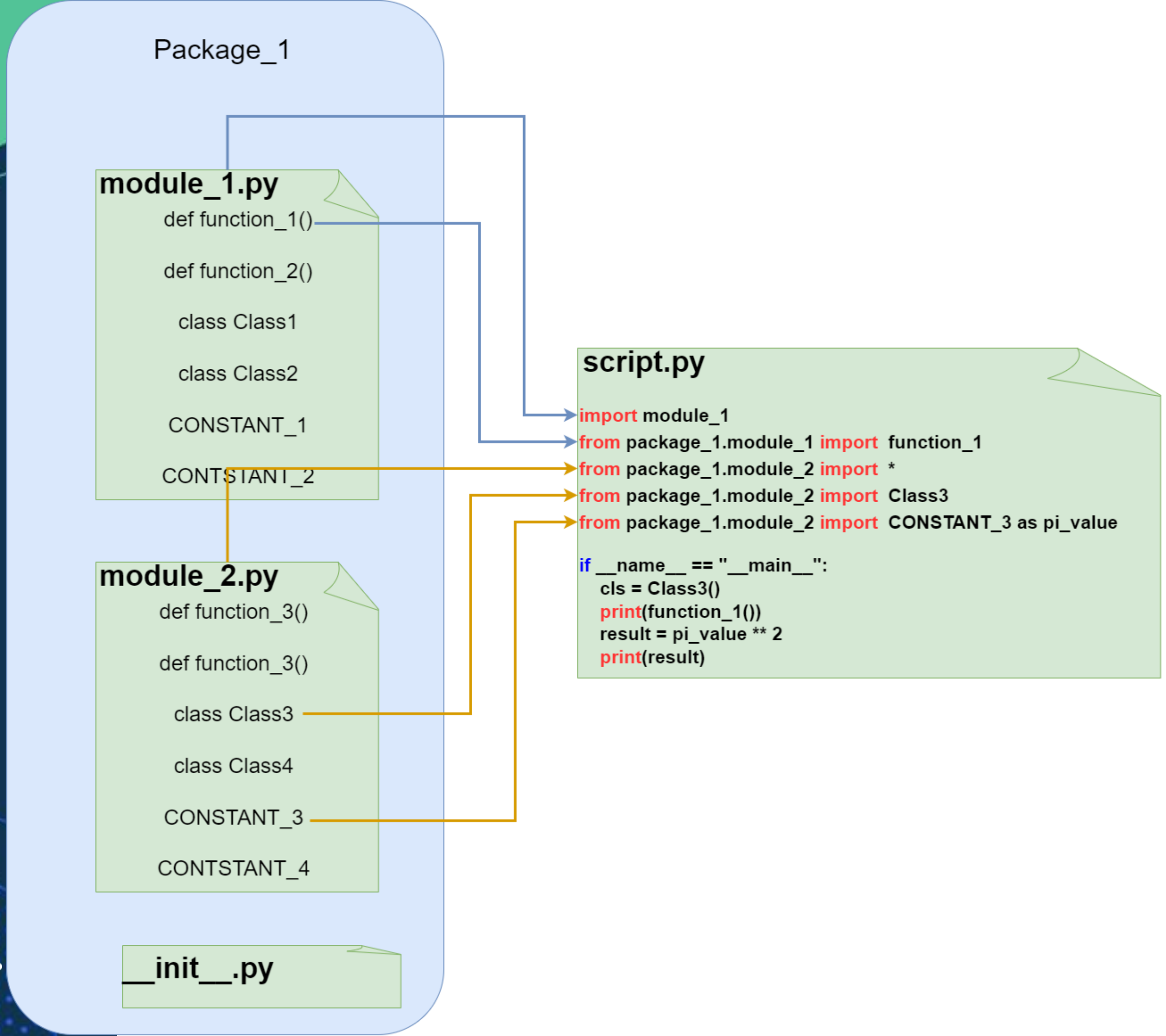
```python
from module_ops import *

x = 10
y = 20


if __name__ == "__main__":

    print(addition(x, y))
    print(subtraction(x, y))
    print(division(x, y))
    print(multiplication(x, y))
```

Package

- A package is a directory that contains Python modules, along with a special __init__.py file that signifies it as a Python package
- Packages are a way of structuring Python's module namespace by using "dotted module names". For example, the module name A.B designates a submodule named B in a package named A.
- The __init__.py file can be empty or contain initialisation code for the package
- This hierarchical structure aids in managing and navigating larger projects by grouping modules into logical units.



```python
from math_operations.basic_operations import *
from string_operations.basic_string_operations import *

x = 10
y = 20
word_1 = "Hello"
word_2 = "world"

if __name__ == "__main__":

    print(addition(x, y))
    print(subtraction(x, y))
    print(division(x, y))
    print(multiplication(x, y))
    print(concatenate_strings(word_1, word_2))
```



Package_1

module_1.py

def function_1()

def function_2()

class Class1

class Class2

CONSTANT_1

CONTSTANT_2

module_2.py

def function_3()

def function_3()

class Class3

class Class4

CONSTANT_3

CONTSTANT_4

__init__.py

script.py

```
import module_1
from package_1.module_1 import  function_1
from package_1.module_2 import  *
from package_1.module_2 import  Class3
from package_1.module_2 import  CONSTANT_3 as pi_value

if __name__ == "__main__":
    cls = Class3()
    print(function_1())
    result = pi_value ** 2
    print(result)
```

Library

- A library is fundamentally a collection of packages. Its objective is to offer a collection of ready-to-use features so that users won't need to be concerned about additional packages.

gunicorn
BeautifulSoup
Bandit
seaborn
Bottle
pandas
It's Dangerous

Why all that?

1. **Code Organisation**: Modules help organise code into logical units, making it easier to navigate and manage as your project grows.
2. **Reusability**: By encapsulating code into modules, you can reuse functions, classes, and variables across different parts of your program or in other projects, saving time and effort
3. **Maintainability**: Modular code is easier to maintain and update. Changes or fixes can be made to specific modules without affecting other parts of the codebase, leading to better organisation and collaboration among developers.

Let’s take a short break

CoGrammar

Python Standard Library Modules, pip and PyPi

CoGrammar

Python Standard Library (PSL)

- The **Python Standard Library (PSL)** is a collection of modules and packages that come pre-installed with Python.
- The PSL contains all the built-in functions commonly used like: **min**, **max**, **float**, **int**, **eval**, **print** ← Those do not even need to be imported
- It is considered to be the set of pillars building up the Python language
- The **Python Standard Library** is comprehensive, providing developers with tools to accomplish common tasks without having to install additional third-party packages.

PSL: Common Modules

Although, many python keywords do not need to be imported, some need the **import** keyword to be used. Those are built-in modules:

- **print**: Allowing you to perform mathematical operations
- **random**: generating random numbers
- **datetime**: Enables manipulation of dates and times
- **os**: Allows you to interact with the operating system
- **math**: Allowing you to perform mathematical operations

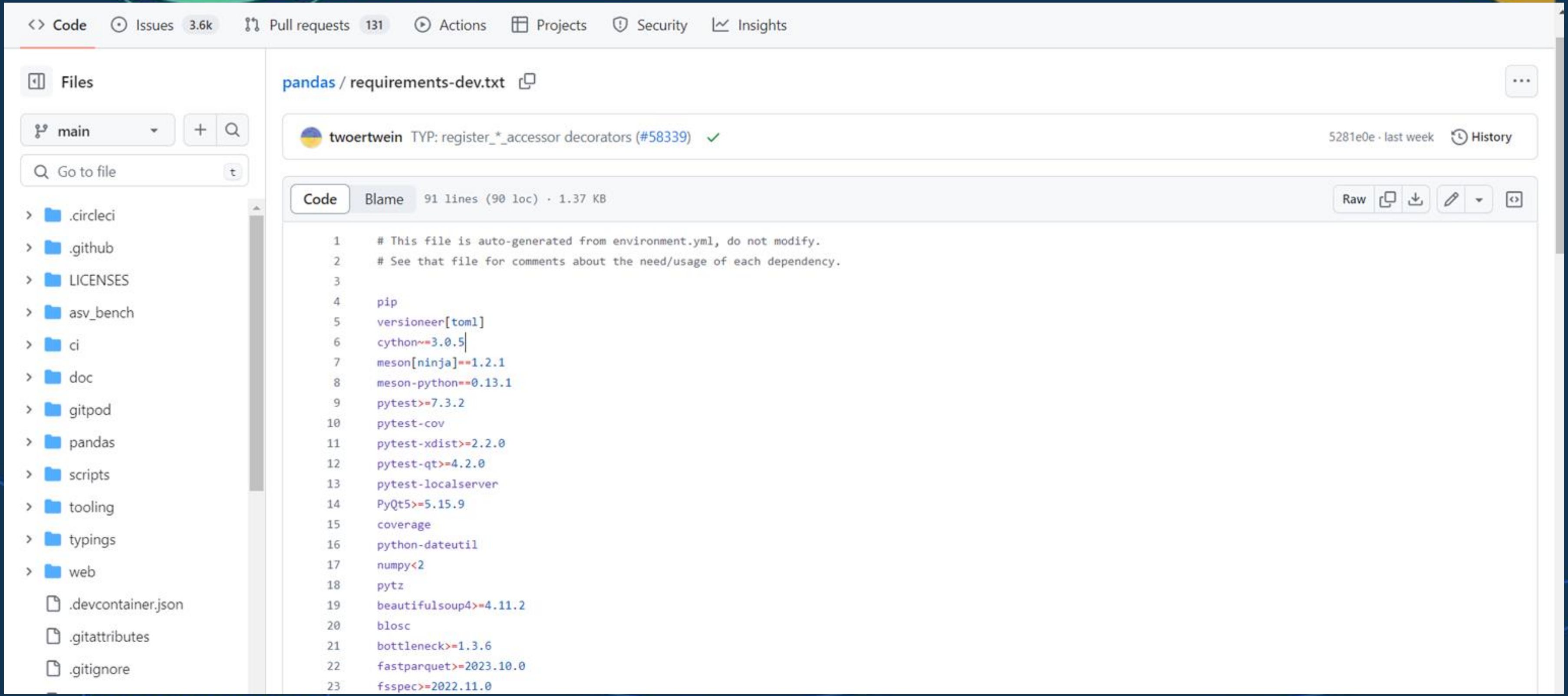
pip, PyPi

- **pip**: **Preferred Installer Program** is the package installer for Python. It allows you to install, upgrade, and manage Python packages from the Python Package Index (PyPi) or other sources. You can use pip to install third-party packages that are not included in the Python Standard Library.
- To install a new package: pip install new_package
- **PyPi**: PyPi is the official Python Package Index, a repository of software packages for Python. It hosts thousands of third-party packages that can be installed using pip.

Requirements File

- Requirements File
  - Text file listing required Python packages and versions
  - Usually call **requirements.txt**
  - Ensures **exact dependencies** are installed for your project
  - **Facilitates replicating your environment for others**

Requirements File



```
# This file is auto-generated from environment.yml, do not modify.
# See that file for comments about the need/usage of each dependency.

pip
versioneer[toml]
cython~=3.0.5
meson[ninja]==1.2.1
meson-python==0.13.1
pytest>=7.3.2
pytest-cov
pytest-xdist>=2.2.0
pytest-qt>=4.2.0
pytest-localserver
PyQt5>=5.15.9
coverage
python-dateutil
numpy<2
pytz
beautifulsoup4>=4.11.2
blosc
bottleneck>=1.3.6
fastparquet>=2023.10.0
fsspec>=2022.11.0
```

Virtual Environment

- Virtual Environment:
  - Self-contained directory with Python interpreter and libraries
  - Isolates project dependencies
  - Prevents conflicts between projects or system-wide installations

CoGrammar

Python Code Style and Type Hinting

CoGrammar

```python
# Examine this code and decide whether it is correct or not. Why?

def AreaOfCircle(radius):
    area = 3.14159 * radius**2
    return area

def Print_Area(radius):
    print("The Area of the Circle with Radius", radius, "is", AreaOfCircle(radius))

def get_radius():
    radius = float(input("Enter the Radius of the Circle:"))
    return radius

def main():
     Radius=get_radius()
Print_Area(Radius)

if __name__=="__main__":
    main()
```

PEP 8 - Python Style Guide

- PEP 8 is the official style guide for Python code, providing guidelines on formatting, naming, and organising Python code.
- It aims to promote consistency and readability in Python code across projects and developers.
- It provides guidelines and best practices on how to write Python code

PEP 8 - Python Style Guide

- **Indentation**: Use 4 spaces (not tabs) per indentation level.
- **Maximum Line Length**: Limit all lines to a maximum of 79 characters.
- **Comments:** Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!
- **Class Names:** Class names should normally use the CapWords convention.
- **Method Names and Instance Variables:** Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

CoGrammar

PEP 8 Law Enforcement: Linting

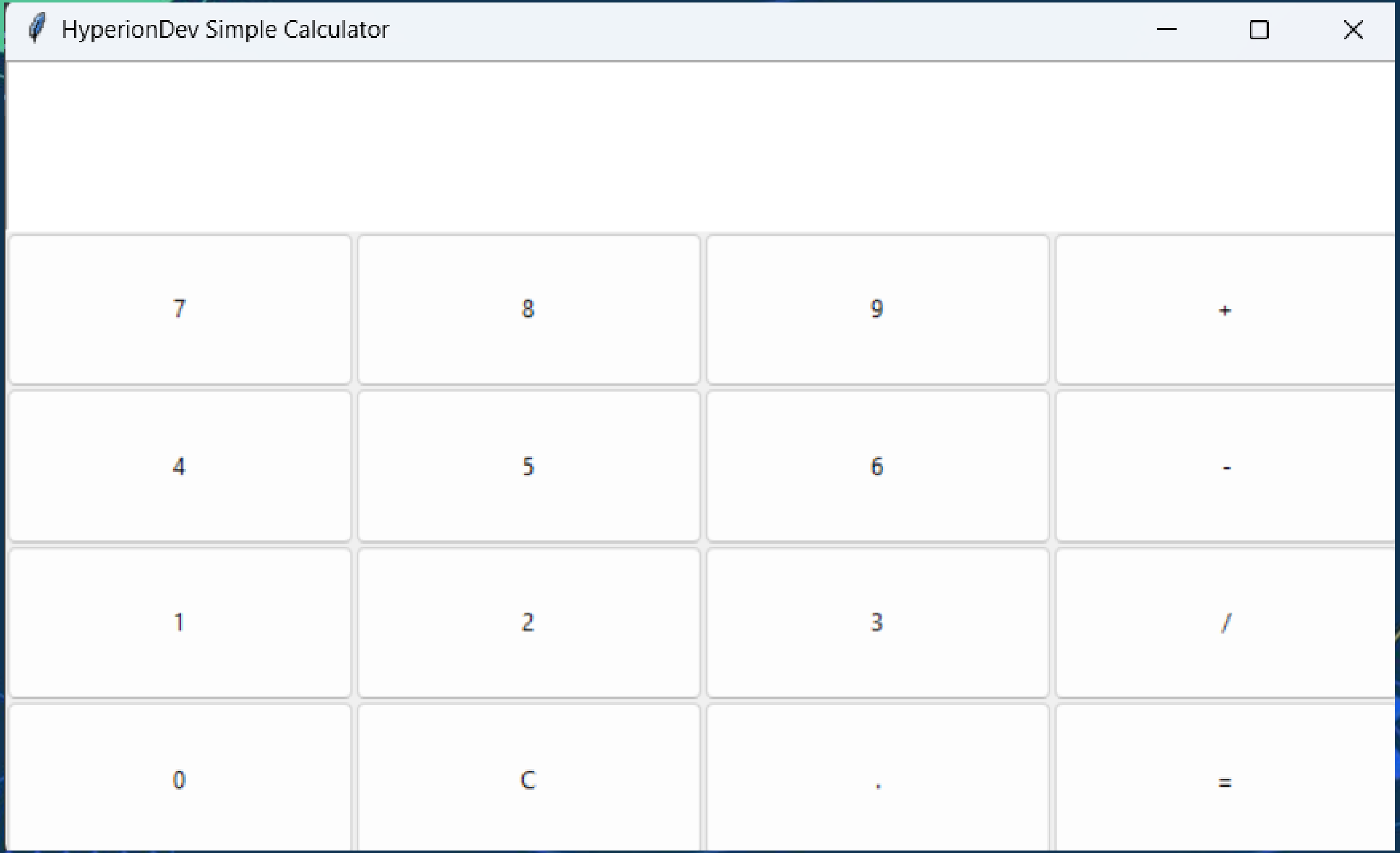
- **Linting (Lint)** is the automated checking of your source code for programmatic and stylistic errors.
- Flags unused constructs such as variables and unreachable code
- Helps standardise code by replacing tabs with spaces or the other way around so that the codebase is written consistently.
- Makes it easier to review code because it ensures the reviewer that certain standards are already met.

Linting: Common Tools

Black
flake8
: my[py]
RUFF v0.1.0

Practical Exercise: Building a GUI Python Calculator

CoGrammar


HyperionDev Simple Calculator
7
8
9
+
4
5
6
-
1
2
3
/
0
C
.
=

Exercise Objectives

I. Design the Calculator GUI:

   a. Design the layout of the calculator interface using tkinter, including buttons for digits, arithmetic operations, and clear/reset functionality.

   b. Organise GUI elements using layout managers (grid, pack, or place) and consider using frames for better organisation.

II. Implement the Calculator Logic:

   a. Create a Calculator class to encapsulate the logic and functionality of the calculator.

   b. Define methods within the Calculator class to perform arithmetic operations (addition, subtraction, multiplication, division) and handle user input.

Exercise Objectives

III. Modularisation and Inheritance

a. Organise the code into separate modules for better maintainability and code organisation.

b. Create a module for the calculator GUI layout and functionality, and another module for the calculator logic

c. Utilise inheritance to extend functionality, if applicable (e.g., creating specialised calculator classes)

IV. Testing and Debugging:

a. Test the calculator’s arithmetic operations

b. Debug any errors or issues encountered during testing, ensuring the calculator functions as expected

c. Use print statements or logging to debug and trace the flow of execution if necessary.

Polls

CoGrammar

Poll

1. **Which of the following accurately describes the purpose of Python modules?**
    A. To organise code into reusable units and promote maintainability.
    B. To execute specific tasks within a Python script
    C. To provide graphical user interfaces (GUIs) for Python applications.
    D. To manage dependencies between Python packages

Poll

2. **Given a Python script named main.py and a module named my_module.py in the same directory, what is the proper way to import the my_module module within main.py?**

   A. import my_module

   B. from my_module import *

   C. import my_module.py

   D. import .my_module

Lesson Conclusion and Recap

CoGrammar

Lesson Conclusion

- Understanding Modules
  - Modules are used in Python to organise code into reusable units, enhancing maintainability and readability.
- Exploring Standard Library Modules
  - Python's Standard Library offers a wide range of modules for common tasks, such as math calculations, file manipulation, and datetime operations.
- Differentiating Components
  - Scripts, modules, packages, and libraries serve distinct purposes in Python, with modules acting as reusable units of code.

Lesson Conclusion

- **Creating Custom Modules**
  - Create own modules by encapsulating related code in separate `.py` files, promoting code organisation and reusability.
- **Importing Modules**
  - Modules are imported into Python scripts using the `import` statement, providing access to their contents.
- **Understanding Object-Oriented Principles**
  - Object-oriented programming principles like encapsulation and inheritance can be applied to modularisation, allowing for the creation of versatile and extensible modules.

Learner Challenge

CoGrammar

Learner Challenge

Use the code from the practical and add a factorial operation such that 5! = 120.

1. Add the factorial (!) button at the position of your choice
2. Add the factorial functionality
3. Make sure that the result comes out on the screen
4. Have a test case for it
5. Do not break the rest of the code

Questions and Answers

CoGrammar

References

- https://docs.python.org/3/library/index.html
- https://docs.python.org/3/library/functions.html
- https://www.toppr.com/guides/python-guide/references/methods-and-functions/python-standard-library-reference/
- https://peps.python.org/pep-0001/#what-is-a-pep
- https://peps.python.org/pep-0008/
- https://designenterprisestudio.com/2022/05/26/libraries-frameworks/

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar