Ai Assisted Coding

Assignment – 5.1 & 6

Name : G.Samshray

Batch : 23

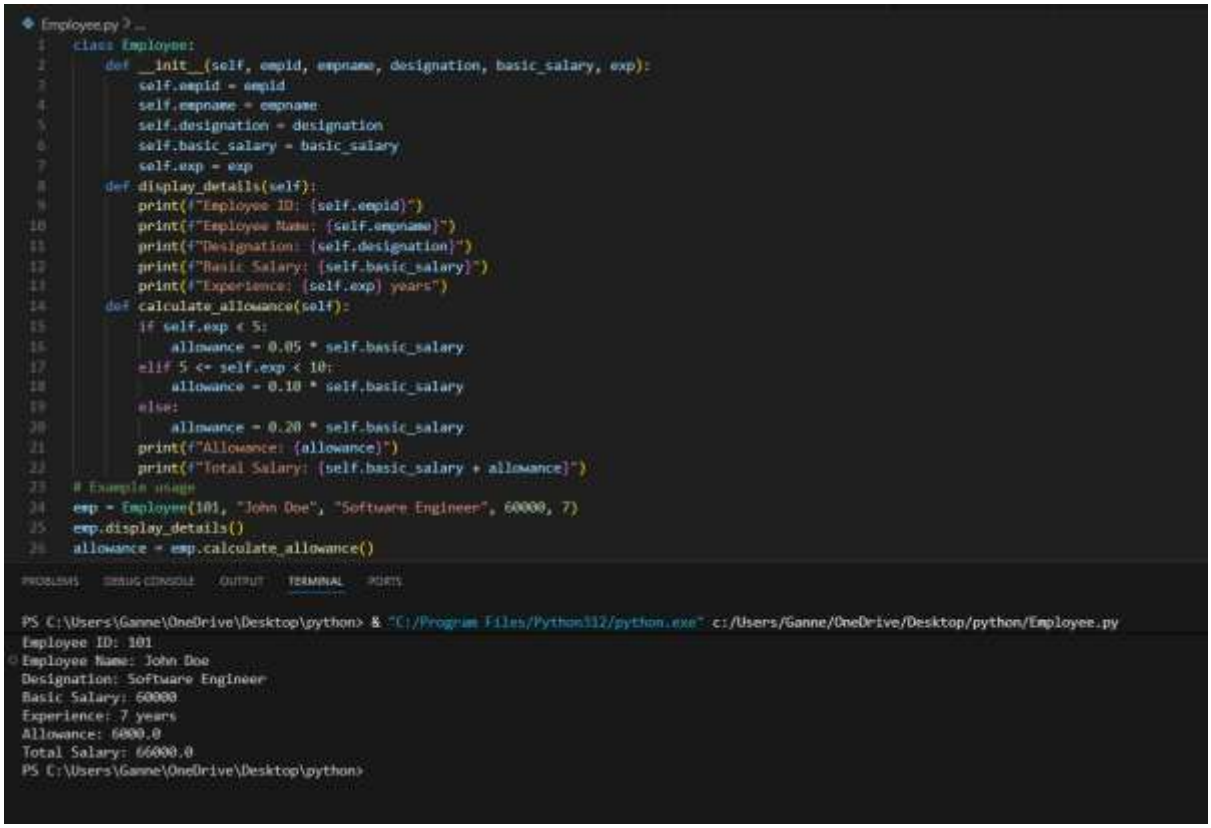
Ht.no : 2303A51670

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.



```python
class Employee:
    def __init__(self, empid, empname, designation, basic_salary, exp):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basic_salary = basic_salary
        self.exp = exp
    def display_details(self):
        print(f"Employee ID: {self.empid}")
        print(f"Employee Name: {self.empname}")
        print(f"Designation: {self.designation}")
        print(f"Basic Salary: {self.basic_salary}")
        print(f"Experience: {self.exp} years")
    def calculate_allowance(self):
        if self.exp < 5:
            allowance = 0.05 * self.basic_salary
        elif 5 <= self.exp < 10:
            allowance = 0.10 * self.basic_salary
        else:
            allowance = 0.20 * self.basic_salary
        print(f"Allowance: {allowance}")
        print(f"Total Salary: {self.basic_salary + allowance}")
# Example usage
emp = Employee(101, "John Doe", "Software Engineer", 60000, 7)
emp.display_details()
allowance = emp.calculate_allowance()
```

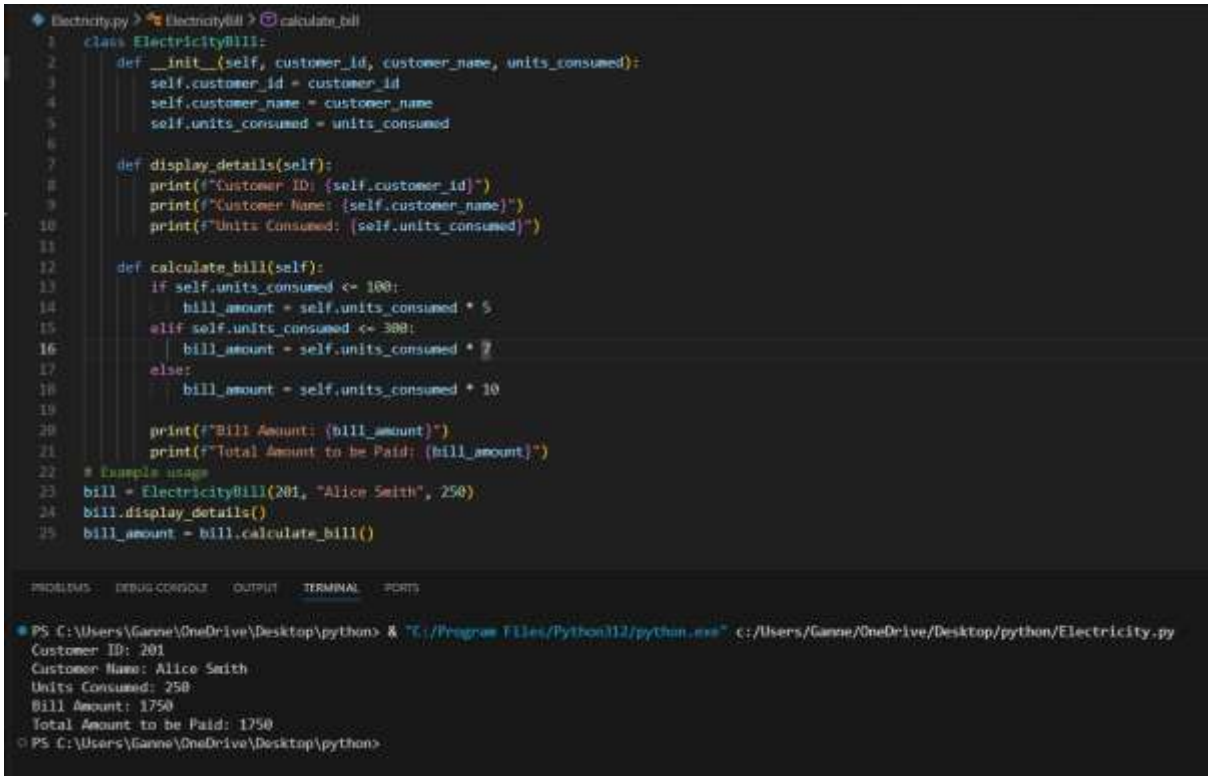
```
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/Employee.py
Employee ID: 101
Employee Name: John Doe
Designation: Software Engineer
Basic Salary: 60000
Experience: 7 years
Allowance: 6000.0
Total Salary: 66000.0
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

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.

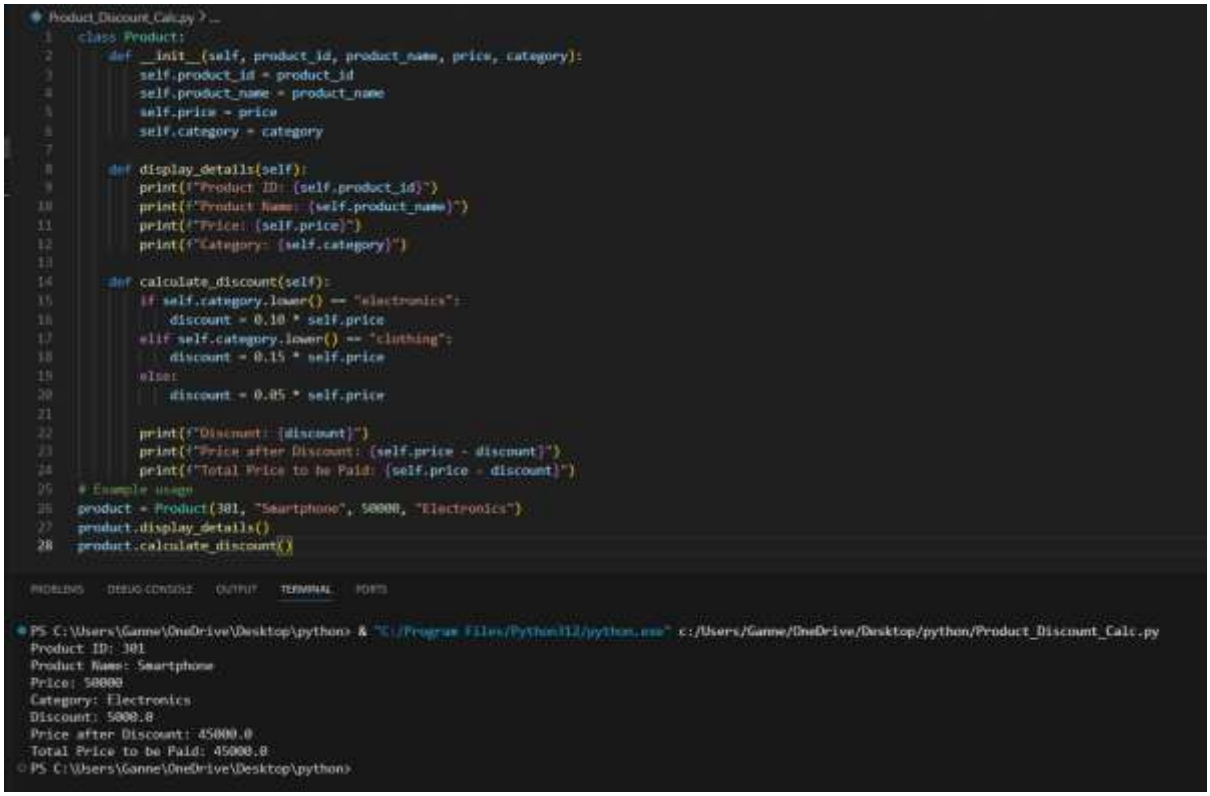


Task 3:

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.



```
Product_Discount_Calc.py > ...
class Product:
    def __init__(self, product_id, product_name, price, category):
        self.product_id = product_id
        self.product_name = product_name
        self.price = price
        self.category = category

    def display_details(self):
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Price: {self.price}")
        print(f"Category: {self.category}")

    def calculate_discount(self):
        if self.category.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        else:
            discount = 0.05 * self.price

        print(f"Discount: {discount}")
        print(f"Price after Discount: {self.price - discount}")
        print(f"Total Price to be Paid: {self.price - discount}")
# Example usage
product = Product(301, "Smartphone", 50000, "Electronics")
product.display_details()
product.calculate_discount()
```

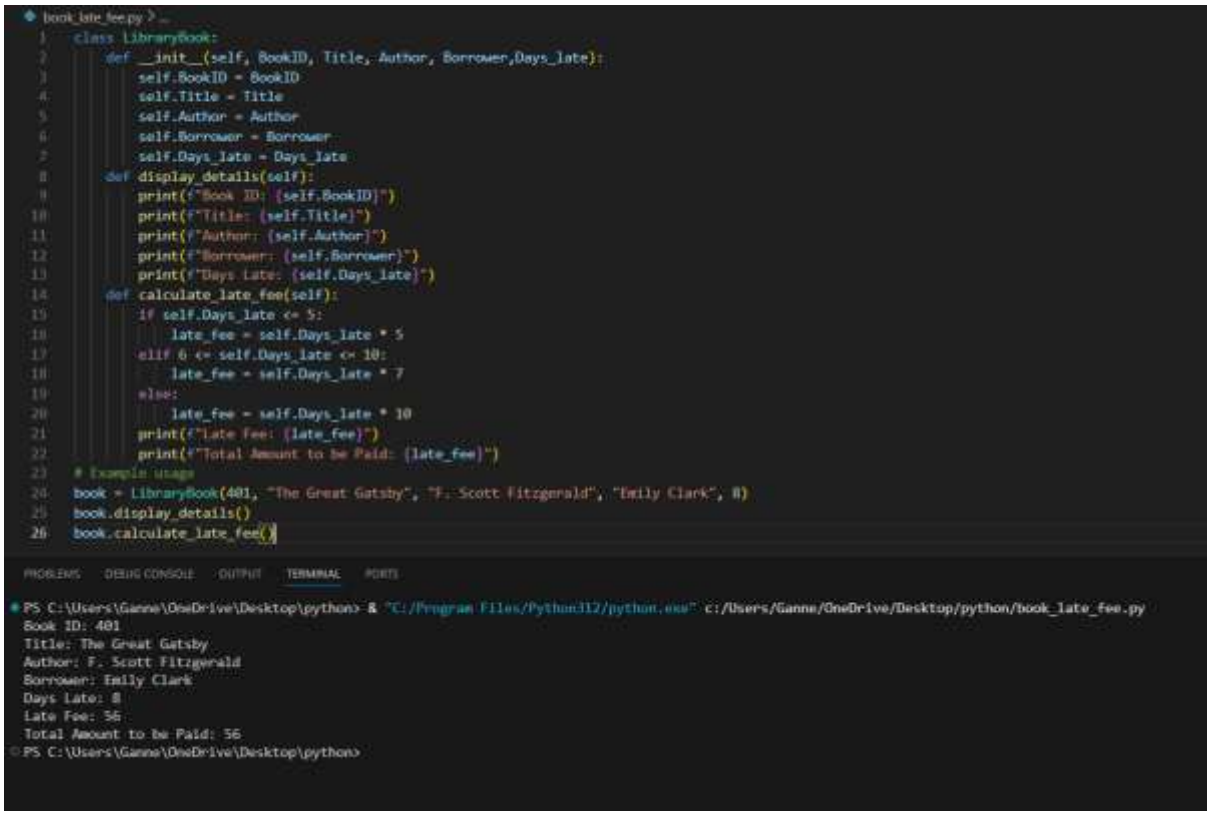
```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/Product_Discount_Calc.py
Product ID: 301
Product Name: Smartphone
Price: 50000
Category: Electronics
Discount: 5000.0
Price after Discount: 45000.0
Total Price to be Paid: 45000.0
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

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:

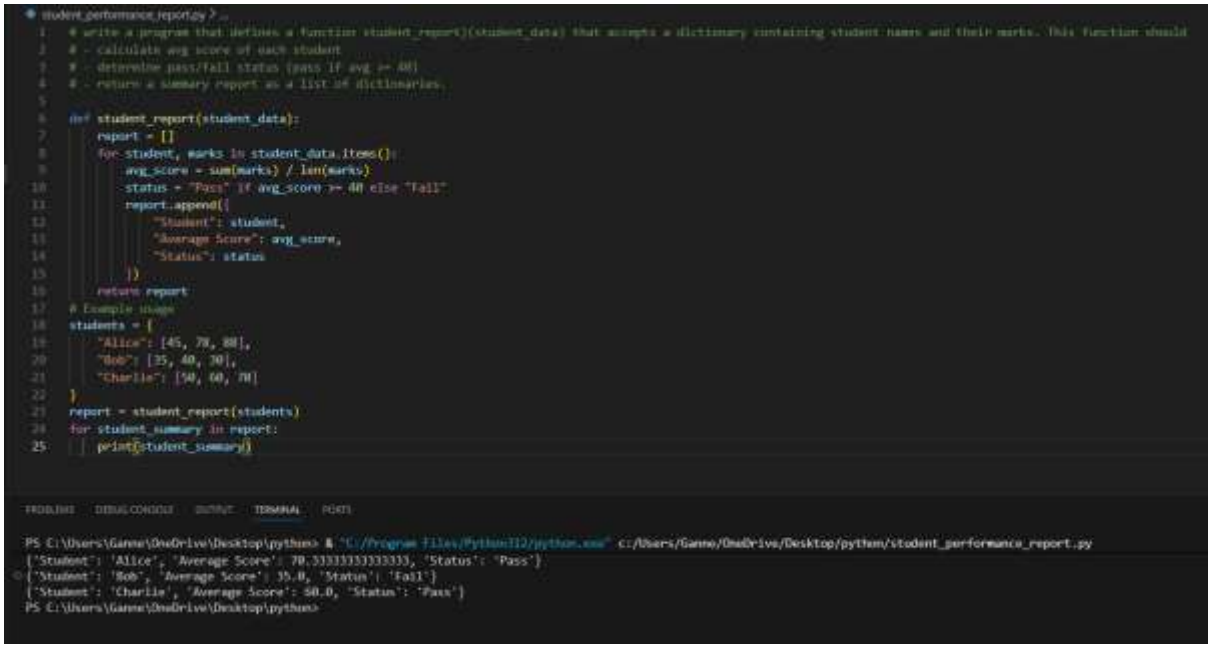
- Calculate the average score for each student

- 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:

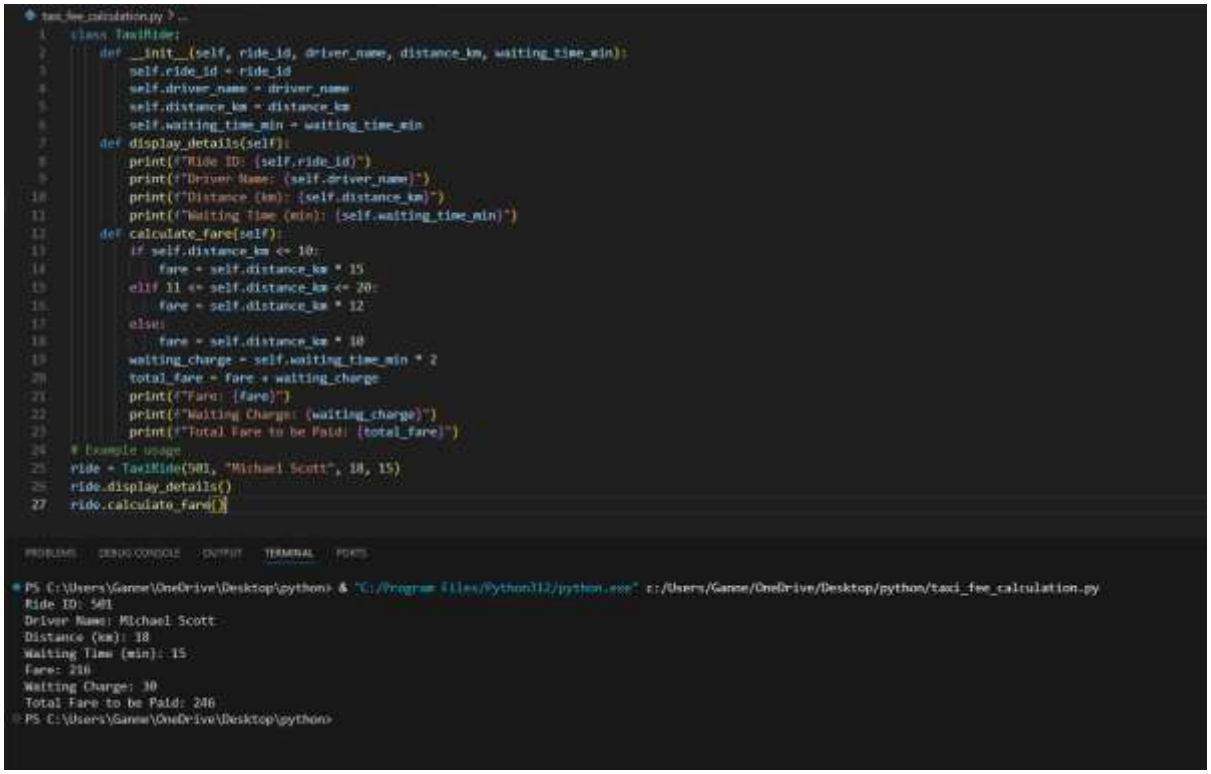
- ₹15 per km for the first 10 km

- ₹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.



```python
class TaxiRide:
    def __init__(self, ride_id, driver_name, distance_km, waiting_time_min):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.waiting_time_min = waiting_time_min
    def display_details(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Driver Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Waiting Time (min): {self.waiting_time_min}")
    def calculate_fare(self):
        if self.distance_km <= 10:
            fare = self.distance_km * 15
        elif 11 <= self.distance_km <= 20:
            fare = self.distance_km * 12
        else:
            fare = self.distance_km * 10
        waiting_charge = self.waiting_time_min * 2
        total_fare = fare + waiting_charge
        print(f"Fare: {fare}")
        print(f"Waiting Charge: {waiting_charge}")
        print(f"Total Fare to be Paid: {total_fare}")
# Example usage
ride = TaxiRide(501, "Michael Scott", 18, 15)
ride.display_details()
ride.calculate_fare()
```

```
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/taxi_fee_calculation.py
Ride ID: 501
Driver Name: Michael Scott
Distance (km): 18
Waiting Time (min): 15
Fare: 216
Waiting Charge: 30
Total Fare to be Paid: 246
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

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

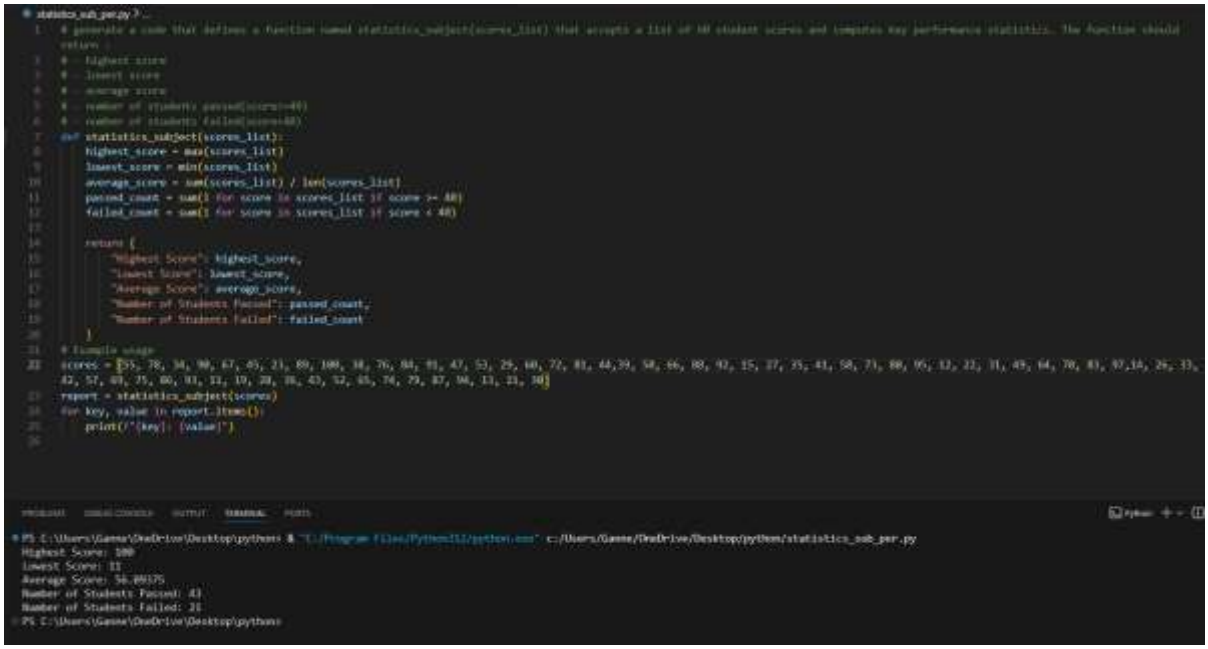
- Lowest score in the class

- Class average score

- Number of students passed (score ≥ 40)

- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic.



Task Description #8 (Transparency in Algorithm Optimization)

Task: 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.

• Validation that explanations align with runtime behavior.

-

prime.py

```python
# Generate Python code for two prime-checking methods and explain
#- how the optimized version improves performance."
# - transparent explanation of time complexity for both methods.
# - comparision highlighting efficiency improvements.

def is_prime_basic(n):
    """Check if a number is prime using basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using optimized method."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

prime.py

```python
# Explanation of time complexity:
# 1. Basic Method:
#    - Time Complexity: O(n)
#    - Explanation: In the basic method, we check all numbers from 2 to n
#      to see if they divide n evenly. In the worst case, we perform n-2
#      checks, leading to linear time complexity.
# 2. Optimized Method:
#    - Time Complexity: O(√n)
#    - Explanation: The optimized method reduces the number of checks
#      significantly by eliminating even numbers and checking only up to
#      the square root of n. This is because if n is divisible by any number
#      greater than its square root, the corresponding factor must be less
#      than the square root. Thus, we only need to check up to √n,
#      leading to a much faster algorithm for large n.
# Comparison:
# The optimized method is significantly more efficient than the basic
# method, especially for large numbers. While the basic method's linear
# time complexity can become impractical for large n, the optimized
# method's square root time complexity allows it to handle much larger
# numbers efficiently. This makes the optimized method preferable for
# prime checking in real-world applications.
# Example usage
number = 29
print(f"Is {number} prime (basic)? {is_prime_basic(number)}")
print(f"Is {number} prime (optimized)? {is_prime_optimized(number)}")
```

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/prime.py
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/prime.py
Is 29 prime (basic)? True
Is 29 prime (optimized)? True
PS C:\Users\Ganne\OneDrive\Desktop\python>
```

Task Description #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.

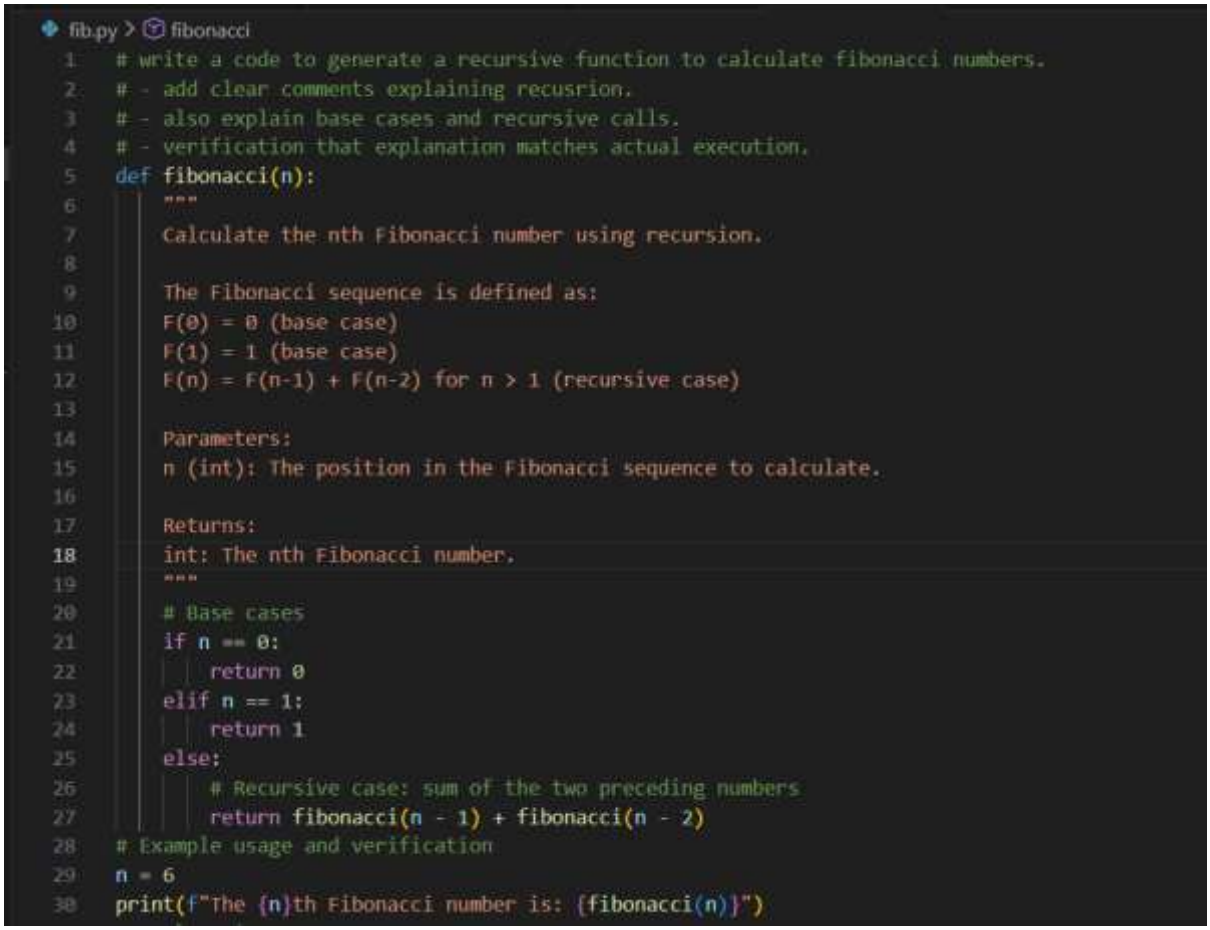
2. Ask AI to explain base cases and recursive calls.

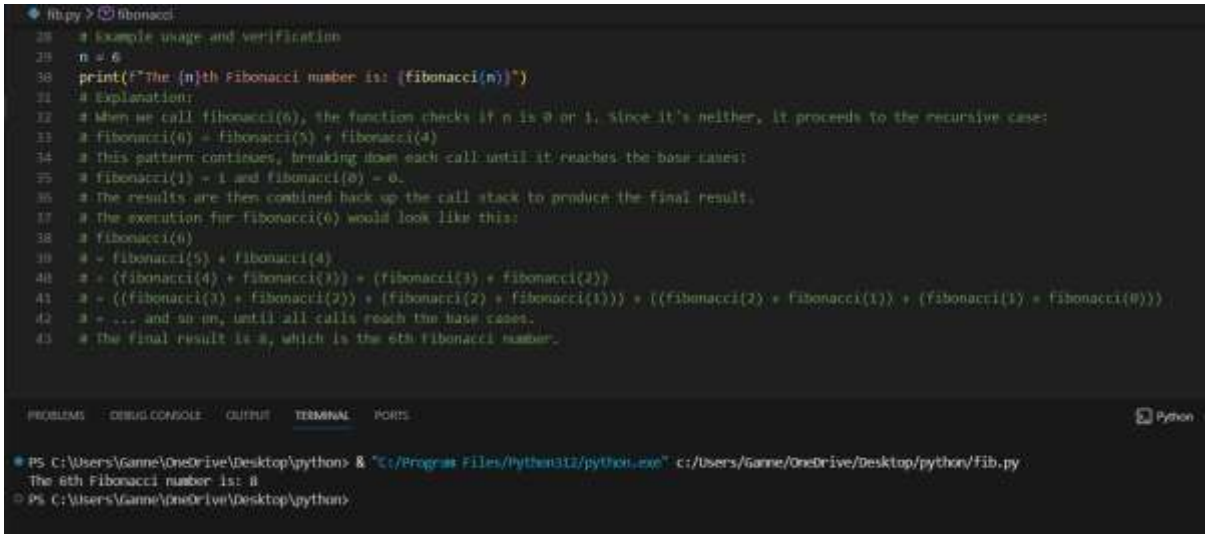
Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.





Task Description #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.

error_handling.py > process_file

```python
# Generate a Python program that reads and processes data from a file using try-except blocks. Include meaningful exception handling and clear comments explaining each possible error scenario.

def process_file(filename):
    """
    Reads a file and processes its data.
    Processing: counts number of lines and words.
    """
    try:
        # Tries to open the file in read mode
        with open(filename, "r") as file:
            content = file.read()
            # Data processing
            lines = content.splitlines()
            words = content.split()

            print("File processed successfully.")
            print("Number of lines:", len(lines))
            print("Number of words:", len(words))
            print(lines)
            print(words)

    except FileNotFoundError:
        # Raised when the file does not exist at the given path
        print("Error: The file was not found. Please check the file name or path.")
```

error_handling.py > process_file

```python
def process_file(filename):
            # Raised when the file does not exist at the given path
            print("Error: The file was not found. Please check the file name or path.")

    except PermissionError:
        # Raised when the program does not have permission to read the file
        print("Error: Permission denied. You do not have access to this file.")

    except Exception as e:
        # Handles any other unexpected runtime errors
        print("Unexpected error occurred:", e)

    else:
        # Executes only if no exception occurs
        print("File reading and processing completed without errors.")

    finally:
        # Always executes, useful for cleanup actions
        print("Program execution finished.")
# Example usage
process_file("sample.txt")
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

```
PS C:\Users\Ganne\OneDrive\Desktop\python> & "C:/Program Files/Python312/python.exe" c:/Users/Ganne/OneDrive/Desktop/python/error_handling.py
File processed successfully.
Number of lines: 3
Number of words: 9
['AI Assisted Coding', 'SR University', 'Roll no : 2303A51670']
['AI', 'Assisted', 'Coding', 'SR', 'University', 'Roll', 'no', ':', '2303A51670']
File reading and processing completed without errors.
Program execution finished.
PS C:\Users\Ganne\OneDrive\Desktop\python>
```