Lab Assignment 5.1 and 6

Task 1:

Employee Data: Create Python code that defines a class named `Employee` with the following attributes: `empid`, `empname`, `designation`, `basic_salary`, and `exp`. Implement a method `display_details()` to print all employee details. Implement another method `calculate_allowance()` to determine additional allowance based on experience:

- If `exp > 10 years` → allowance = 20% of `basic_salary`
- If `5 ≤ exp ≤ 10 years` → allowance = 10% of `basic_salary`
- If `exp < 5 years` → allowance = 5% of `basic_salary`

Finally, create at least one instance of the `Employee` class, call the `display_details()` method, and print the calculated allowance.

Task 2:

Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit
- 101 to 300 units → ₹7 per unit
- More than 300 units → ₹10 per unit

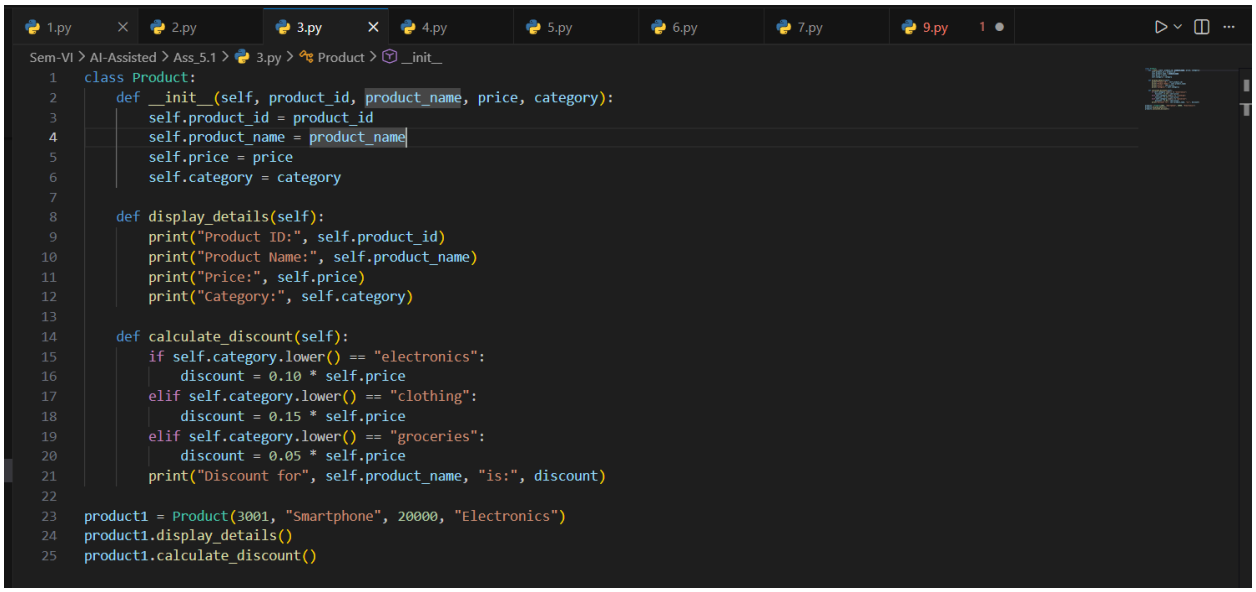
Create a bill object, display details, and print the total bill amount.

Task 3:

Product Discount Calculation- Create Python code that defines a class named `Product` with attributes: `product_id`, `product_name`, `price`, and `category`. Implement a method `display_details()` to print product details. Implement another method `calculate_discount()` where:

- Electronics → 10% discount
- Clothing → 15% discount
- Grocery → 5% discount

Create at least one product object, display details, and print the final price after discount.



Task 4:

Book Late Fee Calculation- Create Python code that defines a class named `LibraryBook` with attributes: `book_id`, `title`, `author`, `borrower`, and `days_late`. Implement a method `display_details()` to print book details, and a method `calculate_late_fee()` where:

- Days late ≤ 5 → ₹5 per day
- 6 to 10 days late → ₹7 per day
- More than 10 days late → ₹10 per day

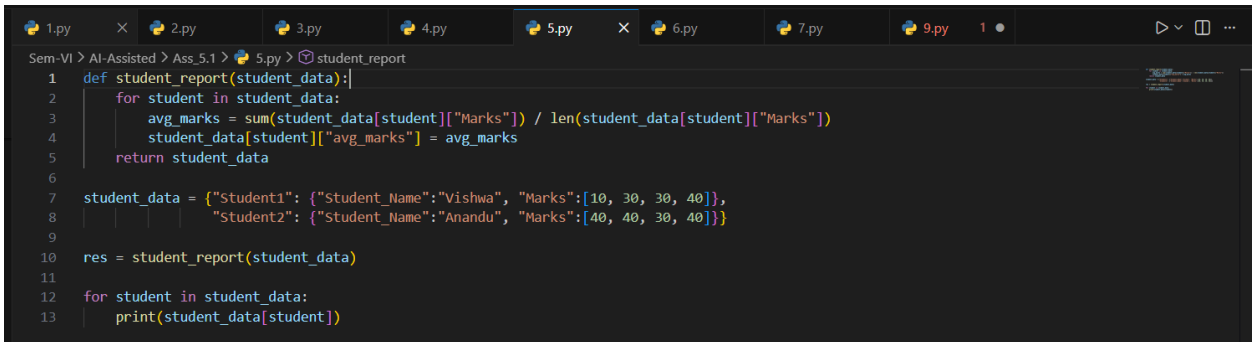
Create a book object, display details, and print the late fee.

Task 5:

Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks. The function should:

- Calculate the average score for each student
- Determine pass/fail status (pass ≥ 40)
- Return a summary report as a list of dictionaries

Use Copilot suggestions as you build the function and format the output.

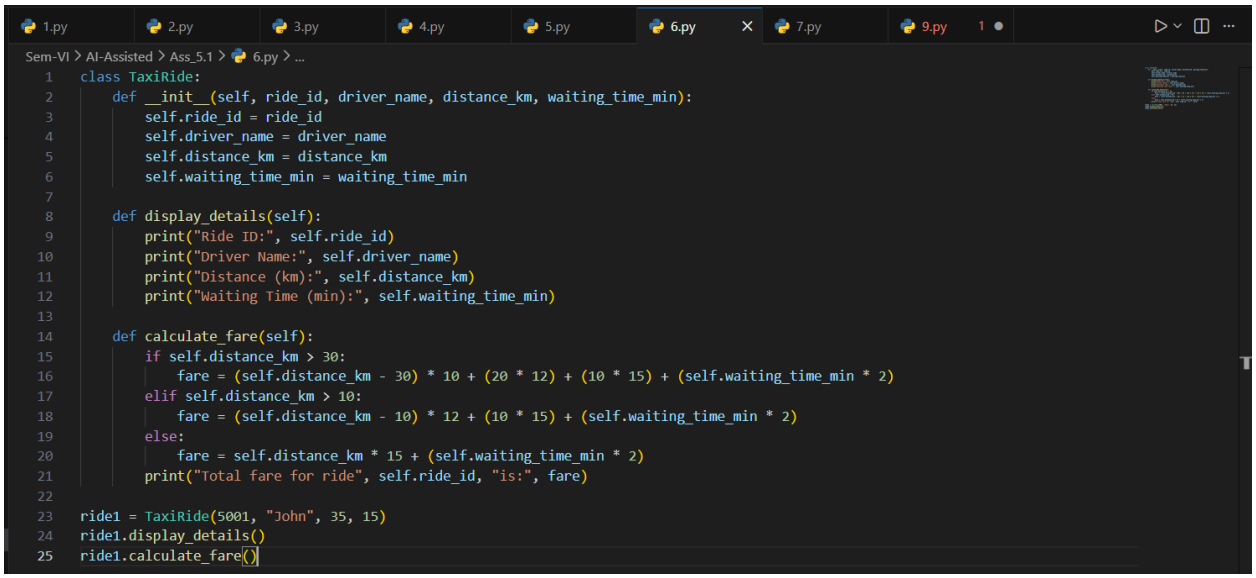


Task 6:

Taxi Fare Calculation-Create Python code that defines a class named `TaxiRide` with attributes: `ride_id`, `driver_name`, `distance_km`, and `waiting_time_min`. Implement a method `display_details()` to print ride details, and a method `calculate_fare()` where:

- ₹15 per km for the first 10 km
- ₹12 per km for the next 20 km
- ₹10 per km above 30 km
- Waiting charge: ₹2 per minute

Create a ride object, display details, and print the total fare.

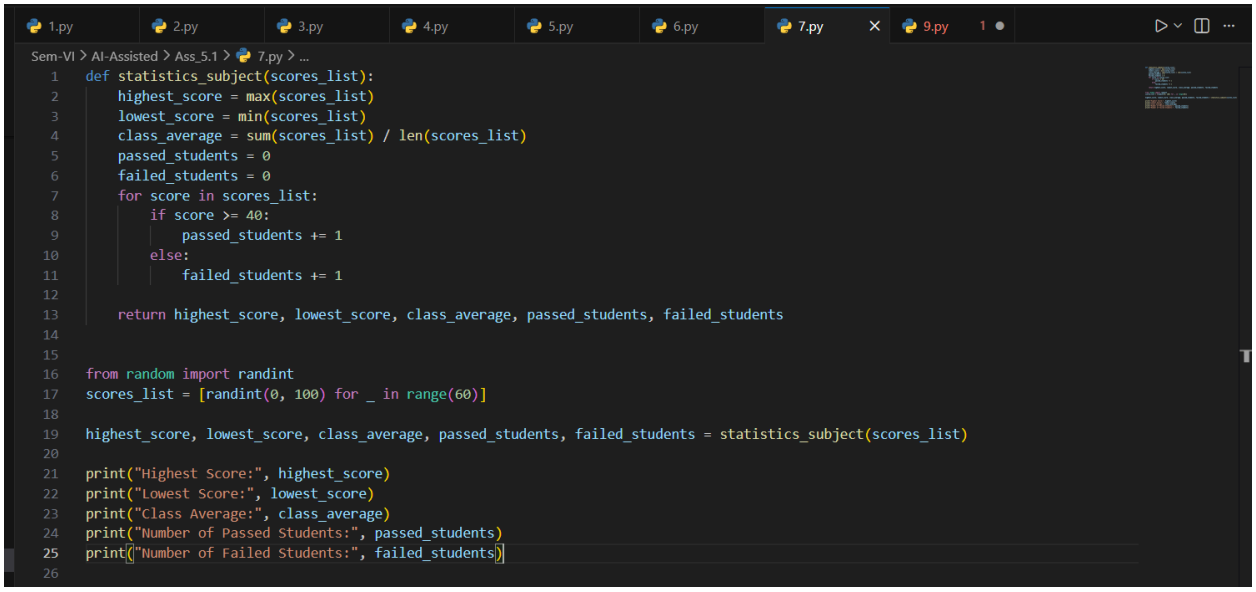


Task 7:

Statistics Subject Performance - Create a Python function `statistics_subject(scores_list)` that accepts a list of 60 student scores and computes key performance statistics. The function should return the following:

- Highest score in the class
- Lowest score in the class
- Class average score
- Number of students passed (score ≥ 40)
- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic



Task 8: (Transparency in Algorithm Optimization)

Description:

Use AI to generate two solutions for checking prime numbers:

Naive approach(basic)

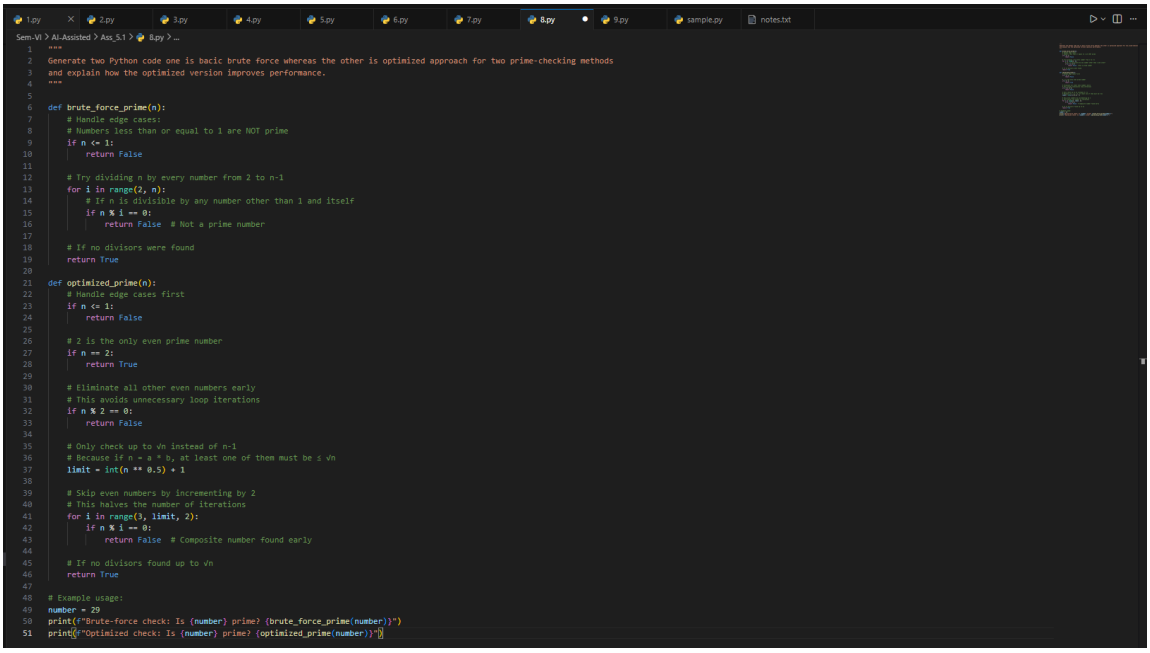
Optimized approach

Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

- Code for both methods.
- Transparent explanation of time complexity.
- Comparison highlighting efficiency improvements.



Task 9: (Transparency in Recursive Algorithms)

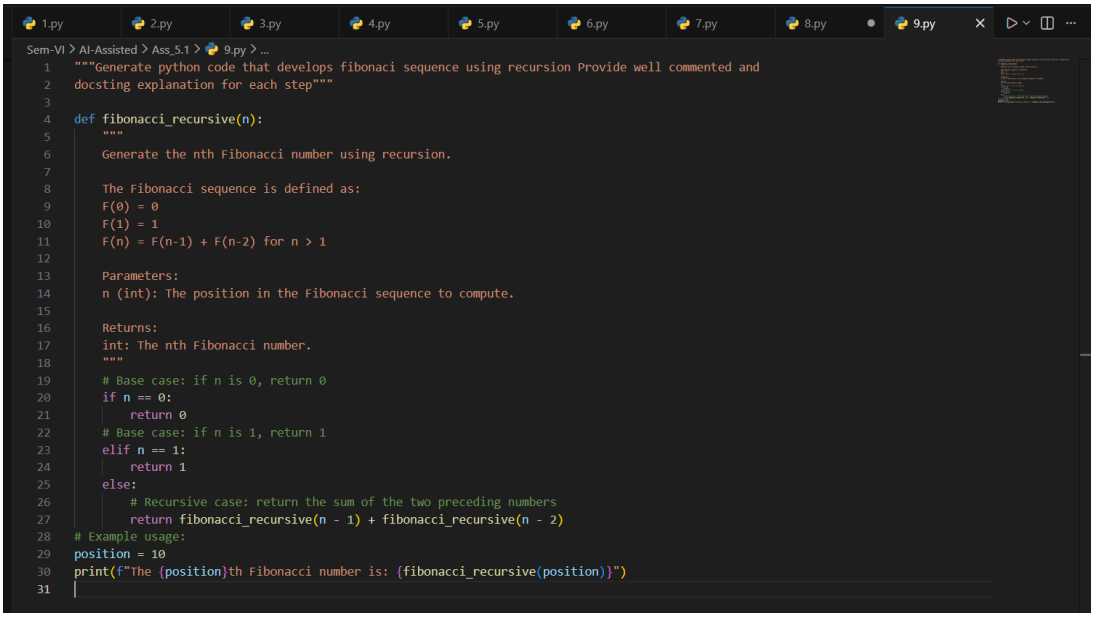
Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.

Expected Output:

- Well-commented recursive code.
- Clear explanation of how recursion works.
- Verification that explanation matches actual execution.



Task 10: (Transparency in Error Handling)

Task:

Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

- Code with meaningful exception handling.
- Clear comments explaining each error scenario.
- Validation that explanations align with runtime behavior.

```
"""
Write a python program that reads a text file and processes the data
Generate code with proper error handling and clear explanations for each exception
"""

try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()

            # Check if file is empty
            if not data:
                print("The file is empty.")
            else:
                # Process the data (for example, count the number of lines)
                lines = data.splitlines()
                print(f"The file contains {len(lines)} lines.")

        except MemoryError:
            # Handle case when file is too large
            print("Error: File is too large to process.")

        except UnicodeDecodeError:
            # Handle encoding issues
            print("Error: Unable to read file due to encoding issues.")

        except Exception as e:
            # Handle any exceptions that occur during data processing
            print(f"An error occurred while processing the data: {e}")

except FileNotFoundError:
    # Handle the case where the file does not exist
    print("Error: The file 'data.txt' was not found.")

except PermissionError:
    # Handle permission issues
    print("Error: You don't have permission to read 'data.txt'.")

except IsADirectoryError:
    # Handle case where 'data.txt' is actually a directory
    print("Error: 'data.txt' is a directory, not a file.")

except IOError:
    # Handle other I/O errors
    print("Error: An I/O error occurred while trying to read the file.")

except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
```