A logo of the university of scotland

Description automatically generated

School of Computing, Engineering and Physical Sciences

MSc Information Technology

**COMP11124**

**Object Oriented Programming**

**Course Work**

**Submitted By:**

|  |  |
| --- | --- |
| **Banner ID** | **Name** |
| Sambat Adhikari | B01804202 |
| Mahesh Regmi | B01805066 |
| Suraj Shrestha | B01811462 |
| Jasleen | B01804328 |
| Muhammad Fahad | B01813165 |

**15 July 2025**

Table of Contents

[1. INTRODUCTION 3](#_Toc202136643)

[2. Week 1 – Introduction to Python and Development Environment 4](#_Toc202136644)

[2.1 Python Overview 4](#_Toc202136645)

[2.2 Setting up Visual Studio Code (VS Code) 6](#_Toc202136646)

[2.3 Outcome 9](#_Toc202136647)

# INTRODUCTION

The aim of this course work report is to introduce and build upon the programming skills using Python language, with main focus on object-oriented programming (OOP) paradigm. The following report contains a collection of programs and exercises completed as a group to demonstrate the skills and knowledge obtained during the study of Object-Oriented Programming module (COMP11124).

The report consists of multiple concepts learned and practiced from week 2 through to week 8. Every week focuses on a specific number of programming concepts, starting from basic fundamental aspects of python programming to more complex and object-oriented concepts such as class and objects.

This coursework is undertaken by five members, all of whom have worked together to contribute to the completion of tasks, explaining the logic and techniques to apply theoretical knowledge into practical implementation. The report also includes relevant code snippets, screenshots and explanations. The structure of the report follows a week-by-week breakdown ensuring clarity and relevance to module’s learning outcomes.

# Week 1 – Introduction to Python and Development Environment

In Week 1, the goal was to gain a brief insight into the development environment and develop a basic foundation of the Python programming language. Before initiating the learning process, it was crucial to set up the proper tools required for writing and executing python code.

### 2.1 Python Overview

Python is a high-level, interpreted programming language known for its simple syntax and readability. It was created by Guido van Rossum and released in 1991. The most recent major version of Python is Python 3, which is used for the preparation of this report.

Python has similarities to the English language with influence from mathematics. It is a widely popular programming languages in the computer world that can be used for a variety of applications like building web services, software development, handling big data or performing complex mathematics.

**Installation of Python**

Download and install Python from <https://www.python.org/downloads/>



Figure 1: Python Download

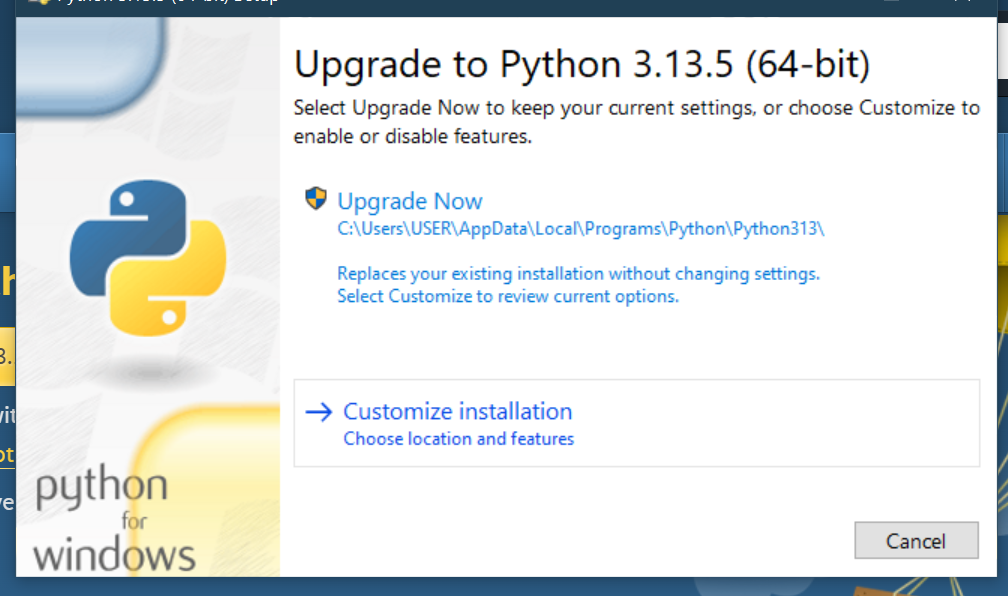


Figure 2: Python Installation

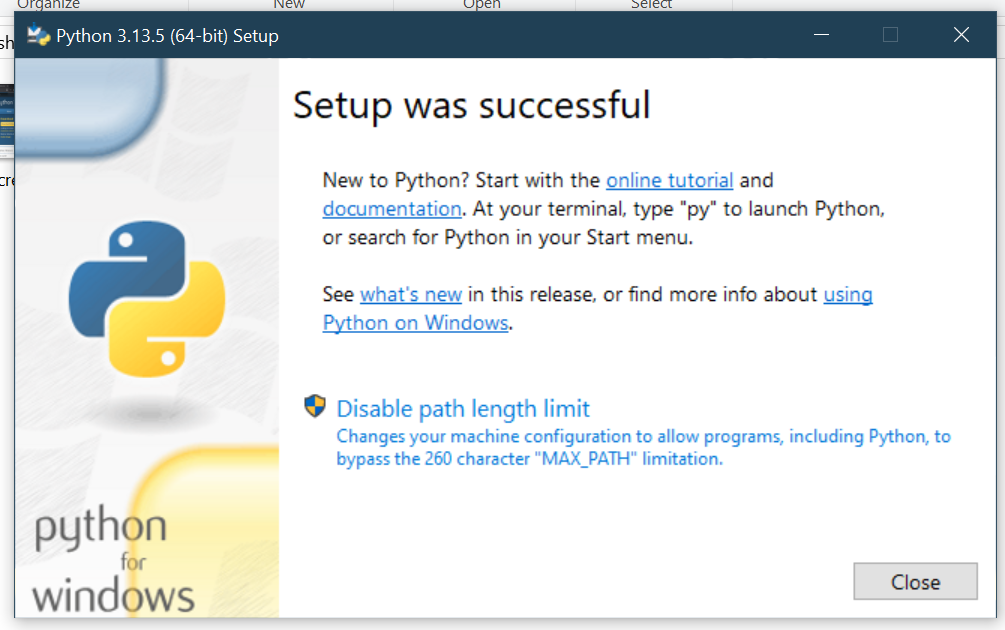


Figure 3 : Python Setup Completed

### 2.2 Setting up Visual Studio Code (VS Code)

Visual Studio Code was chosen as the integrated development environment (IDE) for this coursework. VS Code is a lightweight and opensource IDE developed by Microsoft offering feature such as:

1. Support for Python through the Python extension.
2. Built-in debugger to improve development process.
3. Provides fully featured integrated terminal to run shell commands.
4. User-friendly interface, syntax highlighting and intelligent code completion.

**Installation of VS Code**

Download and install Visual Studio Code from <https://code.visualstudio.com/download>

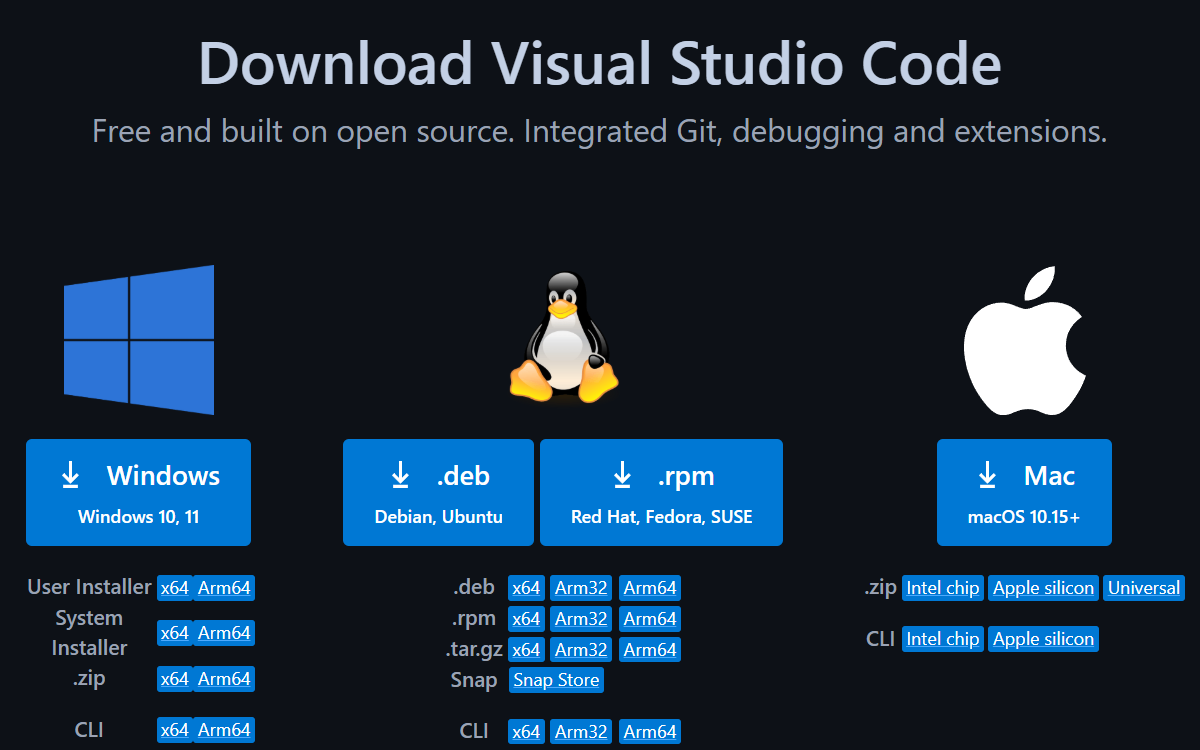


Figure 4:Visual Studio Code Download

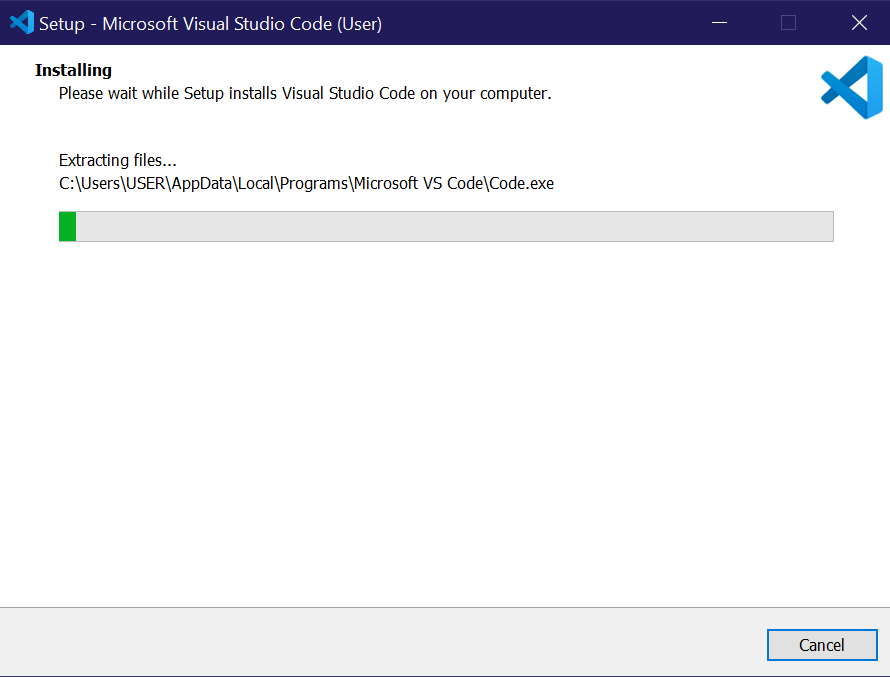


Figure 5: Visual Studio Code Installation

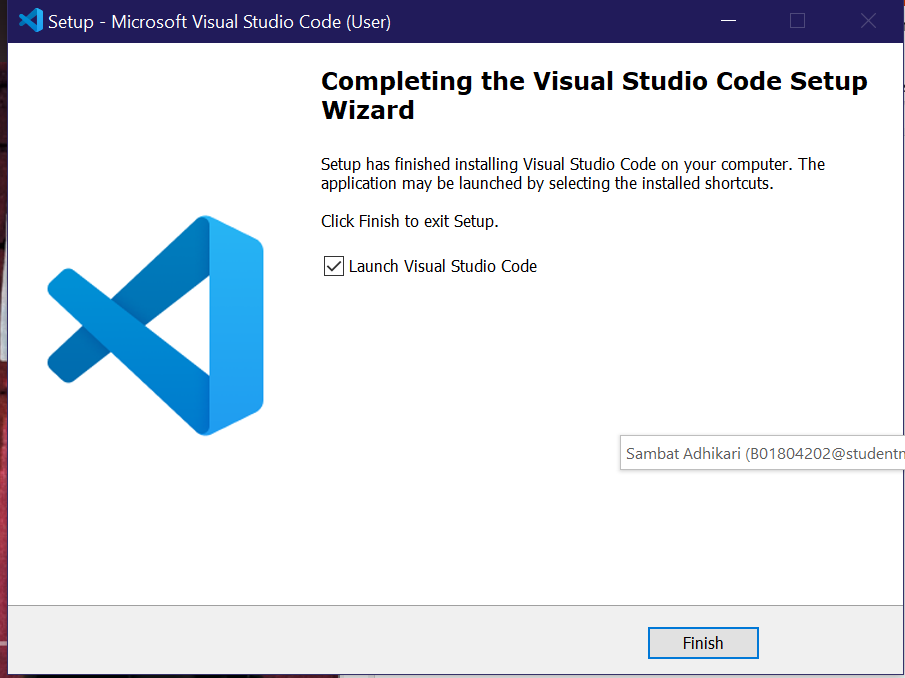


Figure 6: Visual Studio Code Setup Completed

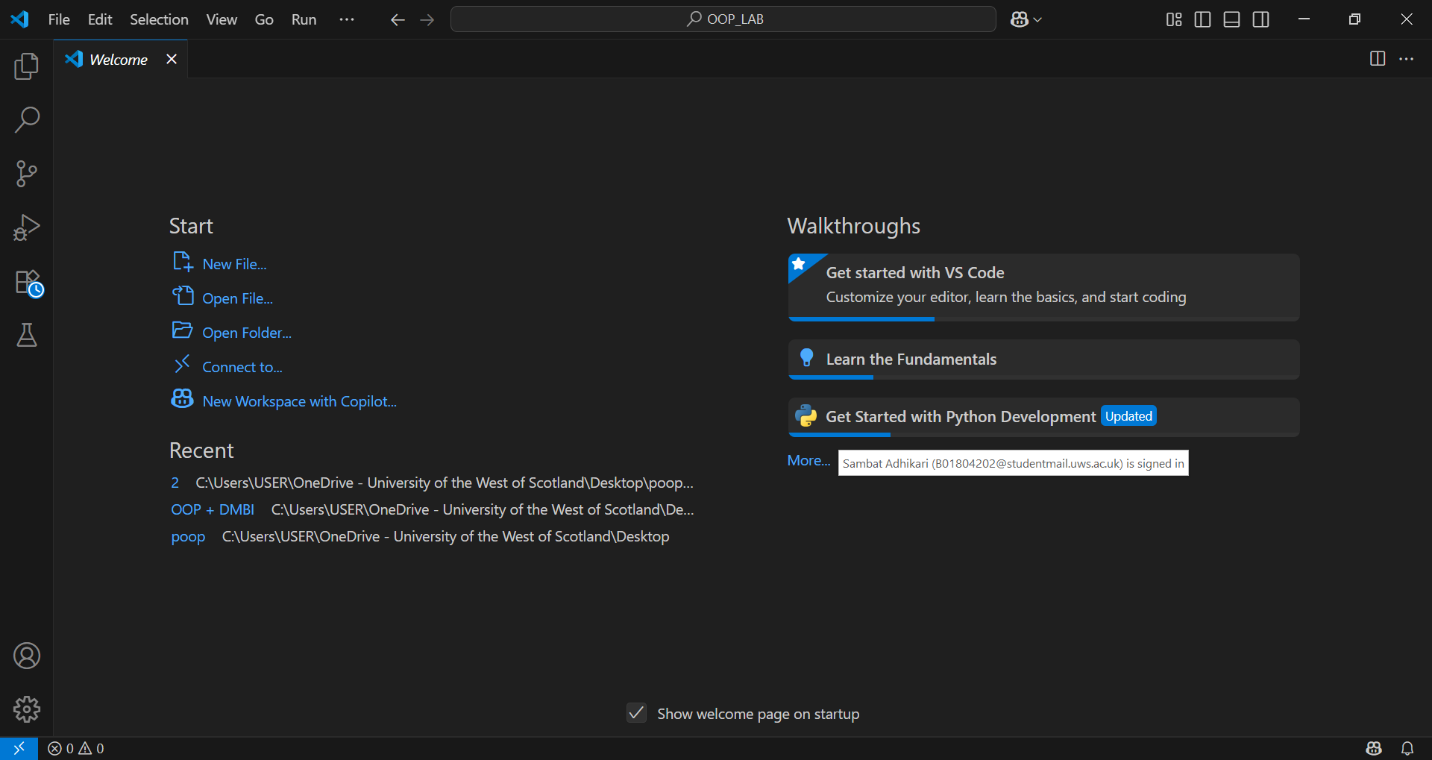


Figure 7: Visual Studio Code

After installation of Visual Studio Code, we also install Python extension in VS Code.

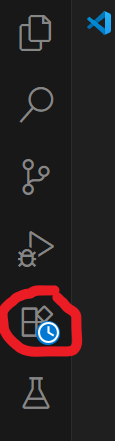


Figure 8: Side Bar containing extension



Figure 9: Installation of Python extension

### 2.3 Writing our first Python Program

After the setup of VS Code, we write our first python program “Hello World” to ensure everything is working properly.



Figure 10: "Hello World" Program

### 2.3 Outcome

By the end of Week 1, all group members had:

1. Successfully installed Python 3 and Visual Studio Code
2. Installed the Python extension within VS Code
3. Executed their first Python program

This ensures a consistent working environment across all group members for future lab exercises.

# 3.Week 2 – Comparisons, Conditionals, Lists, Loops and Input

In Week 2, we dived deeper into the fundamental concepts of Python programming to lay the foundation later required for complex needs. In this time period we learned about concepts that allows us to write basic programs that make decisions, repeat instructions, handle multiple values and interact with user.

### 3.2 Comparison and Logical Operators

**3.2.1. Comparison Operator**

We use comparison operators to compare two or more values. The result of a comparison operator is a Boolean value (either True or False).

The comparison operators used in Python are:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| > | Greater than | 5 > 4 | True |
| 4 > 5 | False |
| < | Less than | 4 < 5 | True |
| 5 < 4 | False |
| == | Equal to | 5 == 5 | True |
| 4 == 5 | False |
| != | Not Equal to | 4 != 5 | True |
| 4 != 4 | False |
| >= | Greater than or equal to | 5 >= 4 | True |
| 4 >= 5 | False |
| <= | Less than or equal to | 4 <= 5 | True |
| 5 <= 4 | False |

**3.2.2. Logical Operator**

We use logical operators to evaluate combine multiple conditions to return a single value. The result of logical operator is also a Boolean value.

The logical operators used in Python are:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Description | Example | Result |
| and | Returns TRUE if all  conditions are TRUE | 5 > 4 and 5 > 2 | True |
| 5 > 4 and 5 > 6 | False |
| or | Returns TRUE if at least  one condition is TRUE | 4 < 5 and 4 < 6 | True |
| 4 < 5 and 4 < 3 | False |
| not | Reverses the result  TRUE if FALSE,  FALSE if TRUE | not (4 > 5) | True |
| not (4 < 5) | False |

**1. Introduction**

This report documents the completion of Week 2's practical exercises focusing on Python programming fundamentals. The lab covered:

* Conditional statements and logical operators
* List creation and manipulation
* Loop structures (while and for loops)
* User input handling

All exercises were implemented in a single Python file (lab\_week\_2.py) with extensive testing and verification.

**2. Implementation Details**

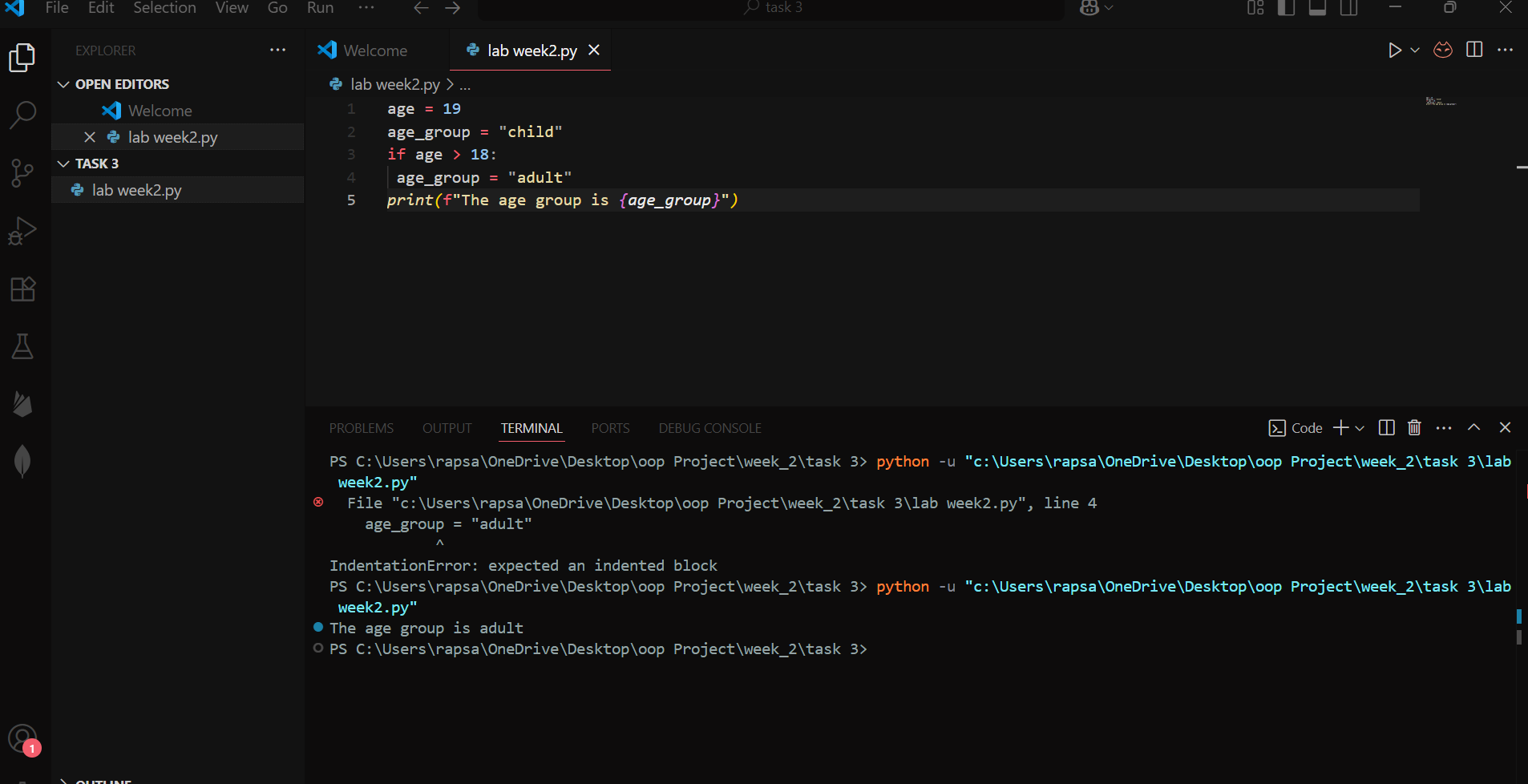
**2.1 Conditionals**

**Basic Conditional Statements**

**1. Conditionals and Comparisons**

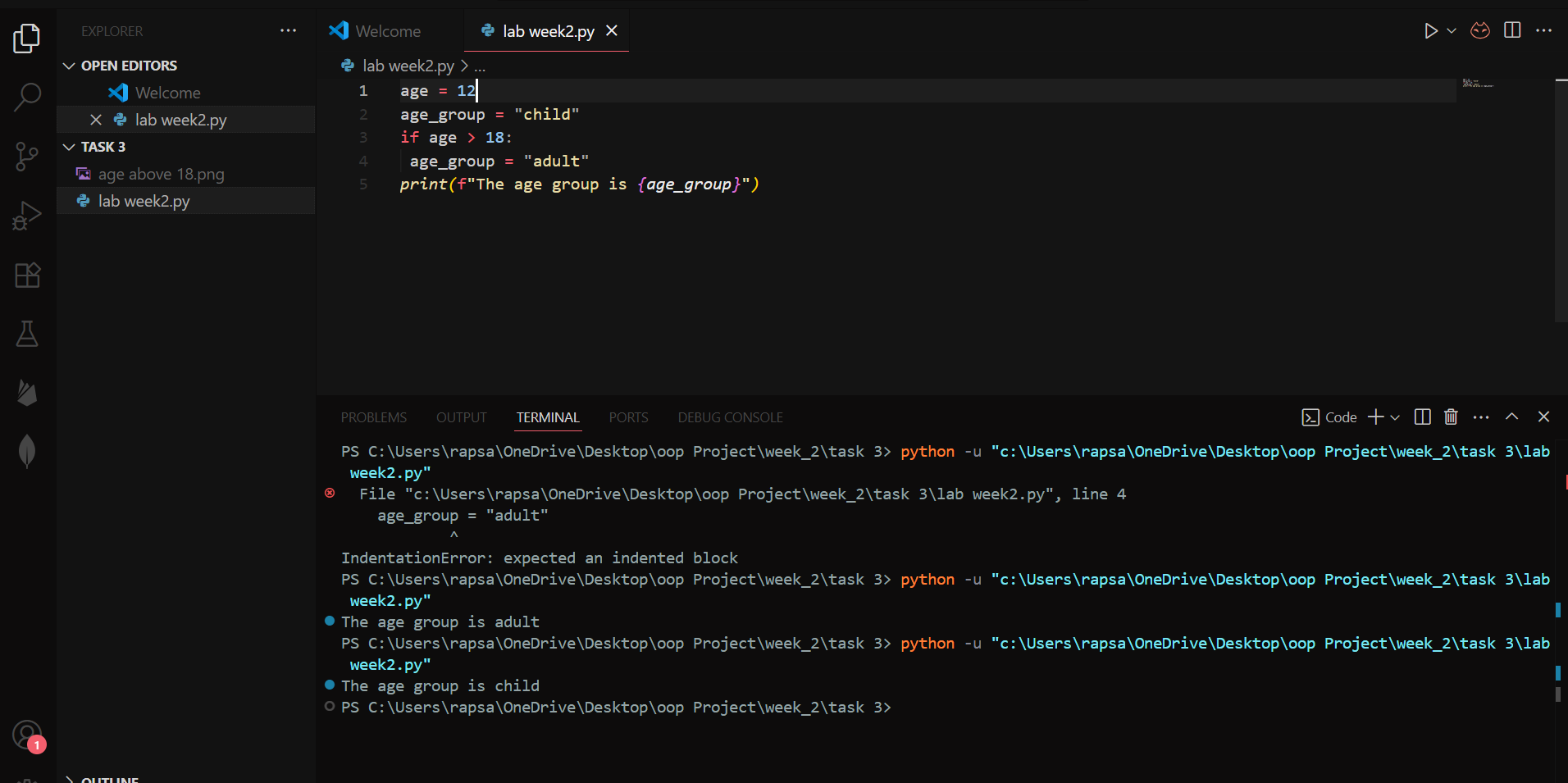
**Task 1: Basic If Conditional**

**Objective:** Demonstrate simple conditional logic based on age  
**Code:**



**Explanation:**  
The code checks if age is greater than 18. Since 19 > 18, the age\_group remains "adult".

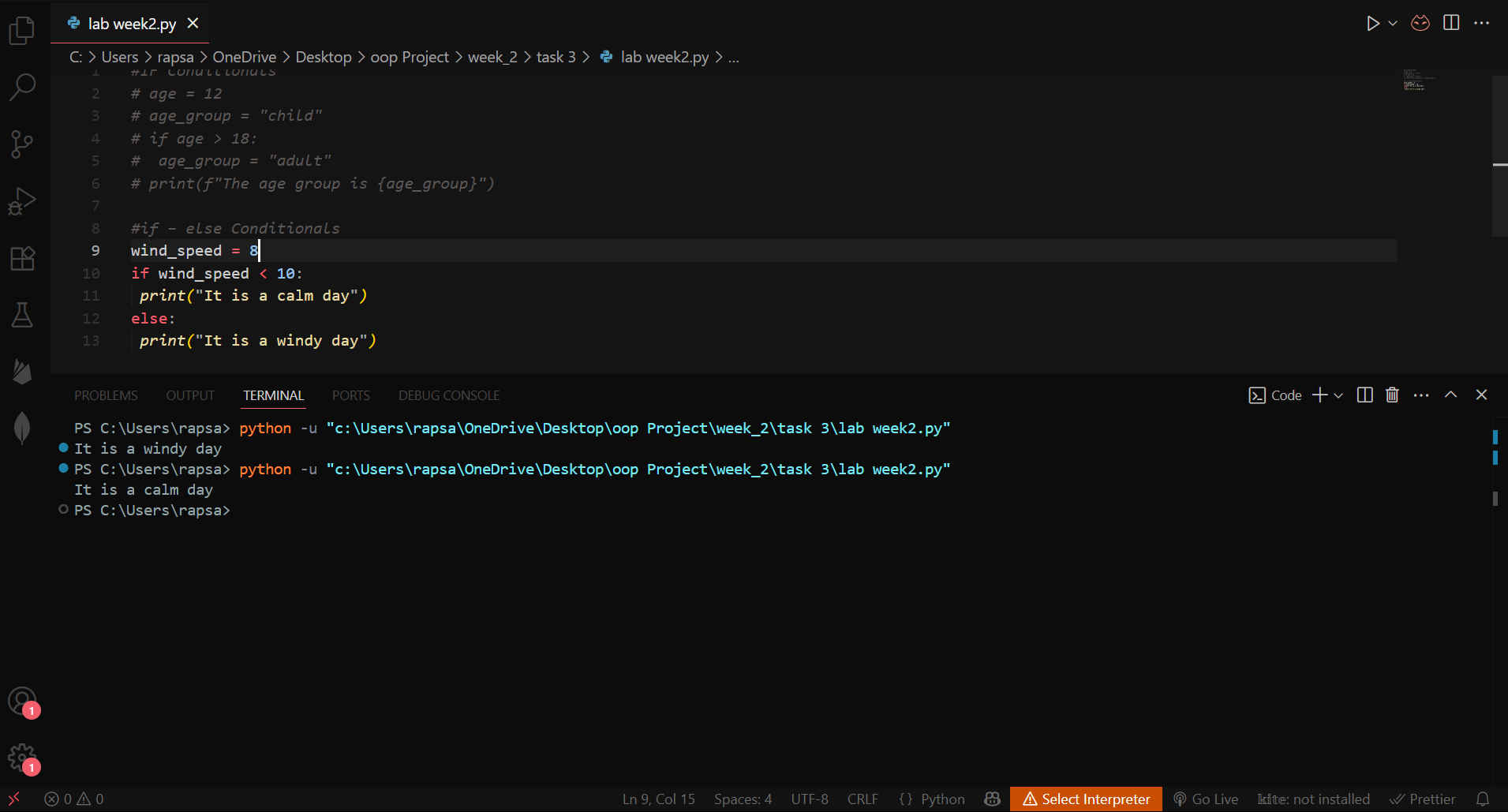
Altering input to get different output



**Explanation:**  
The code checks if age is greater than 18. Since 12 < 18, the age\_group remains "child"

**Task 2: If-Else Conditional**

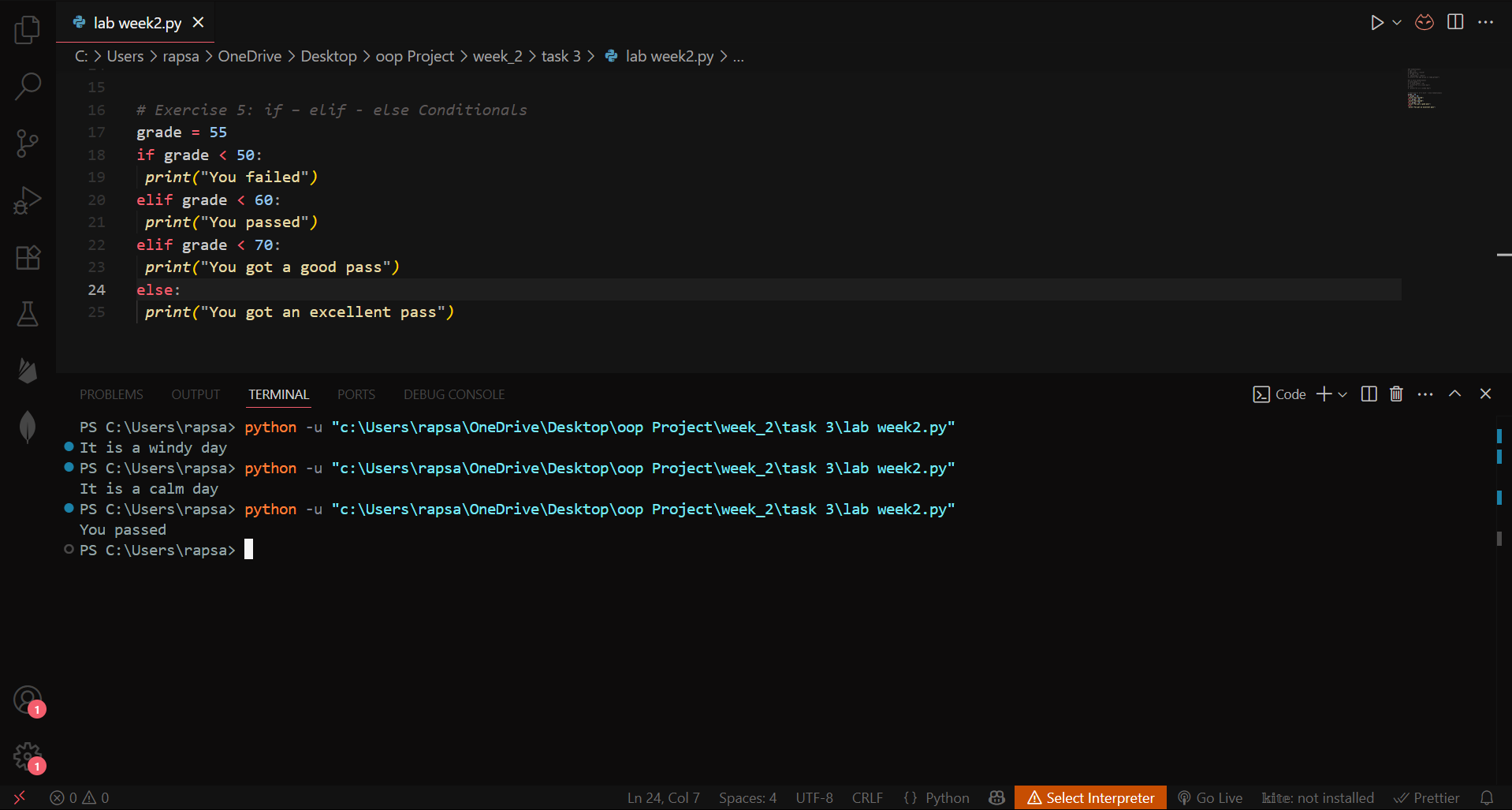
**Objective:** Implement binary decision making based on wind speed  
**Code:**

  
**Explanation:**  
The condition evaluates to True (8 < 10), executing the first print statement.

Checked and executed both conditions and shown on terminal.

**Task 3: If-Elif-Else Ladder**

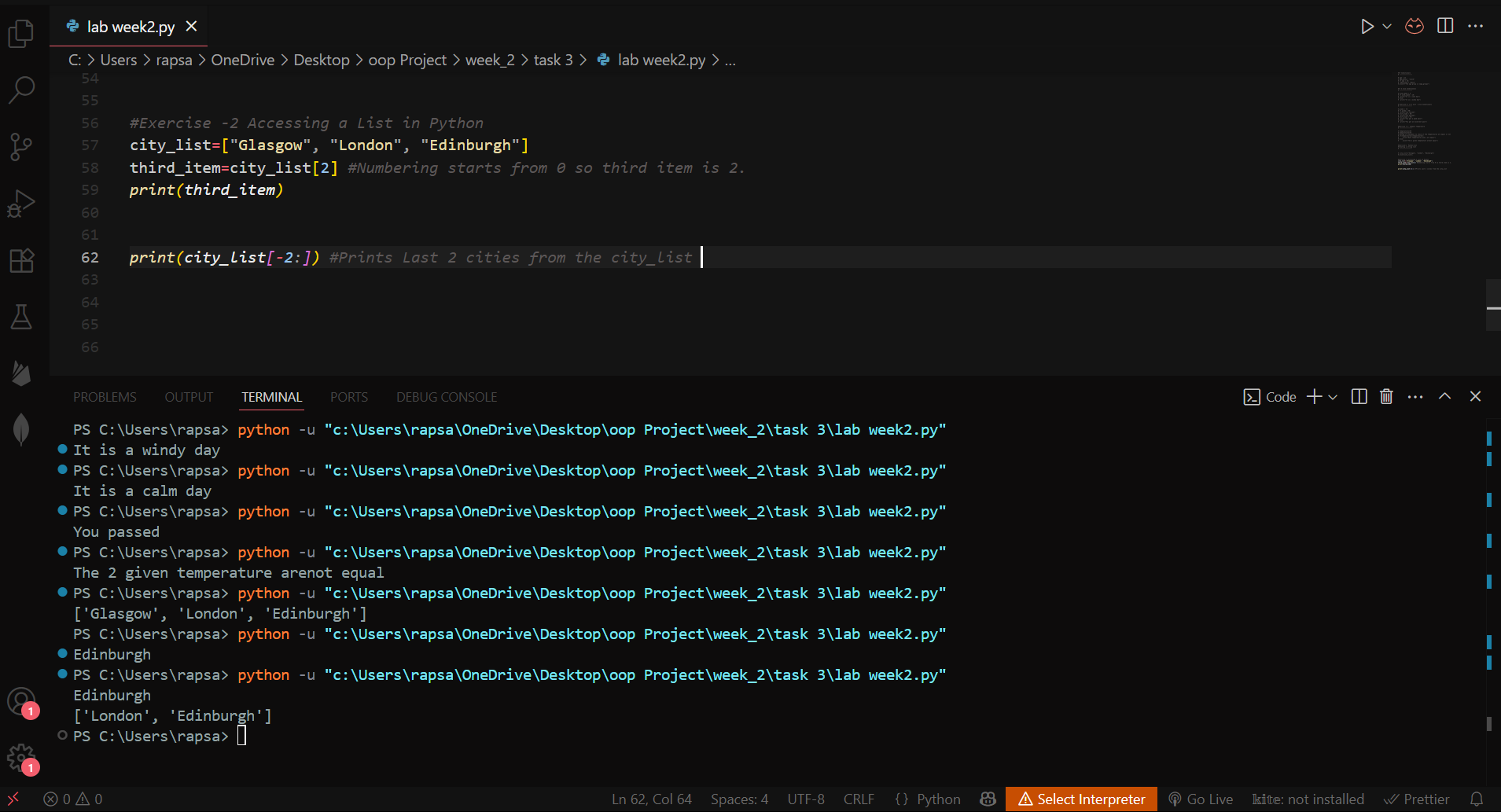
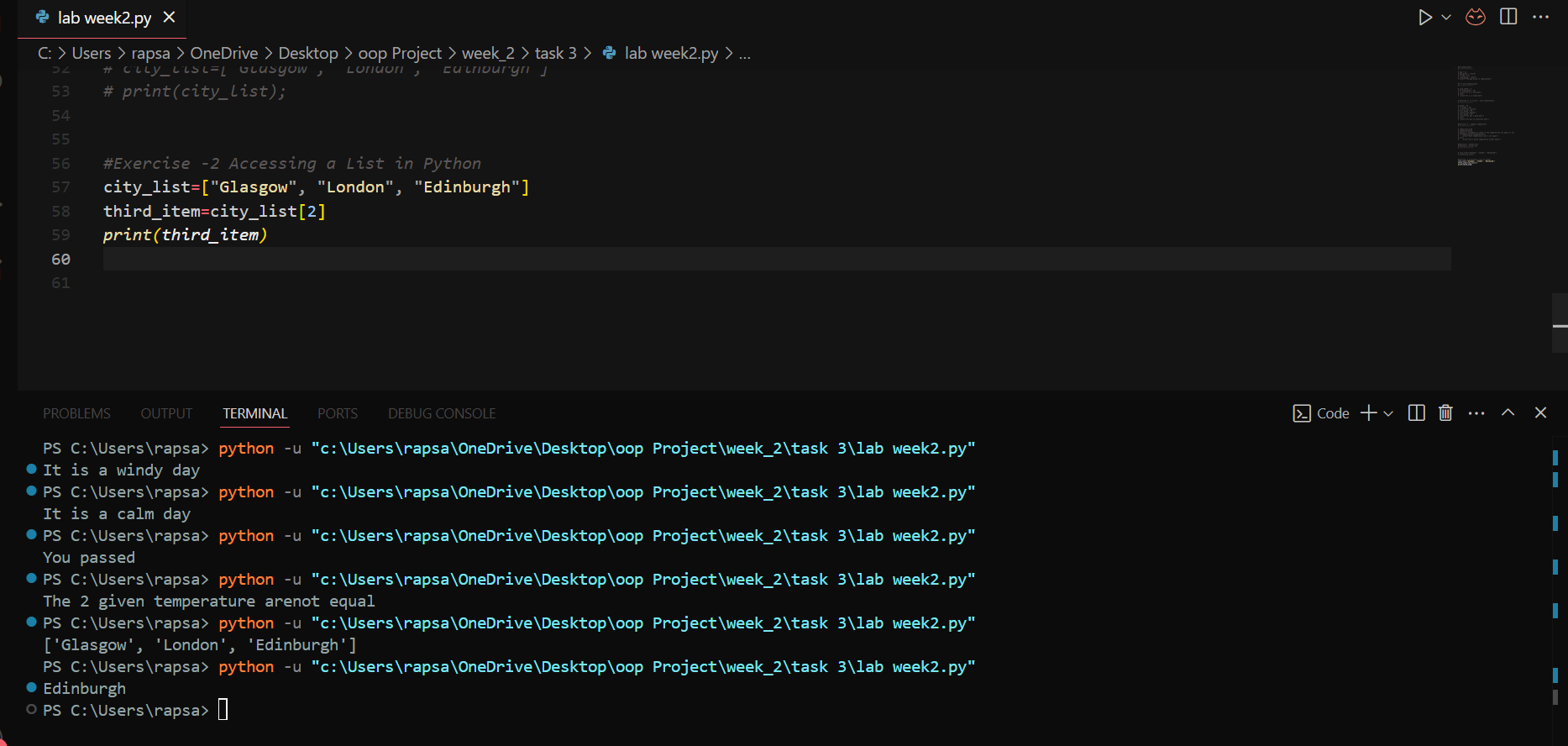
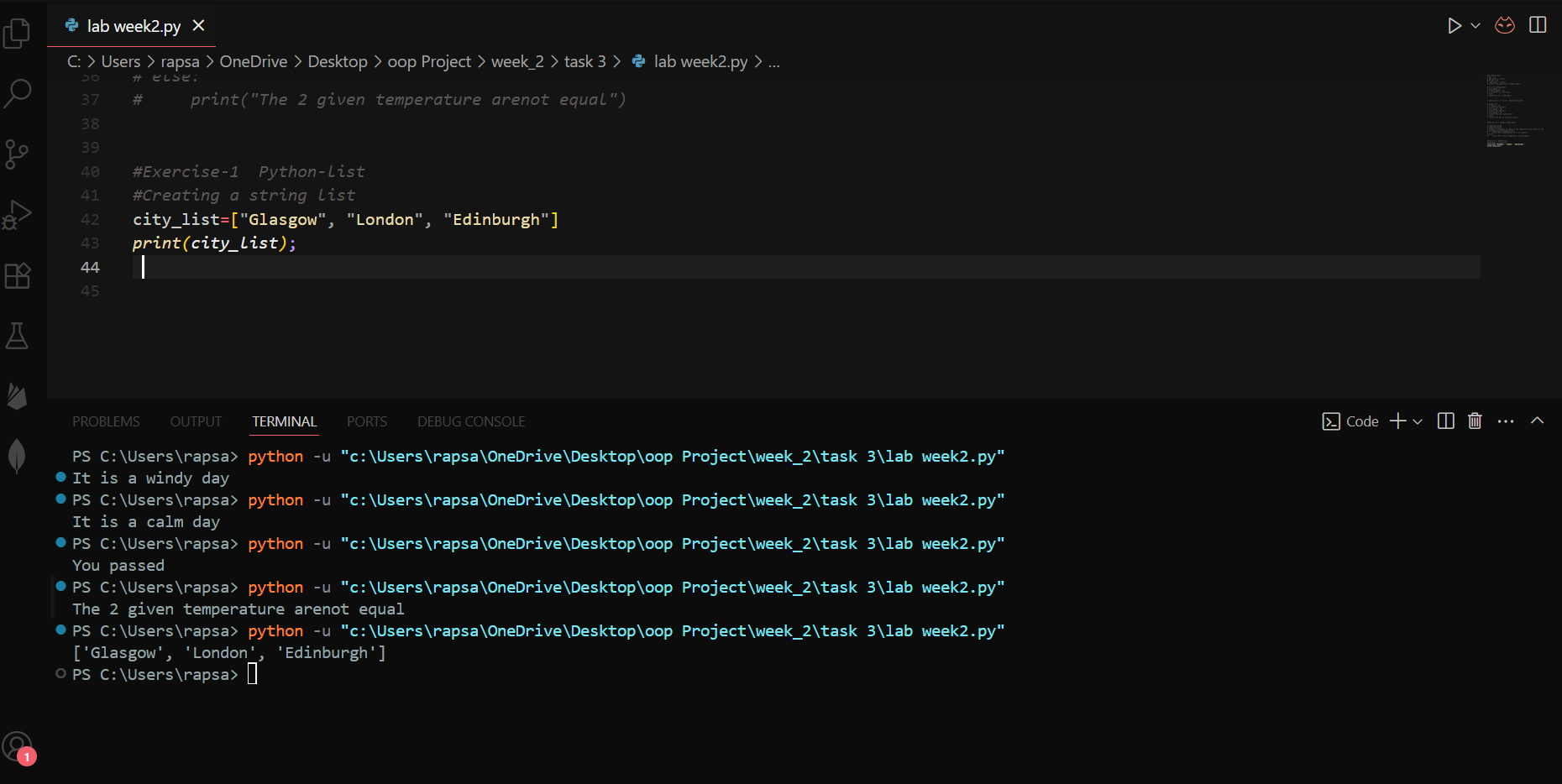
**Objective:** Create multi-condition grading system  
**Code:**



**Explanation:**  
The grade 55 falls in the else condition (>60), triggering “You are passed”.

**2. Python Lists**

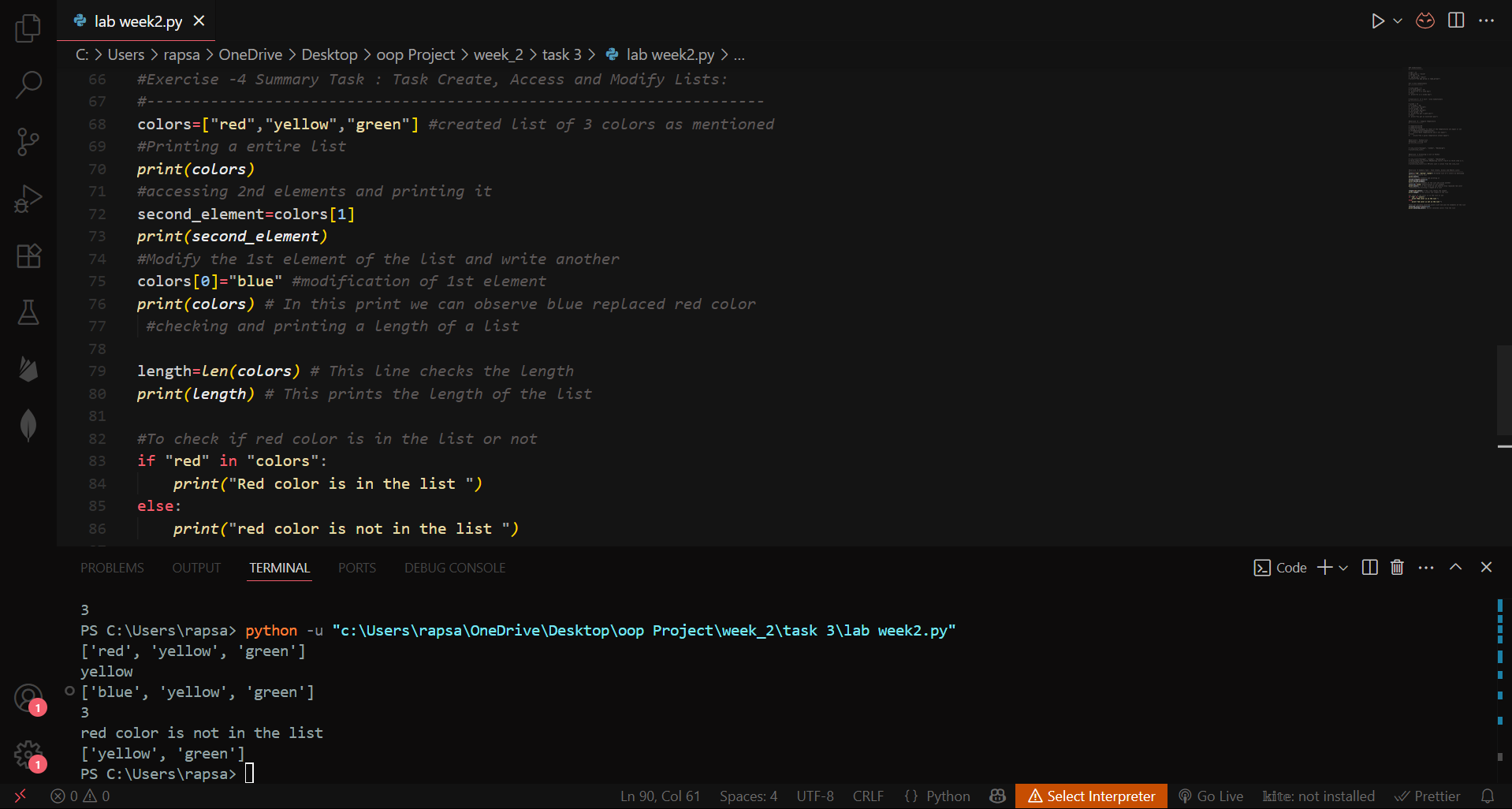
**Task 4: List Creation, Access and Slicing**

**Objective:** Work with list indexing and slicing  
**Code:**

**Explanation:**  
Demonstrates accessing the list in python, zero-based indexing and negative slicing to access the particular element of any list.

**Task 5: List Modification**

**Objective:** Update list elements and append new items  
**Code:**

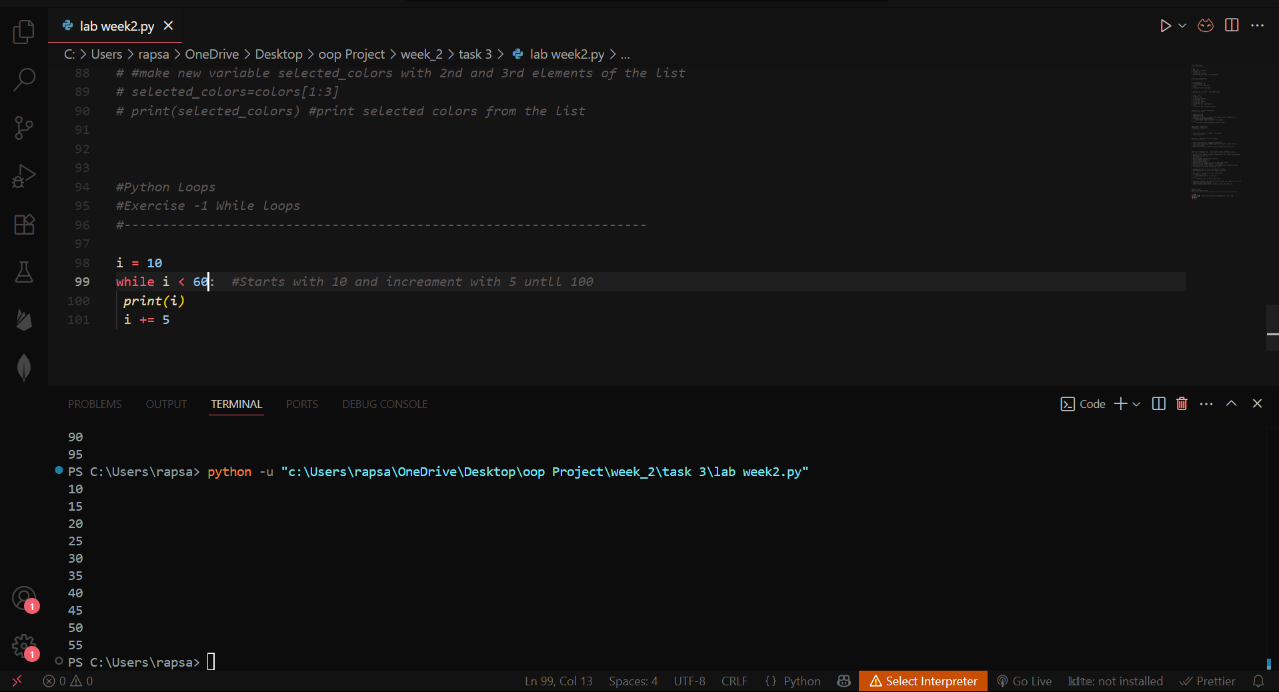


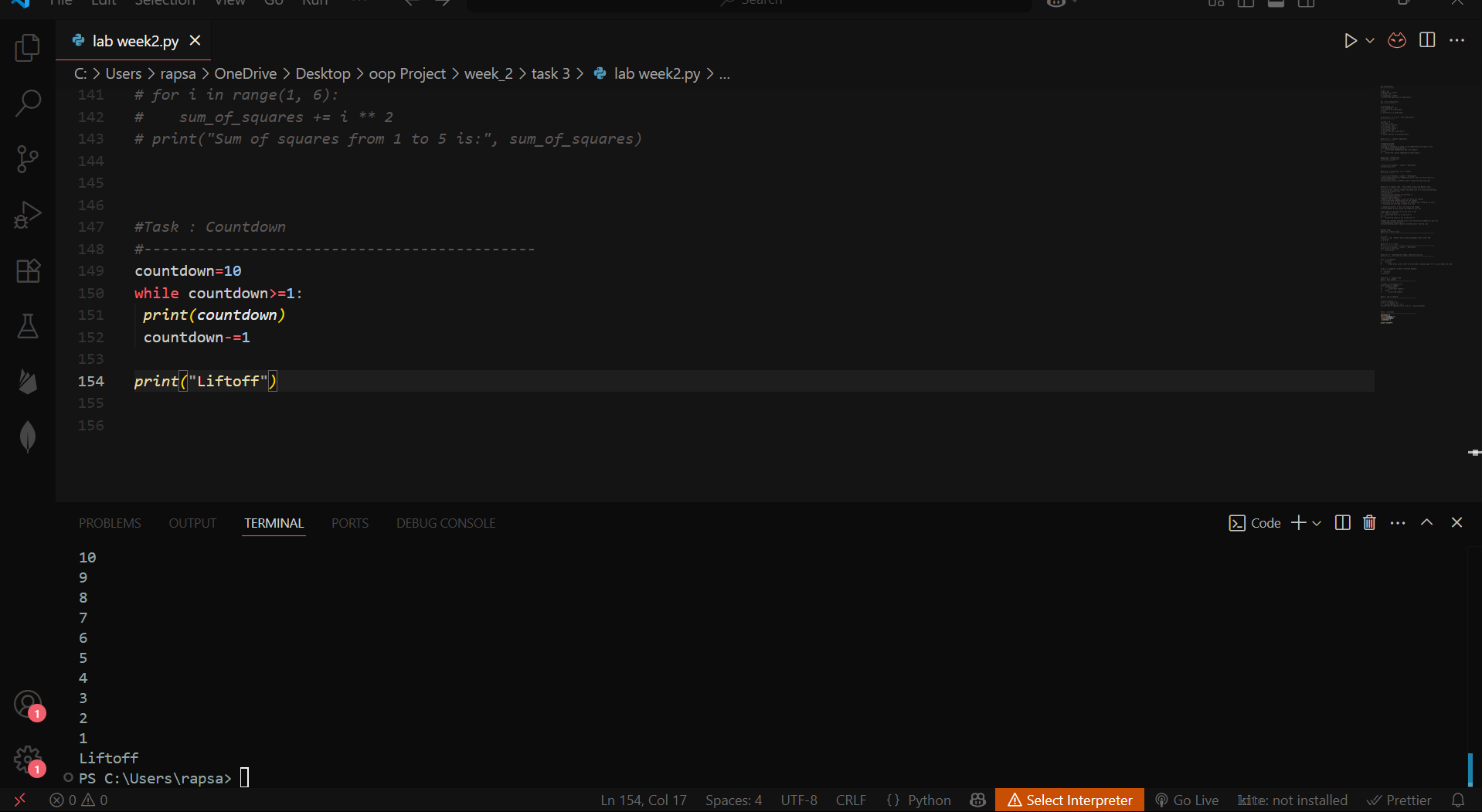
**Explanation:**  
Shows mutable nature of lists with modification where we changed the initial list of colours swapped red colour with blue and expansion along with usage of conditionals.

**3. Loops**

**Task 6: While Loop**

**Objective:** Using while loop and implementing countdown  
**Code:**

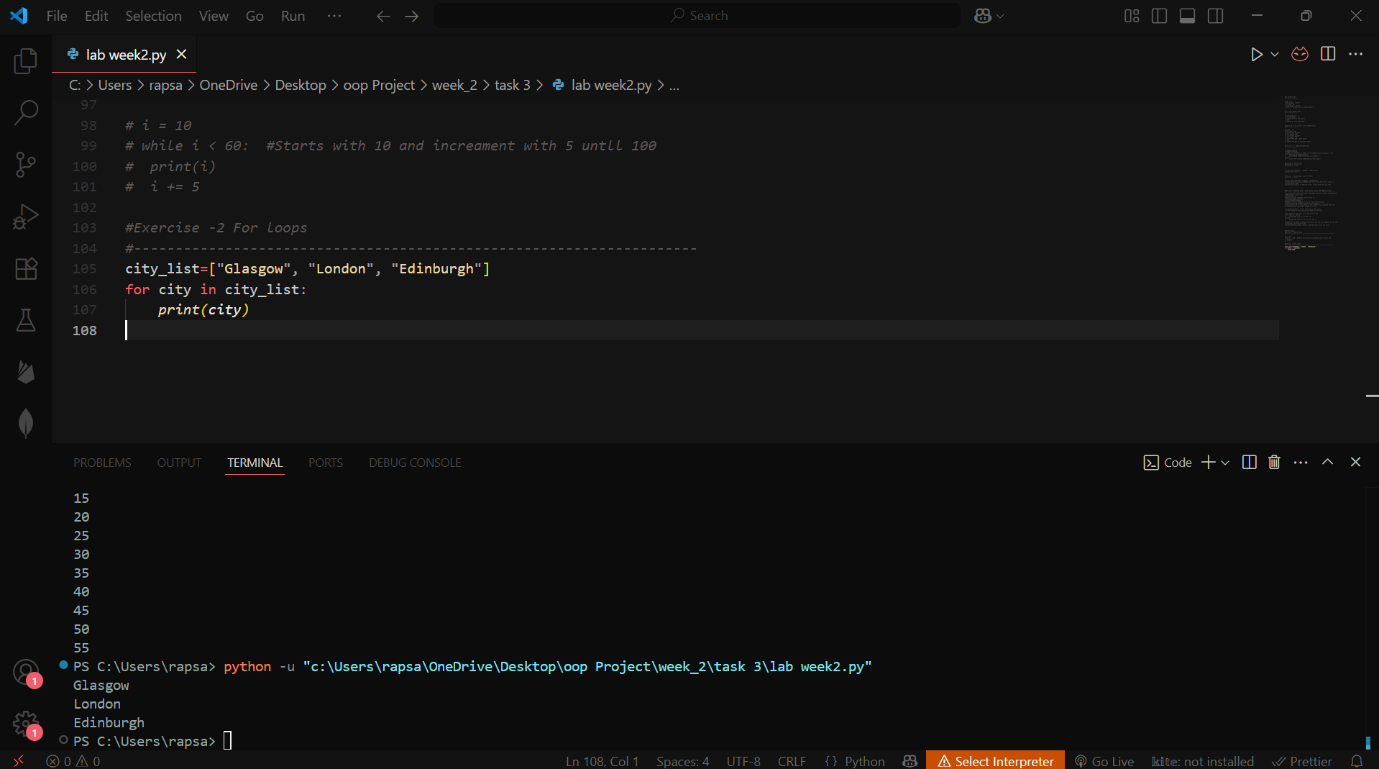
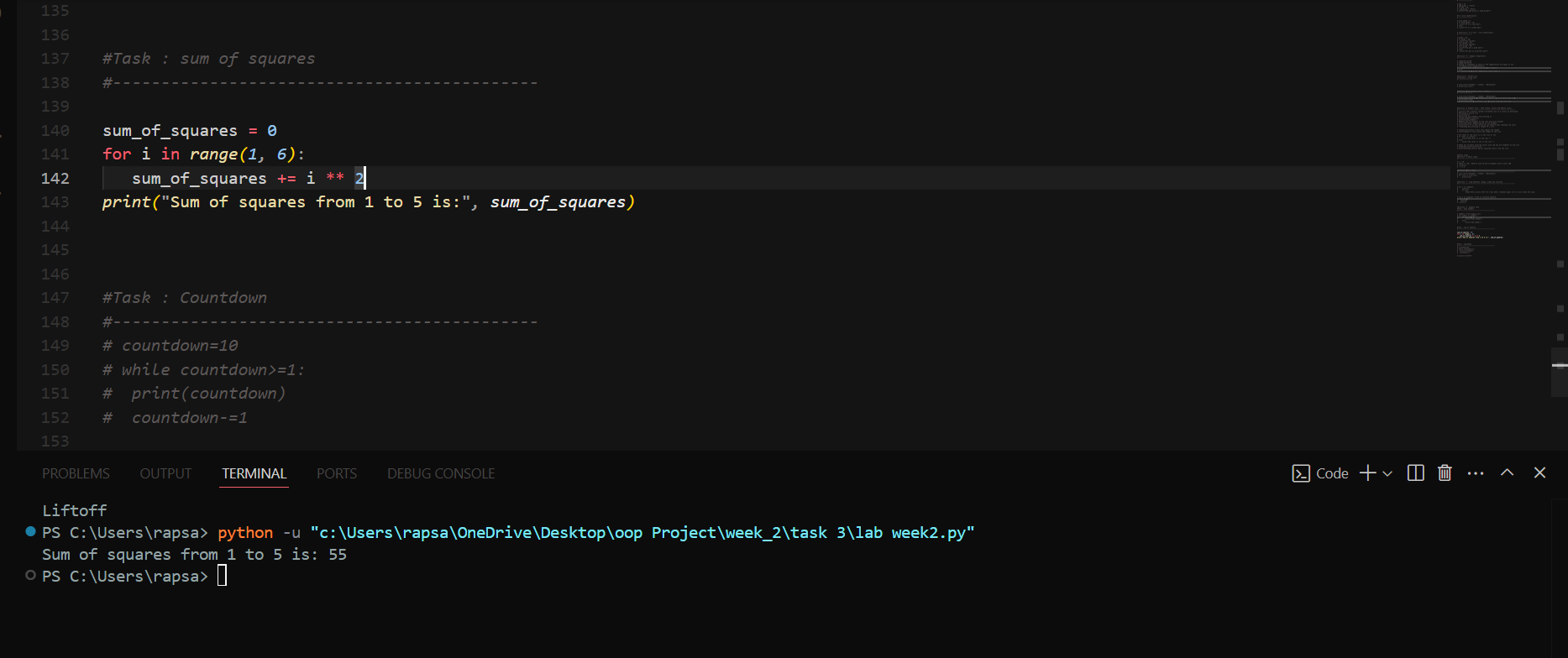




**Explanation:**  
Loop continues and value increases with 5 until it matches 60. The second program is for countdown and value decreases from 10 and when comes down to 1 will exit the while loop and prints the “Liftoff”.

**Task 7: For Loop with Range**

**Objective:** Introduction to For Loop and Range   
**Code:**

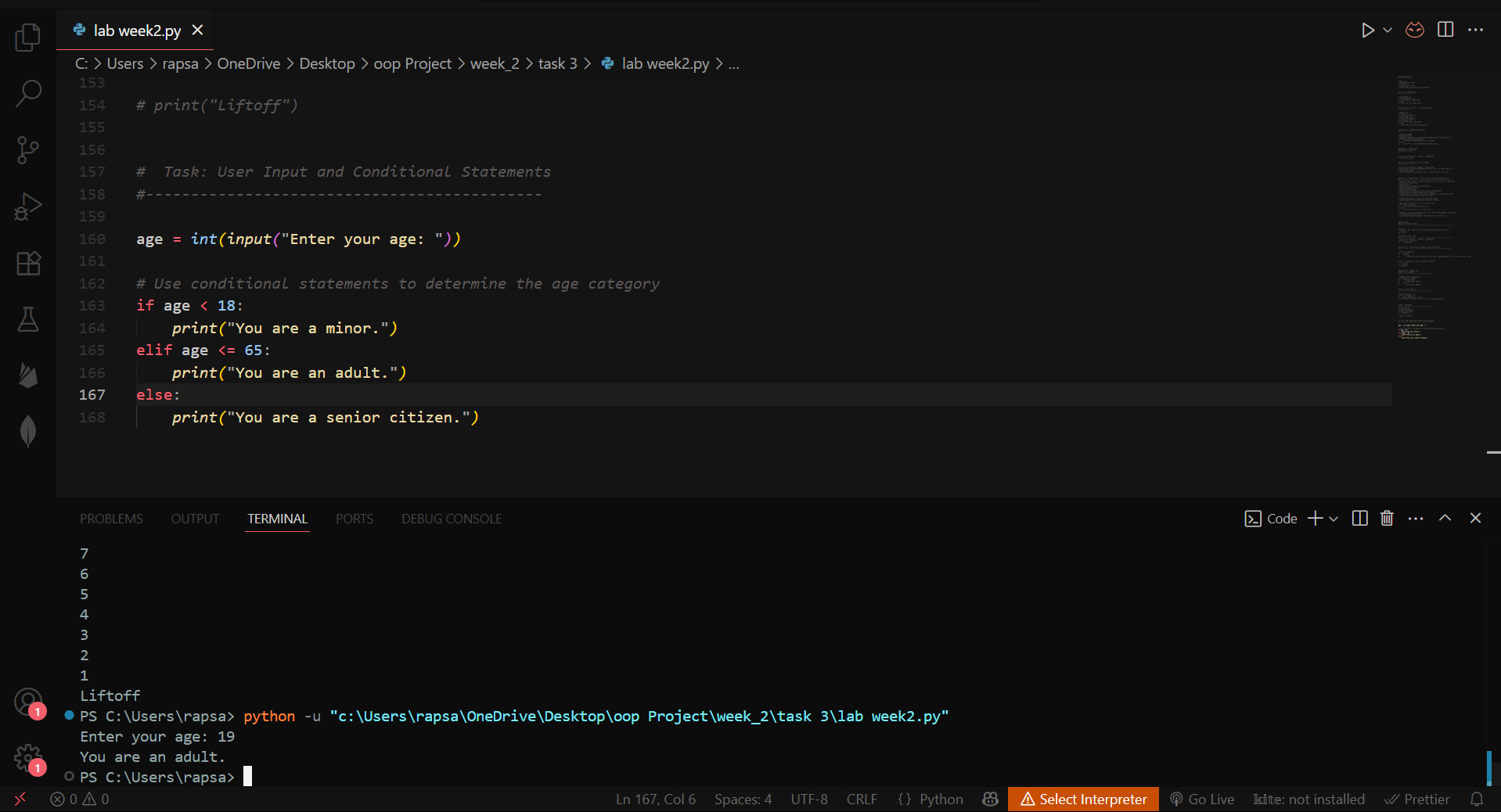
   
**Explanation:**

Access the elements of the list using for loop and print each elements and in second  
accumulates the sum of 1² + 2² + 3² + 4² + 5² using for loop. Range (1, 6) will exclude 6 and will only count up to 5.

**4. User Input**

**Task 8: Age Classifier**

**Objective:** Process user input with conditionals  
**Code:**

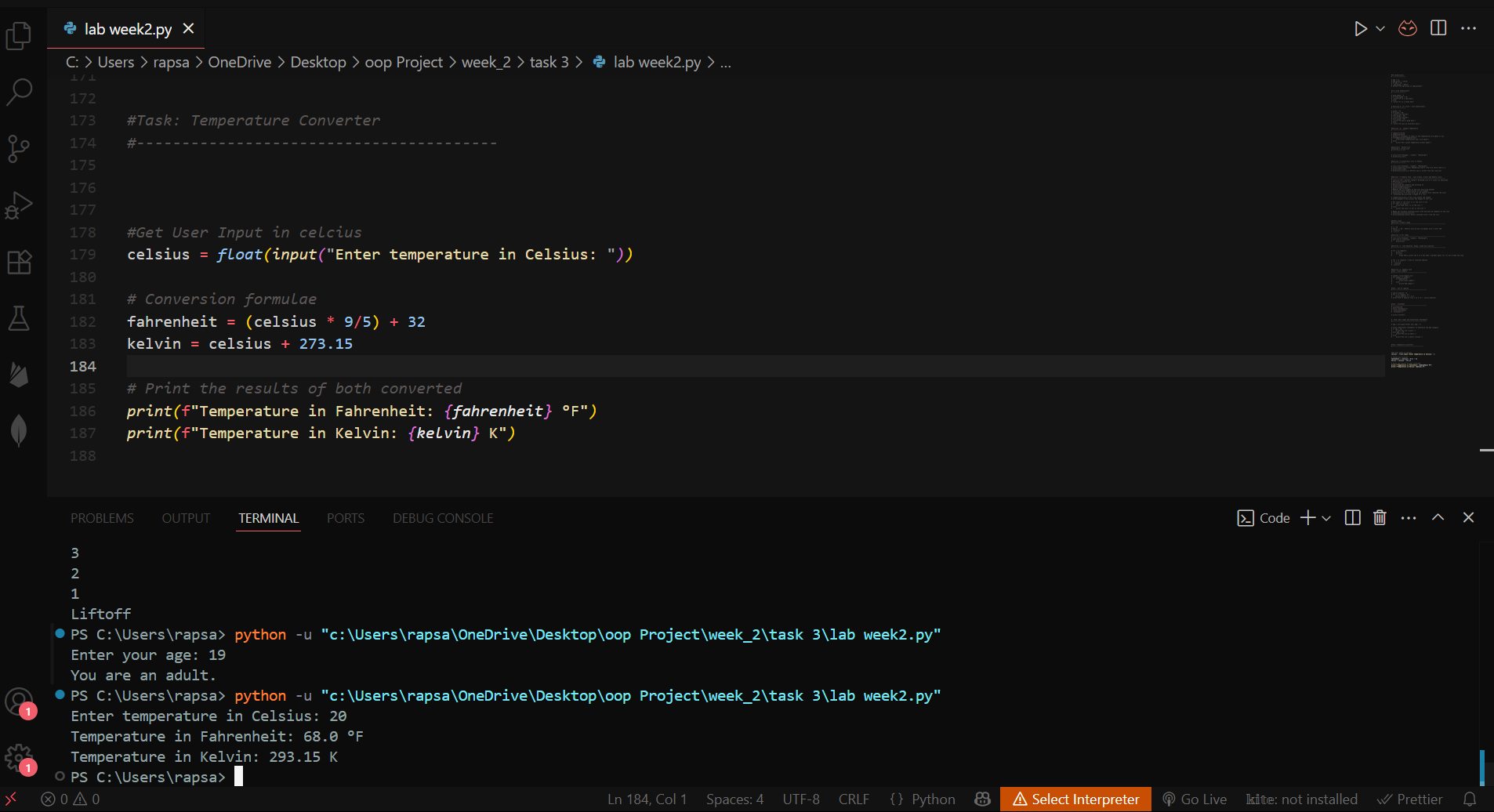


**Explanation:**

Explored the Input and used conditions to take user input and processed the age according to the category we checked with if conditionals.

**Task 9: Temperature Converter**

**Objective:** Convert user-input temperature  
**Code:**



**Explanation:**

Took input from the user using “Input” in Celsius and converted the user input temperature to Fahrenheit and kelvin using respective formulae.

**Conclusion**

This lab session covered fundamental Python concepts including:

* Conditional statements (if/elif/else)
* List operations (creation, access, modification)
* Loop structures (while/for)
* User input processing

All tasks were successfully implemented in the lab file attached demonstrating proficiency in basic Python programming constructs. The temperature converter shows practical application of mathematical operations with user input and other summarised tasks are reported in the file above.

**Files:**

* lab\_week2.py (Introduction and complete implementation of all minor and major tasks along with various exercises )
* Different screenshots in task1 and task2 folder from W3 school as per task instruction.

# Week 3 – Functions, Scope, and Error Handling

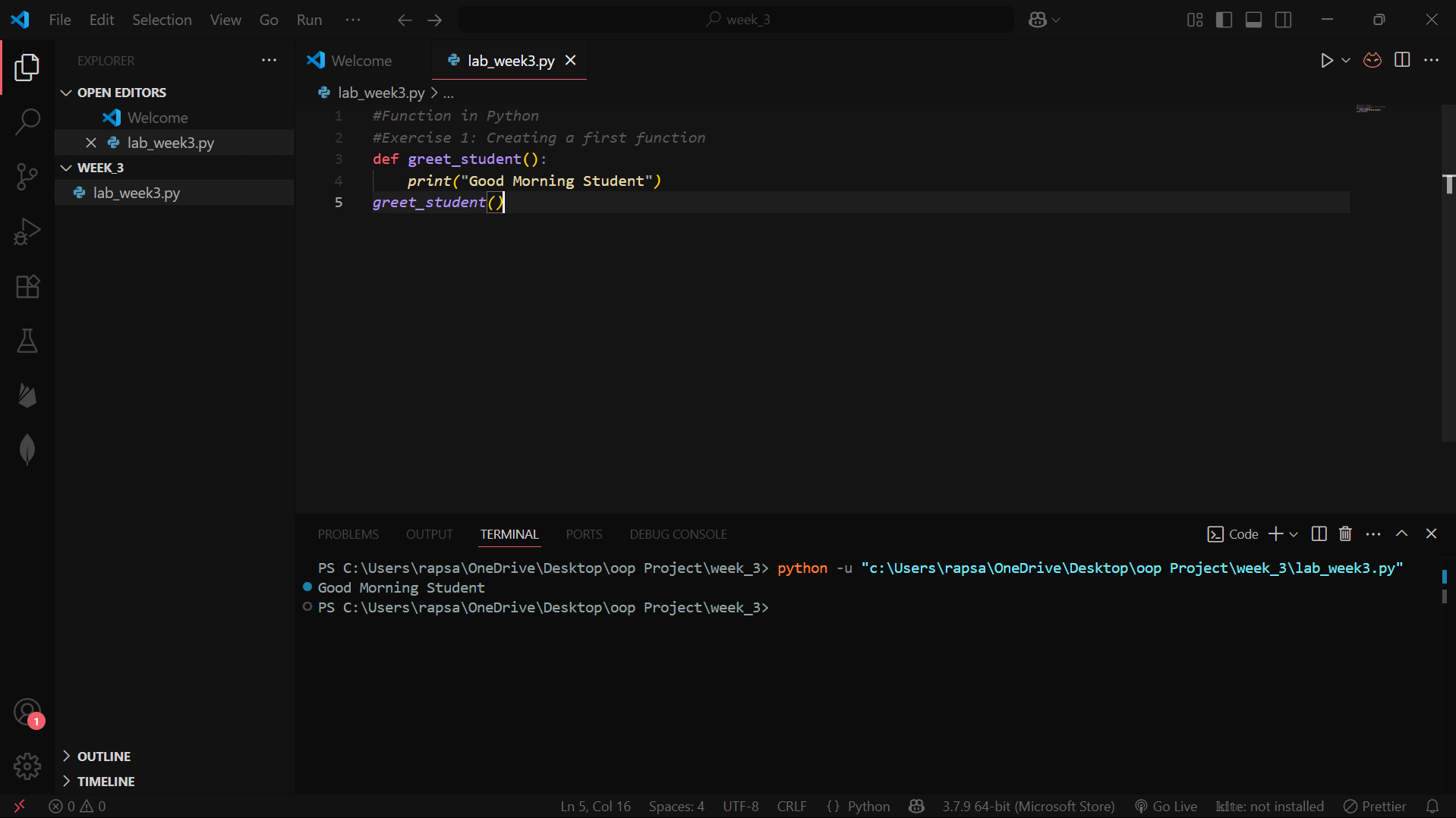
**3.1 Objectives**

* Implement reusable code using functions
* Understand variable scope and lifetime
* Handle common Python errors
* Develop a functional To-Do List application

**3.2 Implementation Details**

**Task 1: Basic Function Creation**

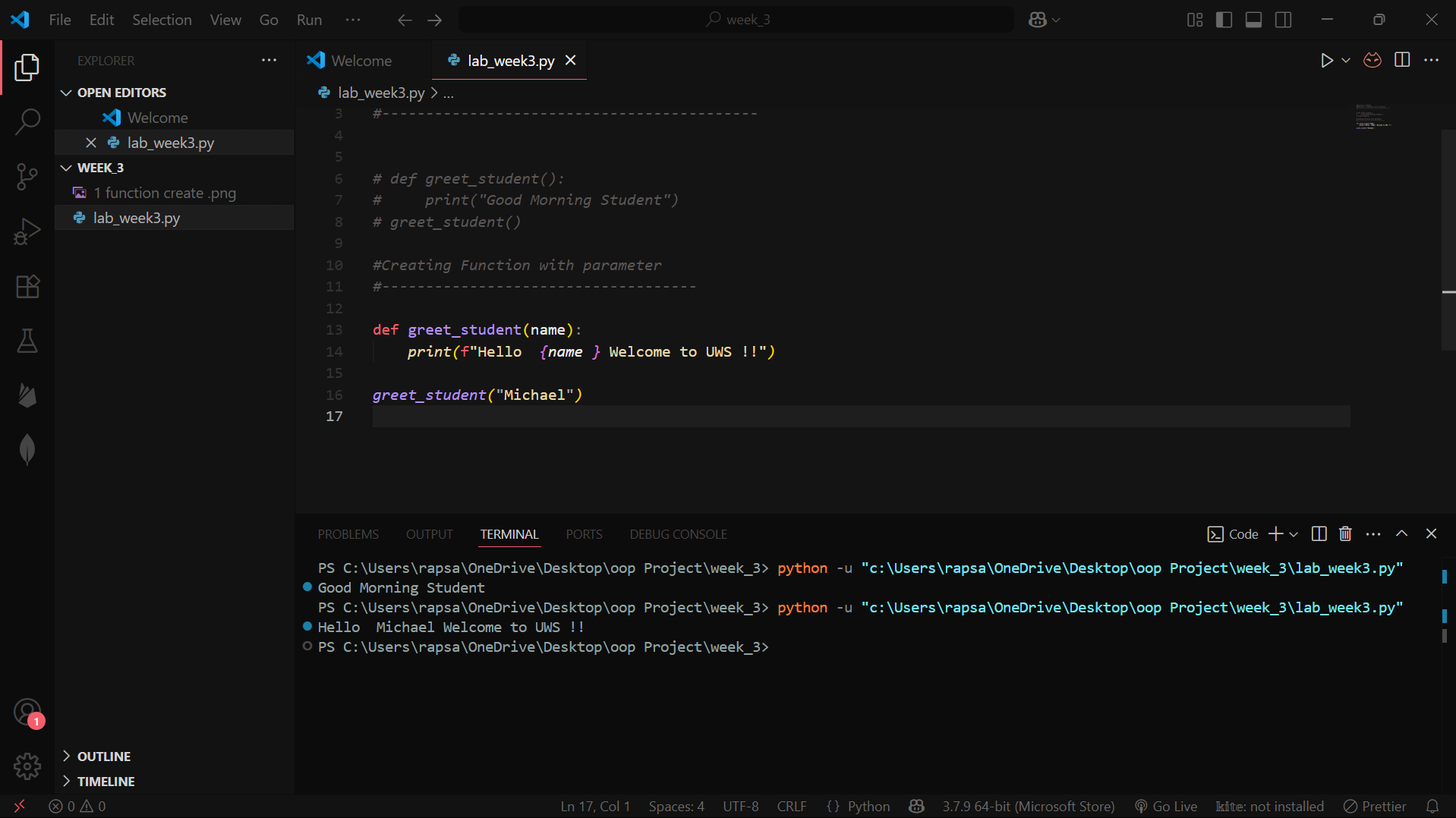
**Objective:** Demonstrate function definition and calling   
**Code:**



**Explanation:**  
The function greet\_student() prints a greeting when called, showing basic function structure.

**Task 2: Parameterized Functions**

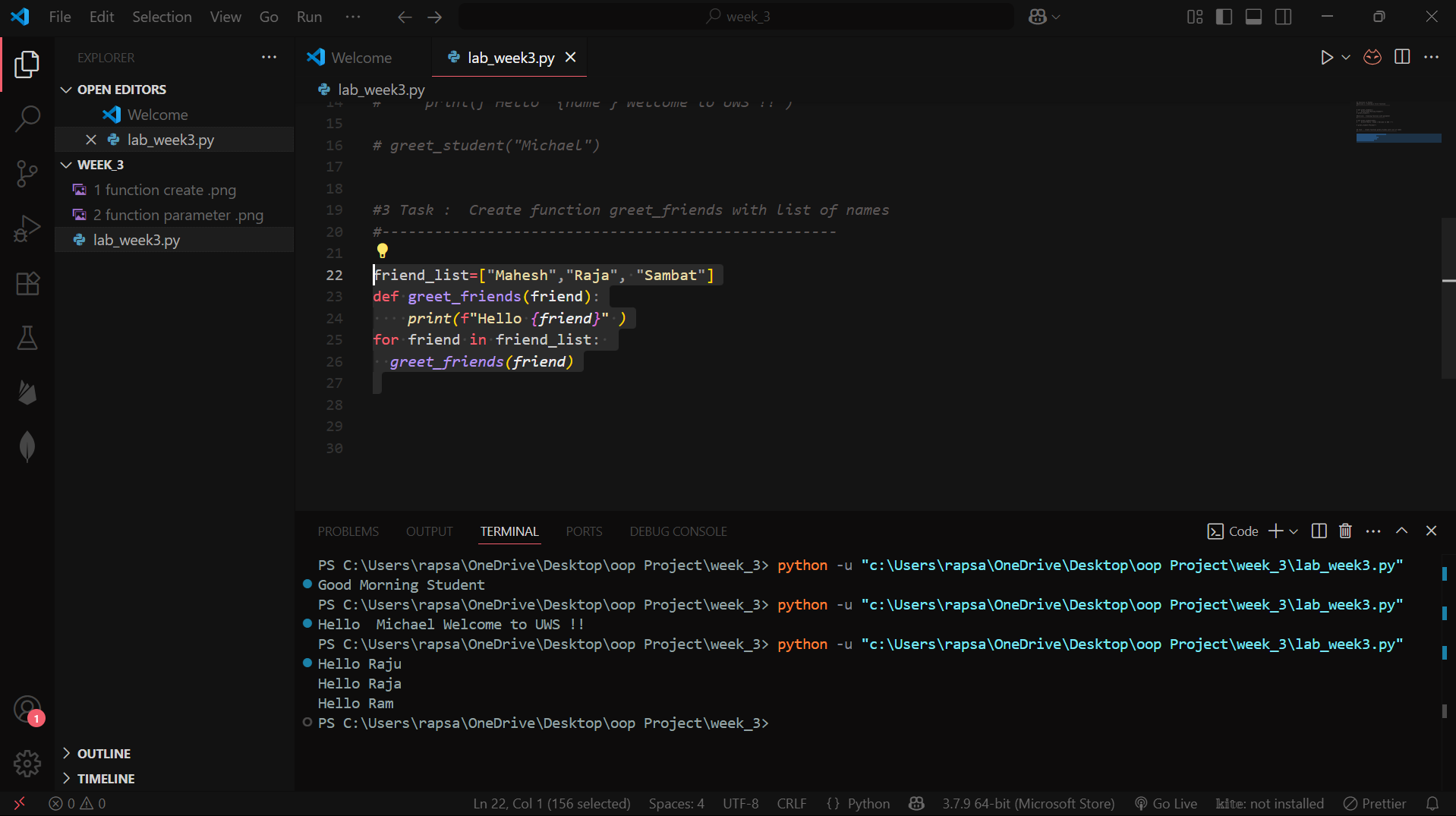
**Objective:** Pass arguments to functions  
**Code:**



**Explanation:**  
The function accepts a name parameter, demonstrating argument passing.

**Task 3: List Processing with Functions**

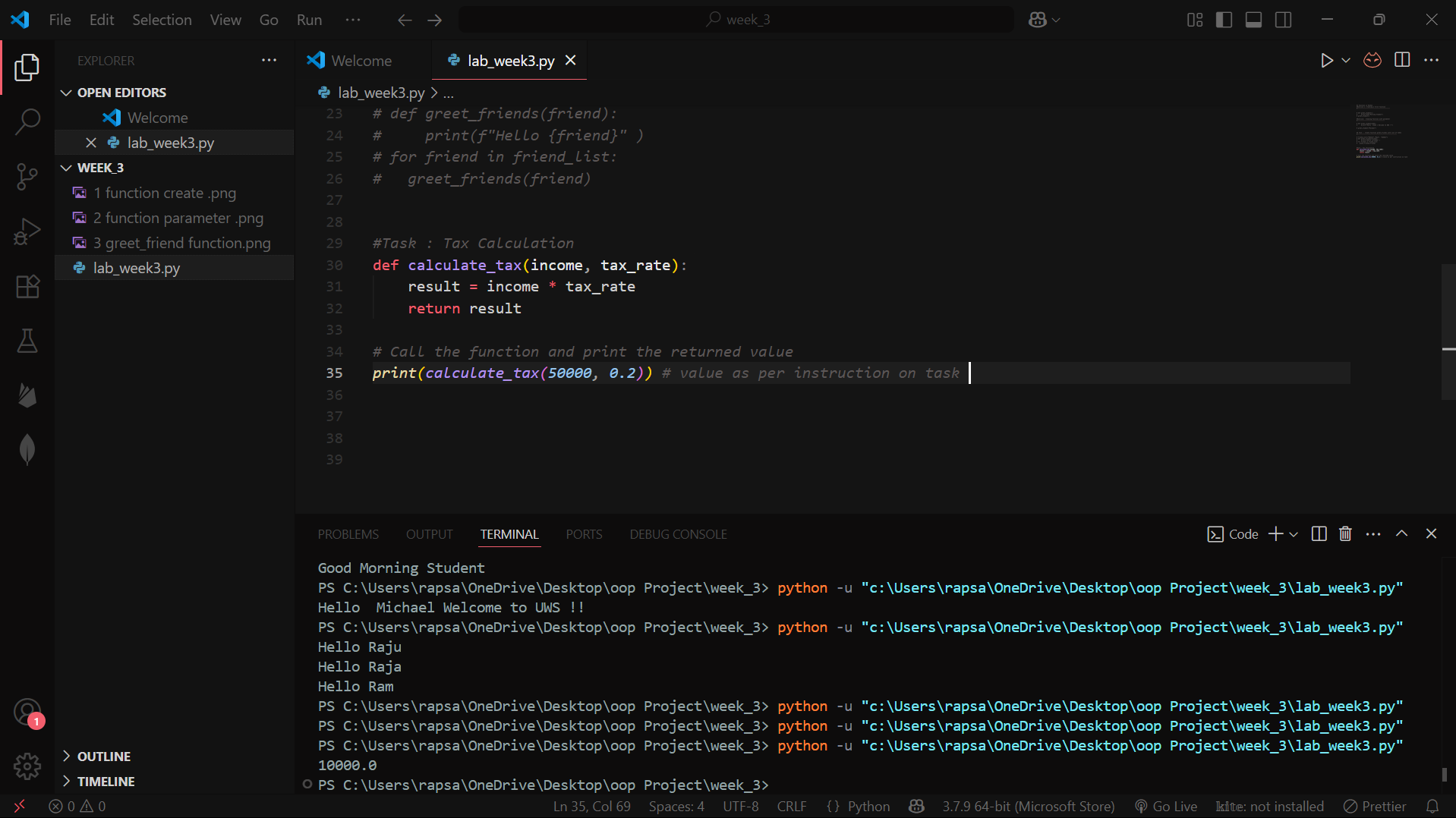
**Objective:** Iterate through lists in functions  
**Code:**



**Explanation:**  
The function processes each item in friend\_list, showing list iteration within functions greeting each name with “Hello” + {name}.

**Task 4: Tax Calculator Function**

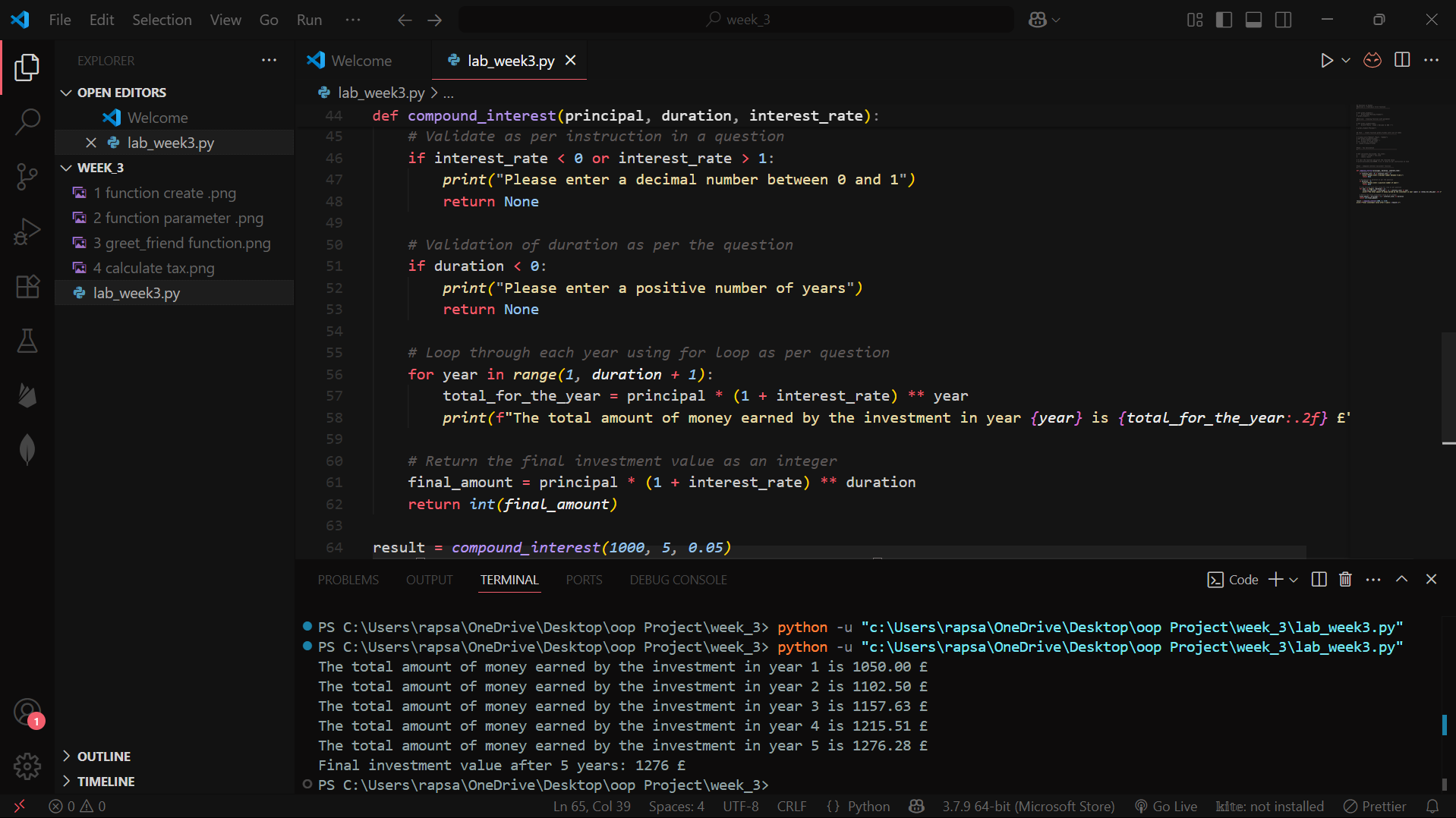
**Objective:** Return values from functions  
**Code:**



**Explanation:**  
The function calculates and returns tax amount based on input parameters when calling the function. We can also take input from the user to run more controlled and desirable calculations.

**Task 5: Compound Interest Calculator**

**Objective:** Implement validation and loops in functions  
**Code:**



**Explanation:**  
The function validates inputs, calculates yearly growth, and returns the final value.

**Task 6: Error Handling**

**Objective:** Identify and fix common errors

| **Error Type** | **Buggy Code** | **Corrected Code** |
| --- | --- | --- |
| Syntax | pritn("Hello") | print("Hello") |
| Name | print(favorite\_color) | Define favorite\_color first |
| Value | int("text") | Convert numeric strings only |
| Index | fruits[3] (list has 3 items) | Use valid indices (0-2) |
| Indentation | Missing indent after if | Properly indent conditional blocks |

**Key Learning:**

* Syntax errors violate Python's grammar rules
* Runtime errors occur during execution (e.g., invalid operations)
* Logical errors produce incorrect results without crashing

**Task 7: To-Do List Application**

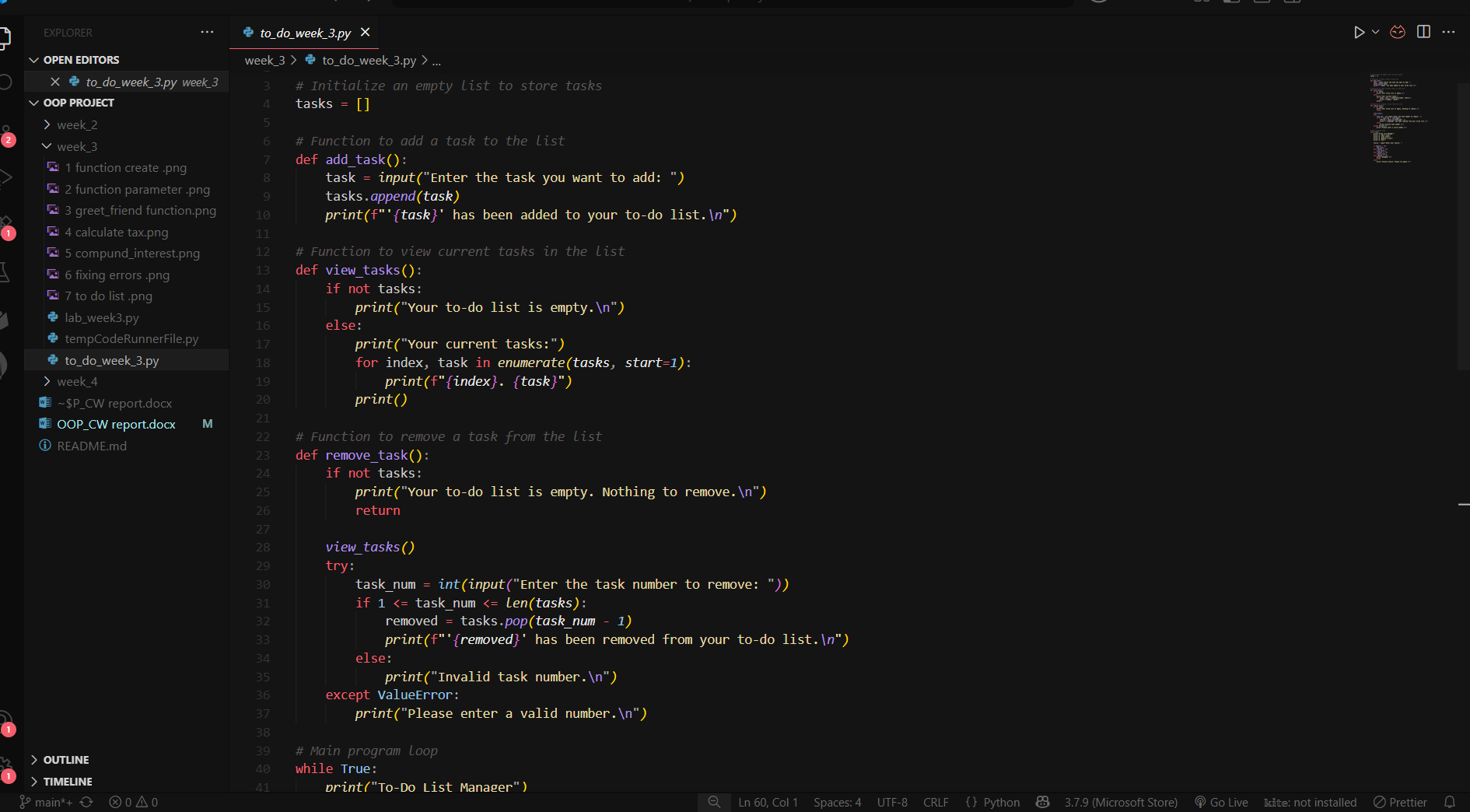
**Objective:** Build an interactive program

**To-Do List Application Report**

**1. Key Features**

* **Task Management:**
  + Add new tasks
  + View all tasks (numbered list)
  + Remove tasks by number
* **User Interface:**
  + Menu-driven console interface
  + Clear input prompts
  + Action confirmations

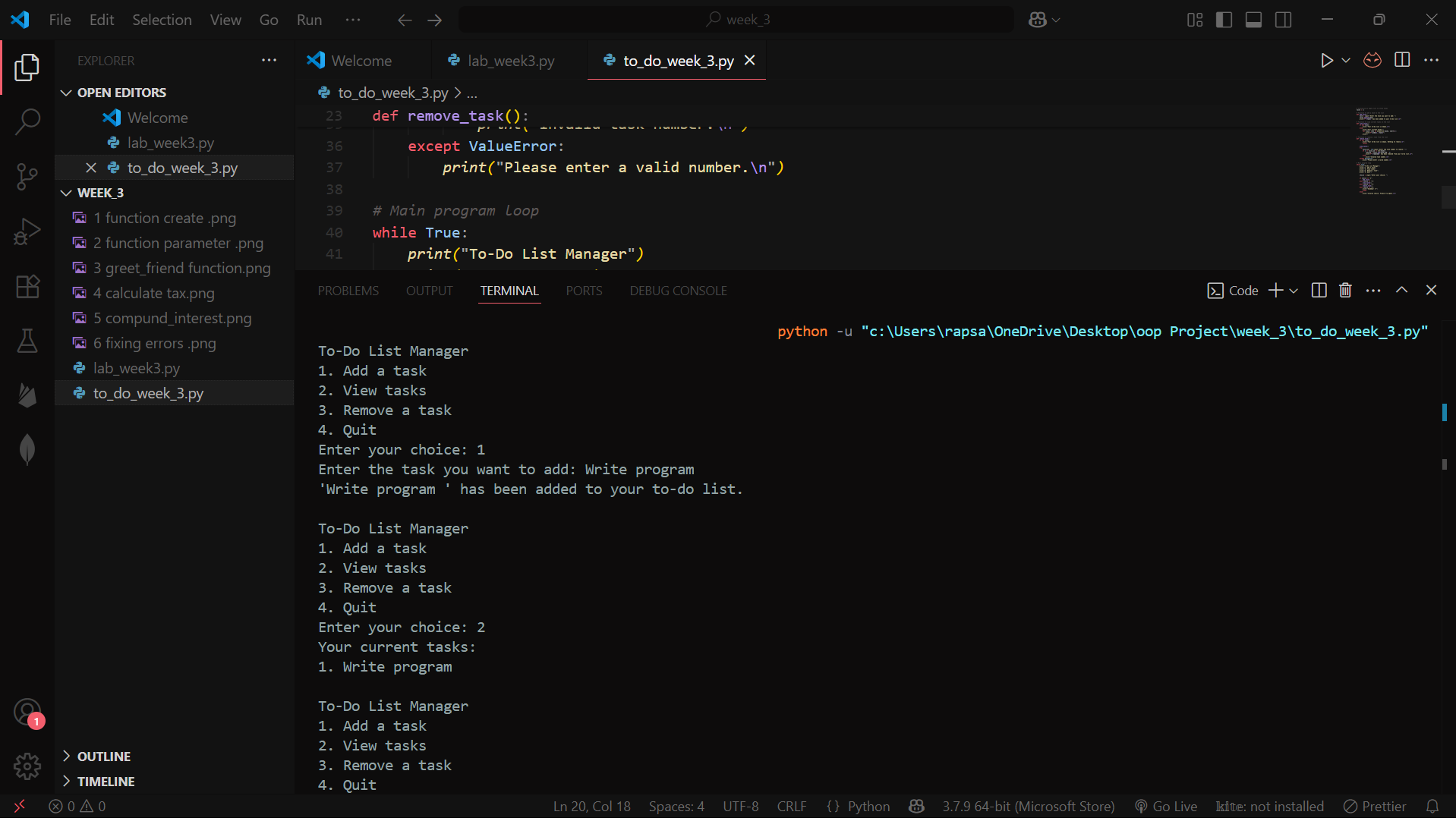
**2. Technical Implementation**



**3. Error Handling**

| **Error Case** | **Solution** |
| --- | --- |
| Empty task list | Conditional check |
| Invalid task number | Range validation |
| Non-numeric input | try-except block |

**4. Sample Output**

**5. Achievements**

* Full CRUD functionality
* Input validation
* Clear user feedback
* Modular code structure

**Code File:** to\_do\_week\_3.py

## Conclusion

This week covered **functions, error handling, and practical Python programming**. Key achievements:

* **Functions:** Created reusable code with parameters and return values (tax calculator, compound interest).
* **Error Handling:** Fixed syntax, index, and value errors using try-except and validation checks.
* **To-Do List App:** Built an interactive task manager with CRUD operations, demonstrating **loops, lists, and functions**.

**Outcome:** Strengthened core Python skills, ready for **OOP in Week 4**.

**Files:**

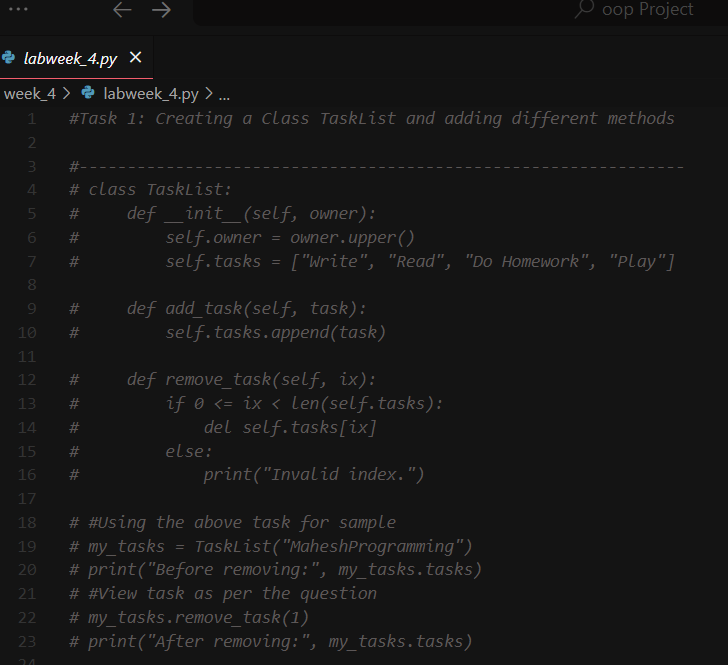
* lab\_week\_3.py (Functions & Errors. Complete implementation of all minor and major tasks along with various exercises )
* to\_do\_week\_3.py (To-Do List App)

# Week 4 Tasks Report

**Task 1: Creating Classes and Objects**

**Objective:**  
Convert procedural To-Do list to OOP implementation using classes and objects.

**Implementation:**

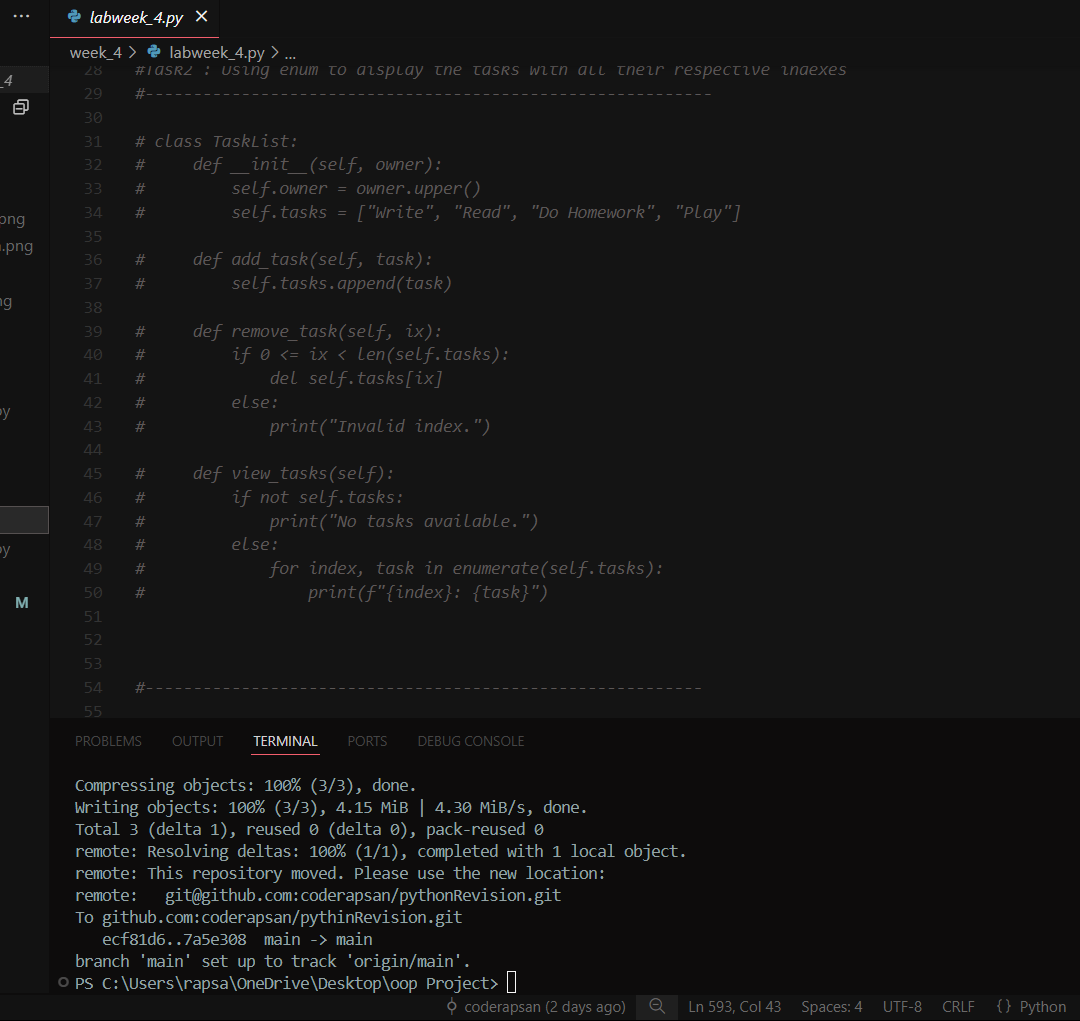
**What We Learned:**

* Class definition syntax using class keyword
* The purpose of \_\_init\_\_ as constructor
* Instance attribute creation
* Object instantiation (my\_list = TaskList("User"))

**Task 2: Adding Methods to Classes**

**Objective:**  
Implement task management methods in TaskList class.

**Implementation:**



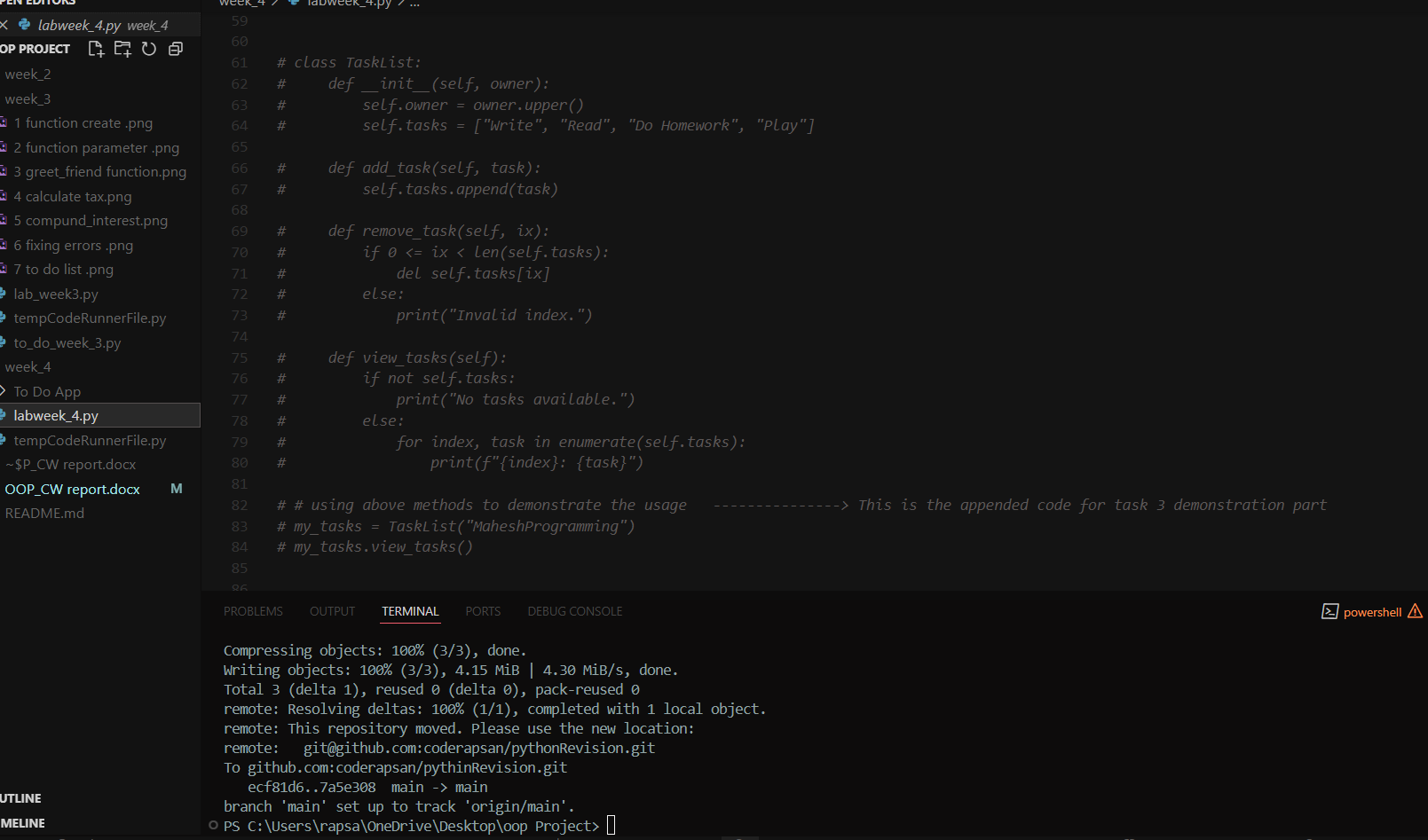
**What We Learned:**

* Instance methods require self parameter
* List manipulation within classes
* Using enumerate() for indexed iteration
* Method calling syntax (my\_list.add\_task(task))

**Task 3: Composition with Task Objects**

**Objective:**  
Store Task objects instead of strings in TaskList.

**Implementation:**

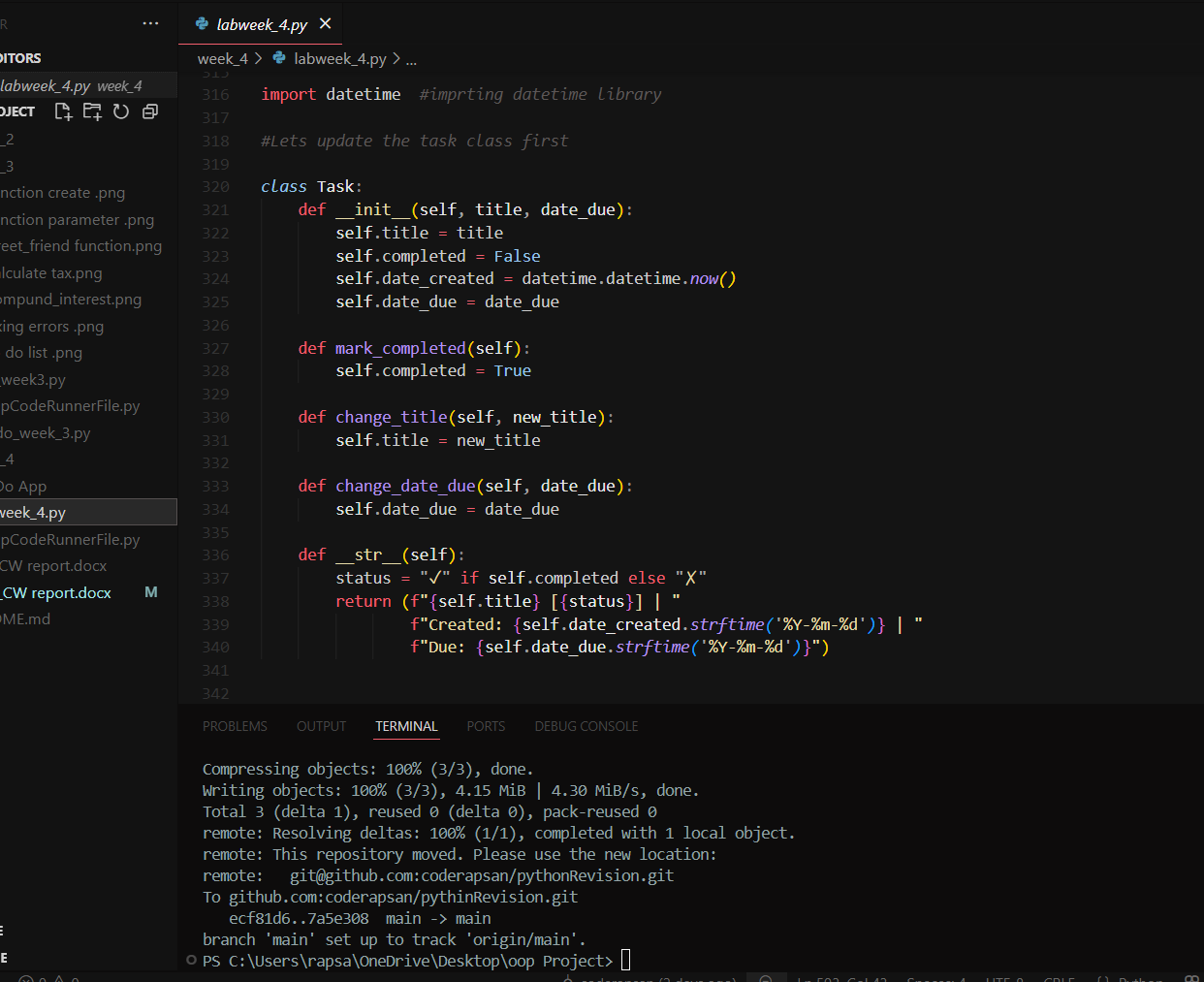
**What We Learned:**

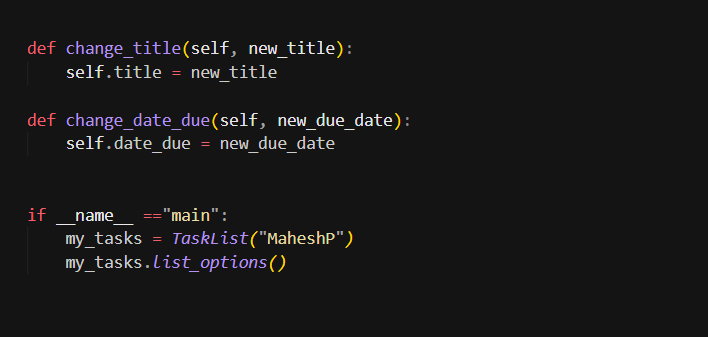
* Object composition pattern
* Difference between objects and primitive types
* Importance of \_\_str\_\_ for object representation

**Task 4: Adding Dates with datetime**

**Objective:**  
Incorporate due dates using datetime module.

**Implementation:**



**What We Learned:**

* Importing and using standard library modules
* Date parsing with strptime()
* Storing datetime objects as attributes
* Using classes to handle the execution

**Task 5: Modularizing Code**

**Objective:**  
Split code into multiple files.

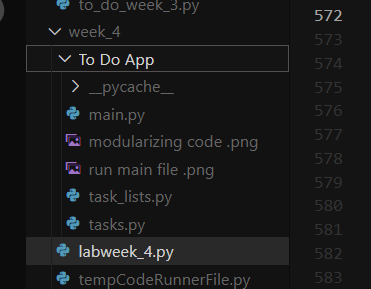
**Implementation:**

ToDoApp/

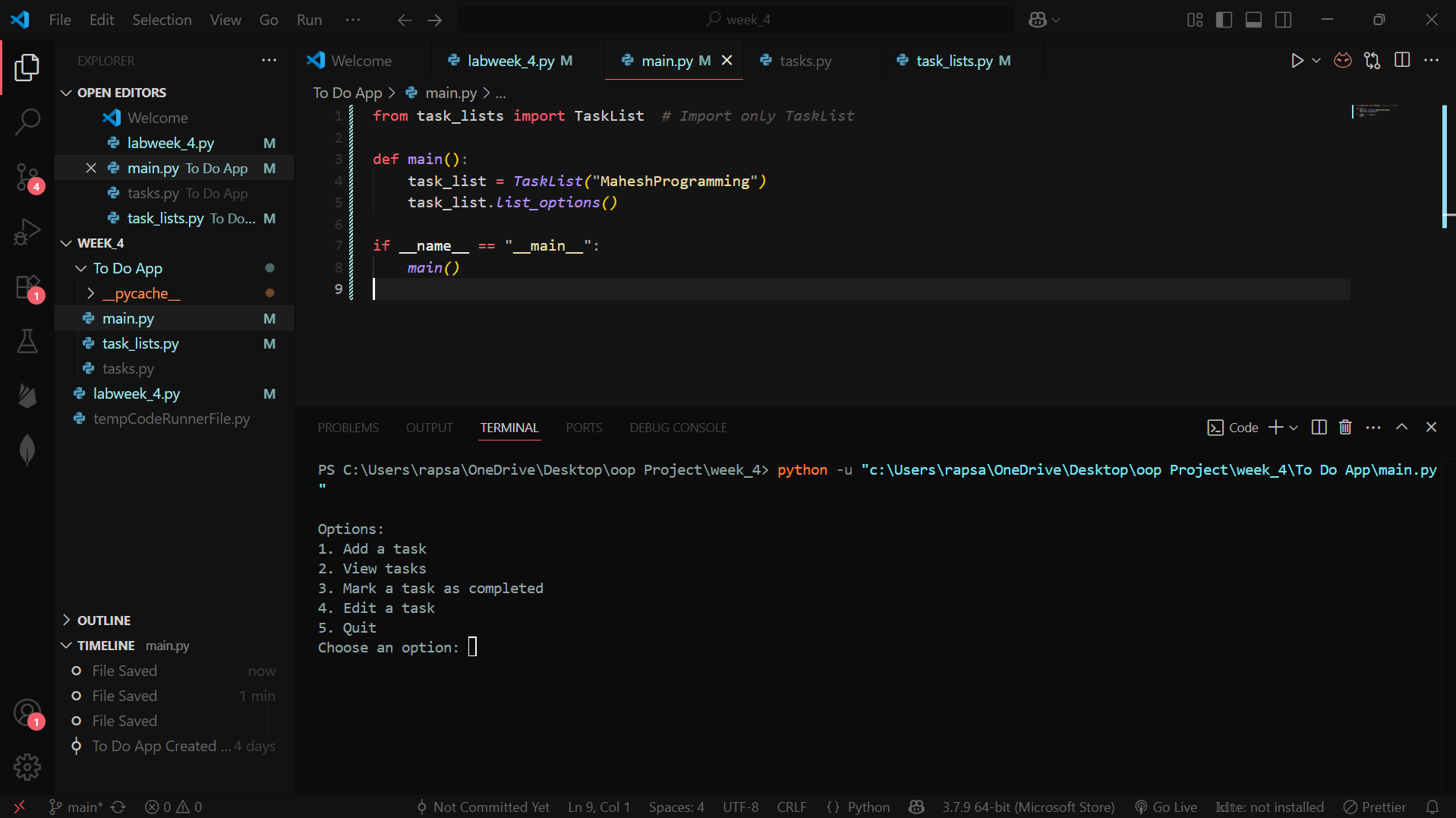
├── main.py

├── task.py

└── task\_list.py







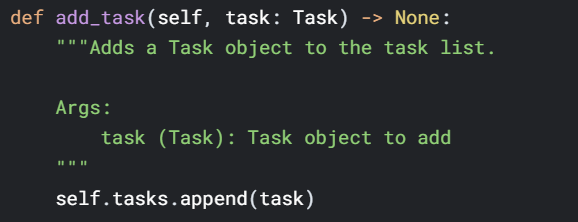
**What We Learned:**

* Python module system
* Import statements (from task import Task)
* if \_\_name\_\_ == "\_\_main\_\_" pattern
* Benefits of separation of concerns

**Task 6: Type Hints and Documentation**

**Objective:**  
Add type annotations and docstrings.

**Implementation:**

**What We Learned:**

* Type hint syntax (variable: type)
* Return type annotations (-> type)
* Docstring conventions (PEP 257)
* Using mypy for static type checking

**Portfolio Exercises**

**1. Task Description Feature**  
Added optional description field with edit capability.

**2. Overdue Tasks Method**  
Implemented date comparison to identify overdue tasks.

**Key Learnings:**

* Extending class functionality
* Working with optional parameters
* Date comparisons in Python
* Maintaining backward compatibility

## Conclusion

**Key Achievements**

1. Successfully converted procedural code to OOP architecture
2. Implemented core OOP concepts:
   * Encapsulation (data + methods in classes)
   * Composition (Task objects in TaskList)
3. Established modular project structure
4. Added professional documentation standards

**Skills Developed**

1. Class design and implementation
2. Object composition patterns
3. Python module system usage
4. Type checking with mypy

* Documentation best practices lab\_week\_4.py (Complete implementation of all major and minor tasks)

**Attachments:**

* ToDoApp/ (Modularized version)
* Program execution screenshots