Assignment- 5.1 & 6

Name: A.LaxmiPrasanna

Ht.No: 2303A51272

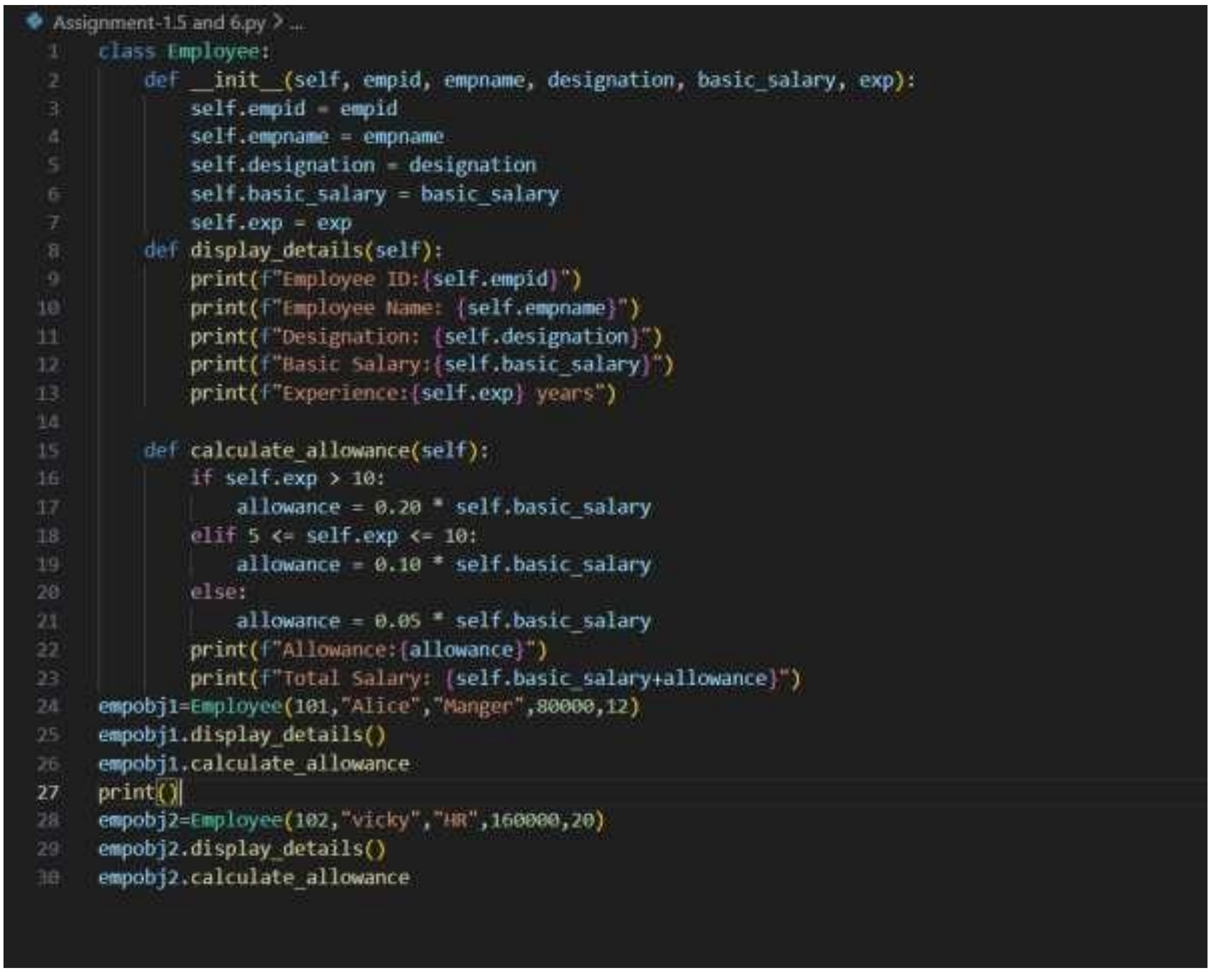
Batch: 23

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:

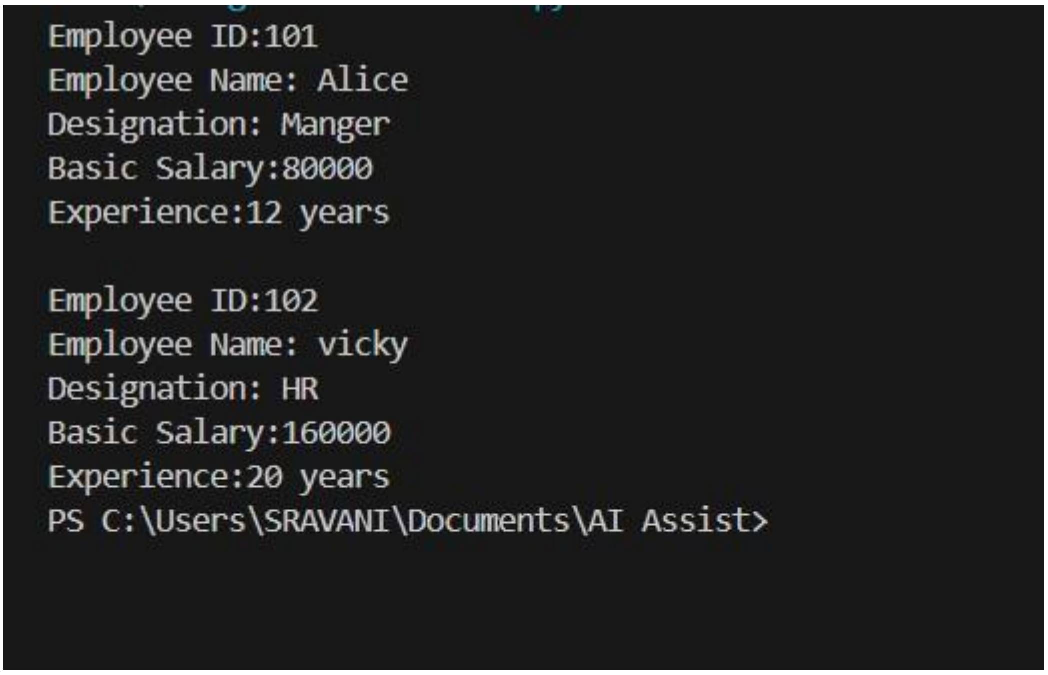
- If \`exp > 10 years\` → allowance = 20% of \`basic_salary\`
- If \`5 ≤ exp ≤ 10 years\` → allowance = 10% of \`basic_salary\`
- If \`exp < 5 years\` → allowance = 5% of \`basic_salary\`

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 > 10:
            allowance = 0.20 * self.basic_salary
        elif 5 <= self.exp <= 10:
            allowance = 0.10 * self.basic_salary
        else:
            allowance = 0.05 * self.basic_salary
        print(f"Allowance:{allowance}")
        print(f"Total Salary: {self.basic_salary+allowance}")
empobj1=Employee(101,"Alice","Manger",80000,12)
empobj1.display_details()
empobj1.calculate_allowance
print()
empobj2=Employee(102,"vicky","HR",160000,20)
empobj2.display_details()
empobj2.calculate_allowance
```



```
Employee ID:101
Employee Name: Alice
Designation: Manger
Basic Salary:80000
Experience:12 years

Employee ID:102
Employee Name: vicky
Designation: HR
Basic Salary:160000
Experience:20 years
PS C:\Users\SRAVANI\Documents\AI Assist>
```

Task 2:

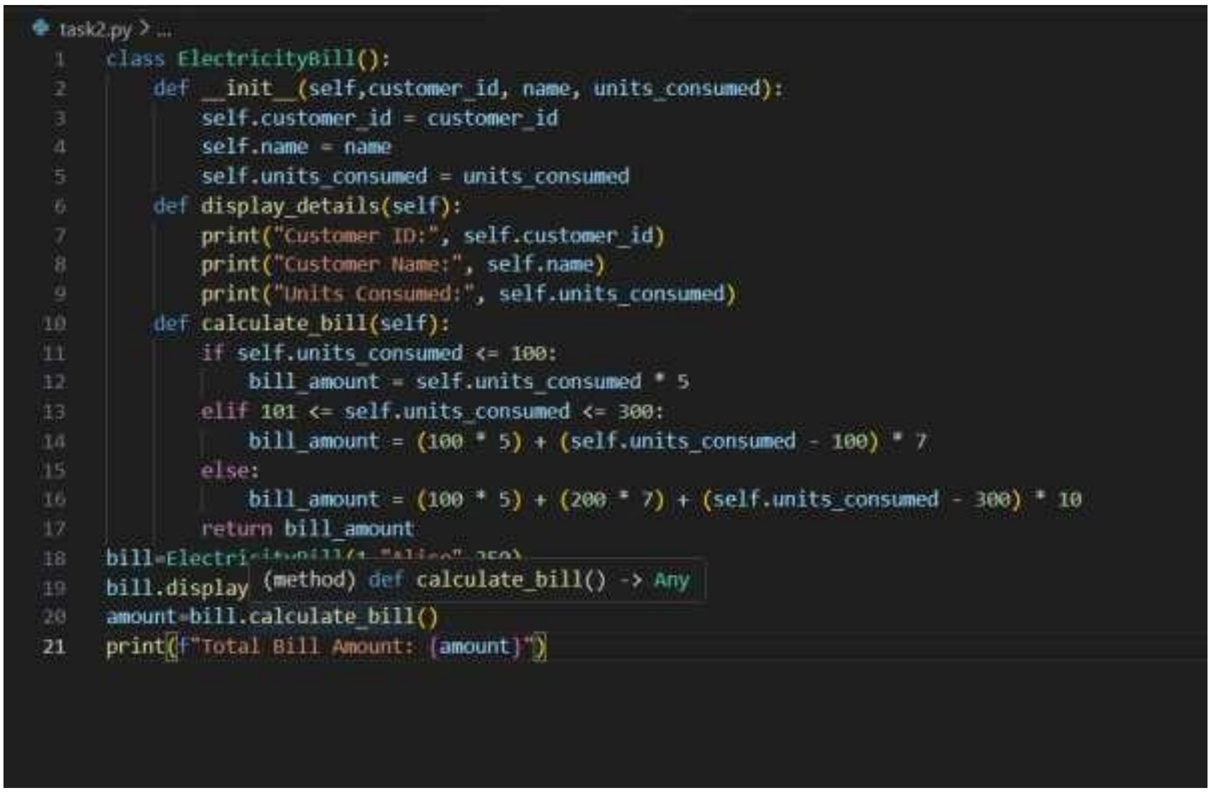
Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit

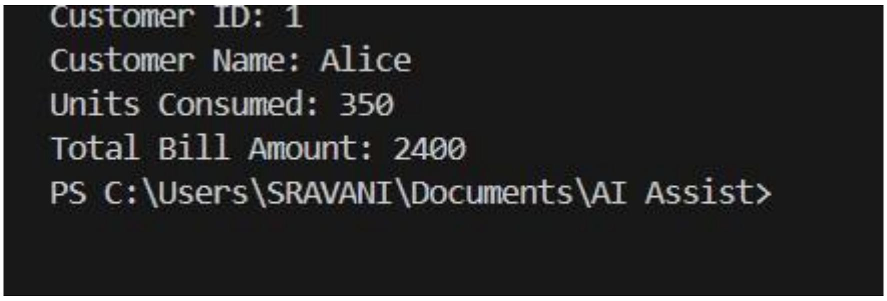
- 101 to 300 units → ₹7 per unit

- More than 300 units → ₹10 per unit

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



```
task2.py > ...
class ElectricityBill():
    def __init__(self,customer_id, name, units_consumed):
        self.customer_id = customer_id
        self.name = name
        self.units_consumed = units_consumed
    def display_details(self):
        print("Customer ID:", self.customer_id)
        print("Customer Name:", self.name)
        print("Units Consumed:", self.units_consumed)
    def calculate_bill(self):
        if self.units_consumed <= 100:
            bill_amount = self.units_consumed * 5
        elif 101 <= self.units_consumed <= 300:
            bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
        else:
            bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10
        return bill_amount
bill=ElectricityBill(1, "Alice", 350)
bill.display (method) def calculate_bill() -> Any
amount=bill.calculate_bill()
print(f"Total Bill Amount: {amount}")
```



```
Customer ID: 1
Customer Name: Alice
Units Consumed: 350
Total Bill Amount: 2400
PS C:\Users\SRAVANI\Documents\AI Assist>
```

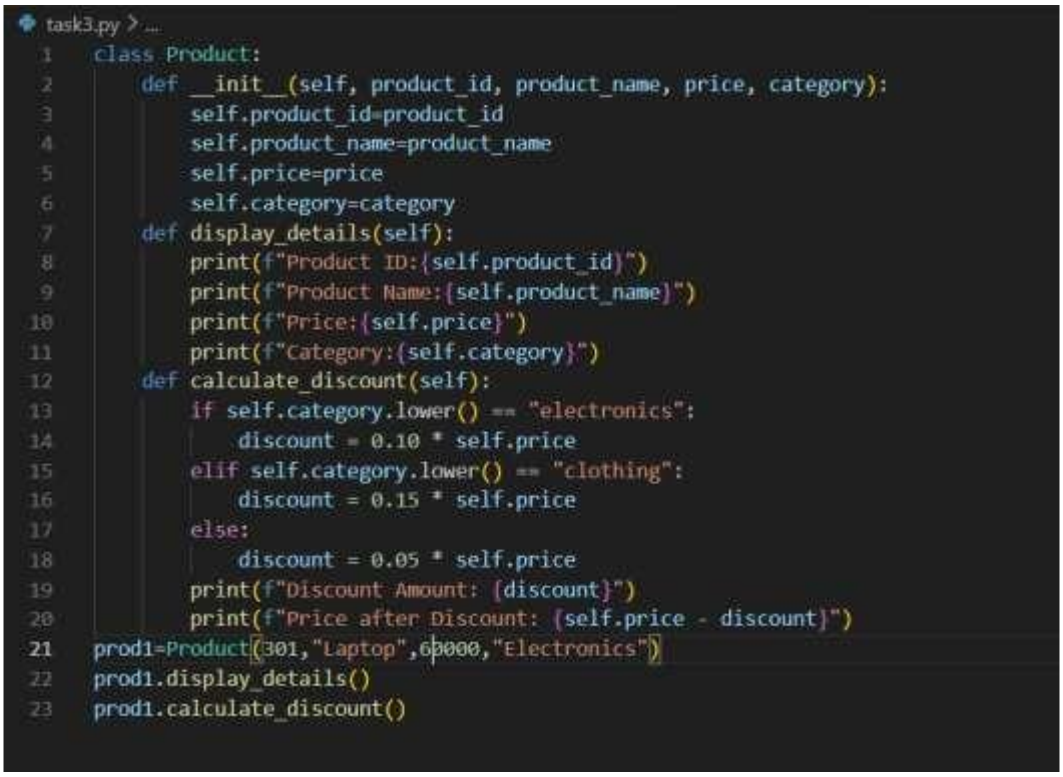
Task 3:

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:

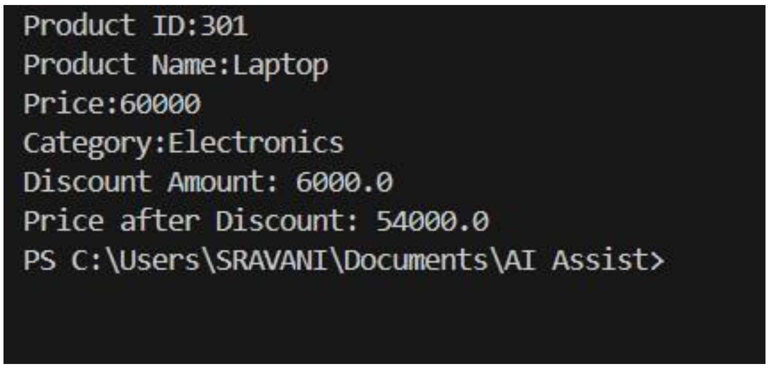
- Electronics → 10% discount
- Clothing → 15% discount
- Grocery → 5% discount

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

Task 4:



```python
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 Amount: {discount}")
        print(f"Price after Discount: {self.price - discount}")
prod1=Product(301,"Laptop",60000,"Electronics")
prod1.display_details()
prod1.calculate_discount()
```



```
Product ID:301
Product Name:Laptop
Price:60000
Category:Electronics
Discount Amount: 6000.0
Price after Discount: 54000.0
PS C:\Users\SRAVANI\Documents\AI Assist>
```

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:

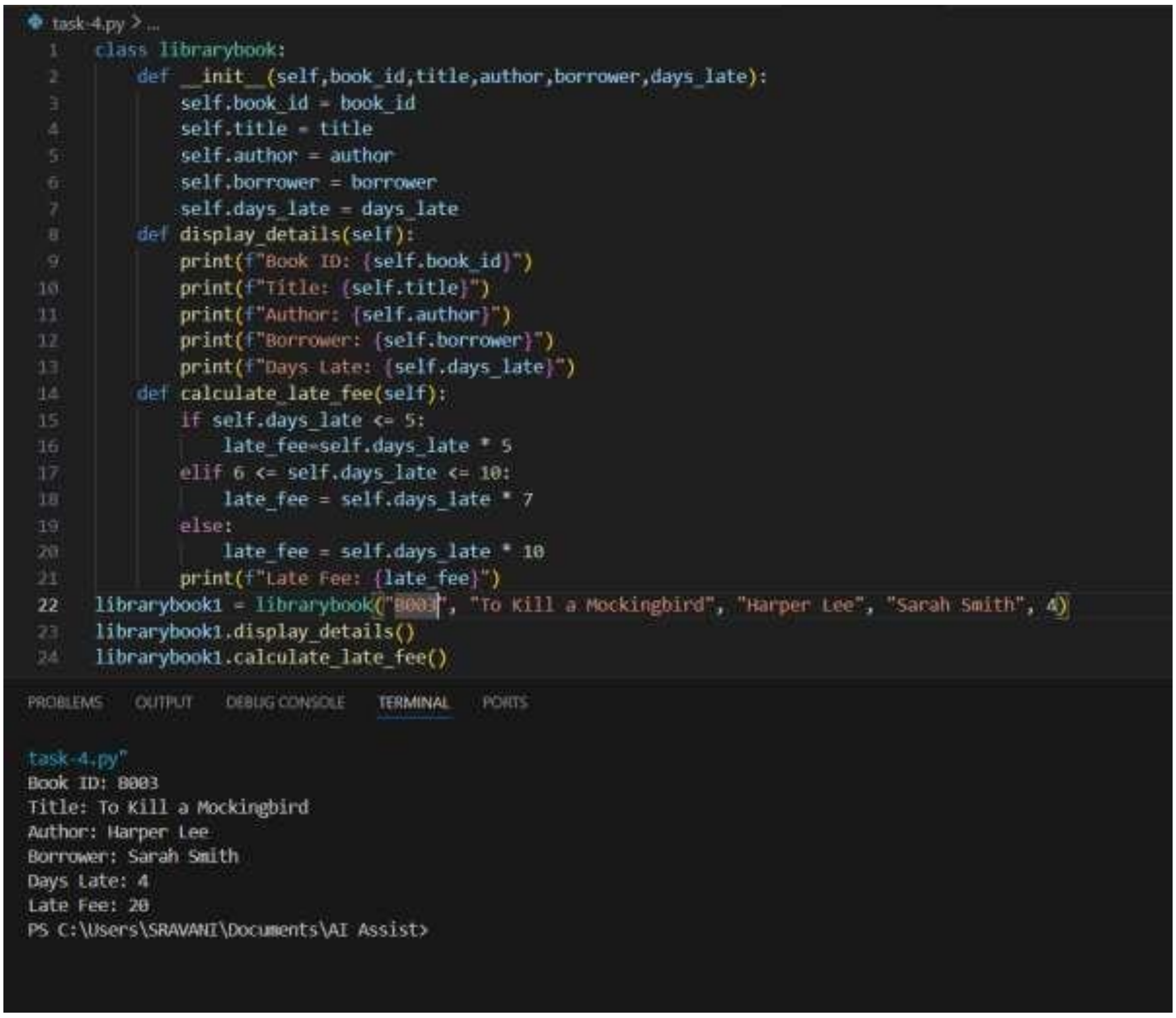
- Days late ≤ 5 → ₹5 per day

- 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:



```
task-4.py > ...
class librarybook:
    def __init__(self,book_id,title,author,borrower,days_late):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.borrower = borrower
        self.days_late = days_late
    def display_details(self):
        print(f"Book ID: {self.book_id}")
        print(f"Title: {self.title}")
        print(f"Author: {self.author}")
        print(f"Borrower: {self.borrower}")
        print(f"Days Late: {self.days_late}")
    def calculate_late_fee(self):
        if self.days_late <= 5:
            late_fee=self.days_late * 5
        elif 6 <= self.days_late <= 10:
            late_fee = self.days_late * 7
        else:
            late_fee = self.days_late * 10
        print(f"Late Fee: {late_fee}")
librarybook1 = librarybook("B003", "To Kill a Mockingbird", "Harper Lee", "Sarah Smith", 4)
librarybook1.display_details()
librarybook1.calculate_late_fee()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

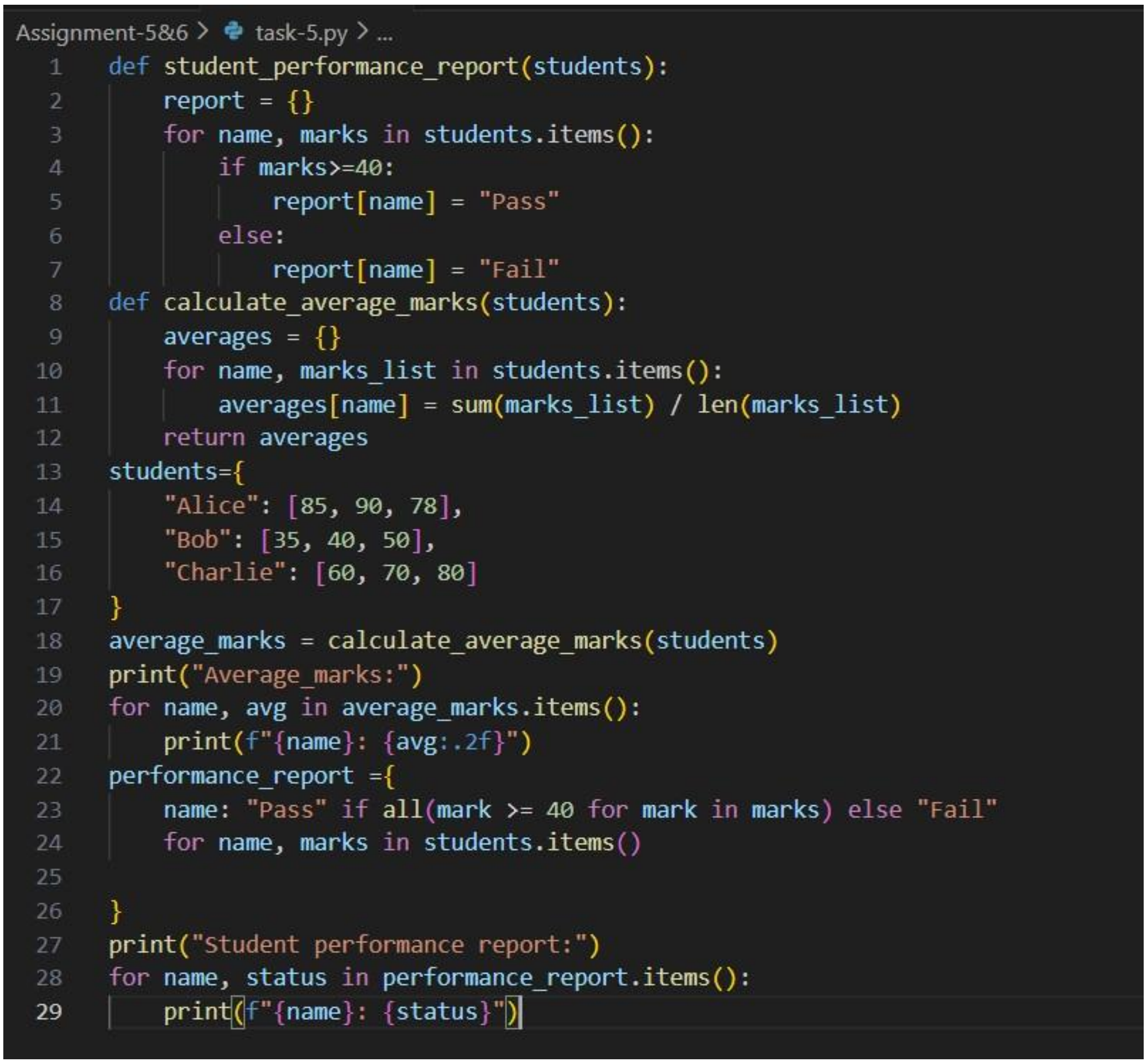
```
task-4.py"
Book ID: B003
Title: To Kill a Mockingbird
Author: Harper Lee
Borrower: Sarah Smith
Days Late: 4
Late Fee: 20
PS C:\Users\SRAVANI\Documents\AI Assist>
```

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.

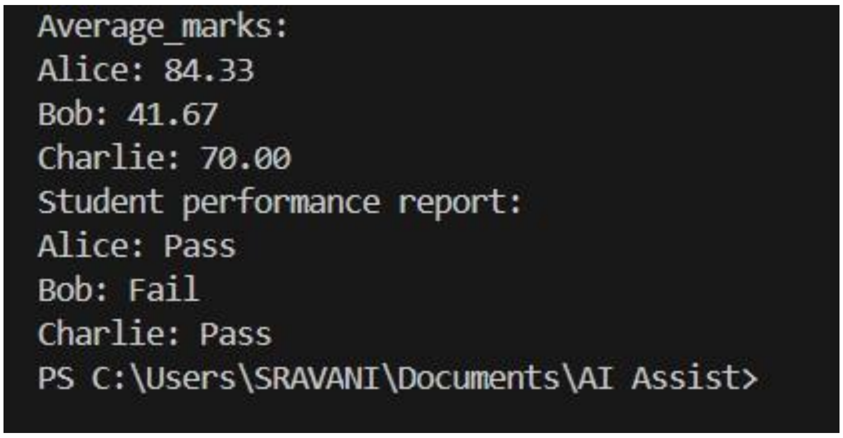
Task 6:



Assignment-5&6 > task-5.py > ...

```python
def student_performance_report(students):
    report = {}
    for name, marks in students.items():
        if marks>=40:
            report[name] = "Pass"
        else:
            report[name] = "Fail"
def calculate_average_marks(students):
    averages = {}
    for name, marks_list in students.items():
        averages[name] = sum(marks_list) / len(marks_list)
    return averages
students={
    "Alice": [85, 90, 78],
    "Bob": [35, 40, 50],
    "Charlie": [60, 70, 80]
}
average_marks = calculate_average_marks(students)
print("Average marks:")
for name, avg in average_marks.items():
    print(f"{name}: {avg:.2f}")
performance_report ={
    name: "Pass" if all(mark >= 40 for mark in marks) else "Fail"
    for name, marks in students.items()

}
print("Student performance report:")
for name, status in performance_report.items():
    print(f"{name}: {status}")
```



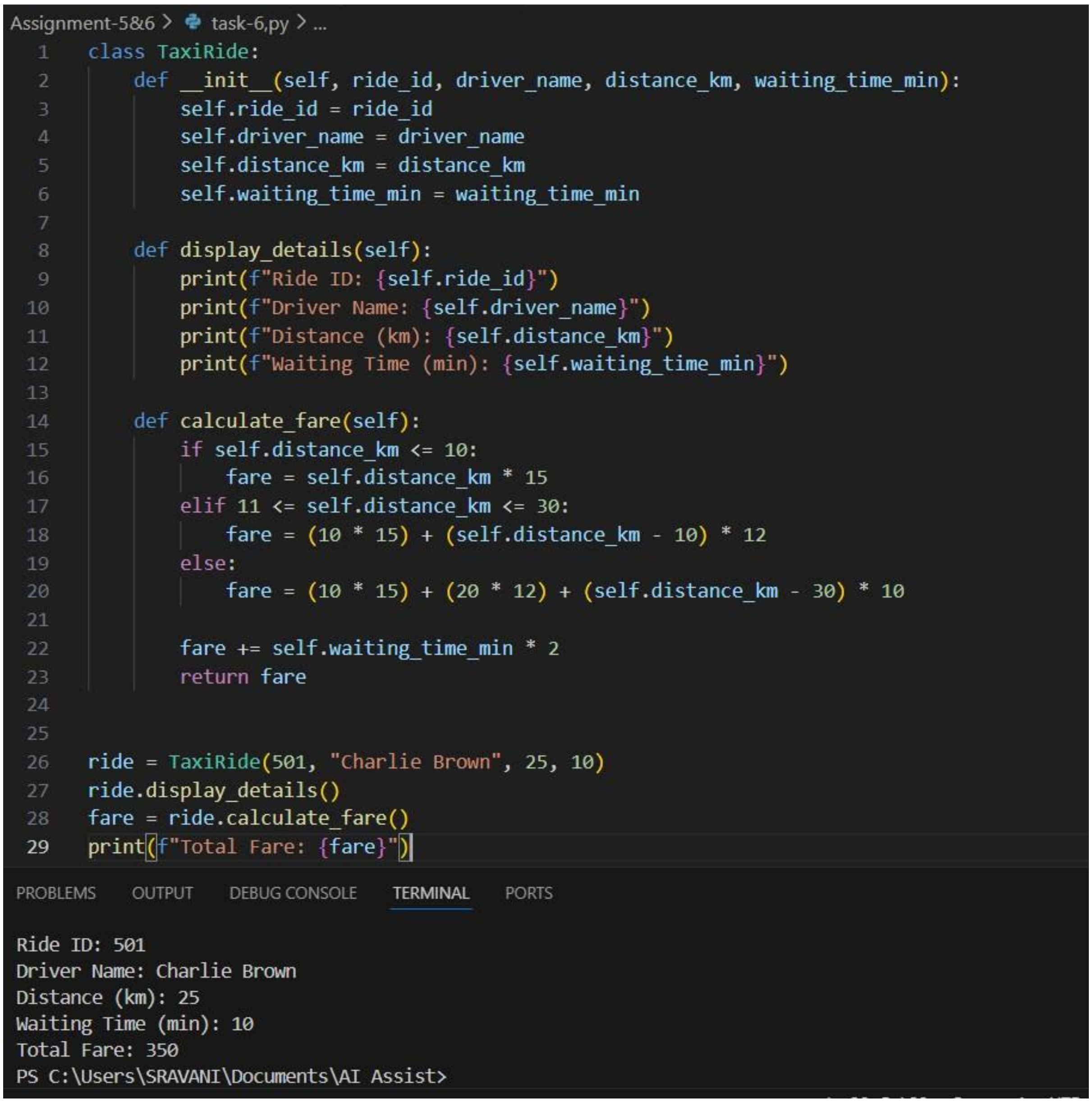
```
Average_marks:
Alice: 84.33
Bob: 41.67
Charlie: 70.00
Student performance report:
Alice: Pass
Bob: Fail
Charlie: Pass
PS C:\Users\SRAVANI\Documents\AI Assist>
```

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.



Assignment-5&6 > task-6.py > ...

```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 <= 30:
            fare = (10 * 15) + (self.distance_km - 10) * 12
        else:
            fare = (10 * 15) + (20 * 12) + (self.distance_km - 30) * 10

        fare += self.waiting_time_min * 2
        return fare


ride = TaxiRide(501, "Charlie Brown", 25, 10)
ride.display_details()
fare = ride.calculate_fare()
print(f"Total Fare: {fare}")
```

```
Ride ID: 501
Driver Name: Charlie Brown
Distance (km): 25
Waiting Time (min): 10
Total Fare: 350
PS C:\Users\SRAVANI\Documents\AI Assist>
```

Task 7:

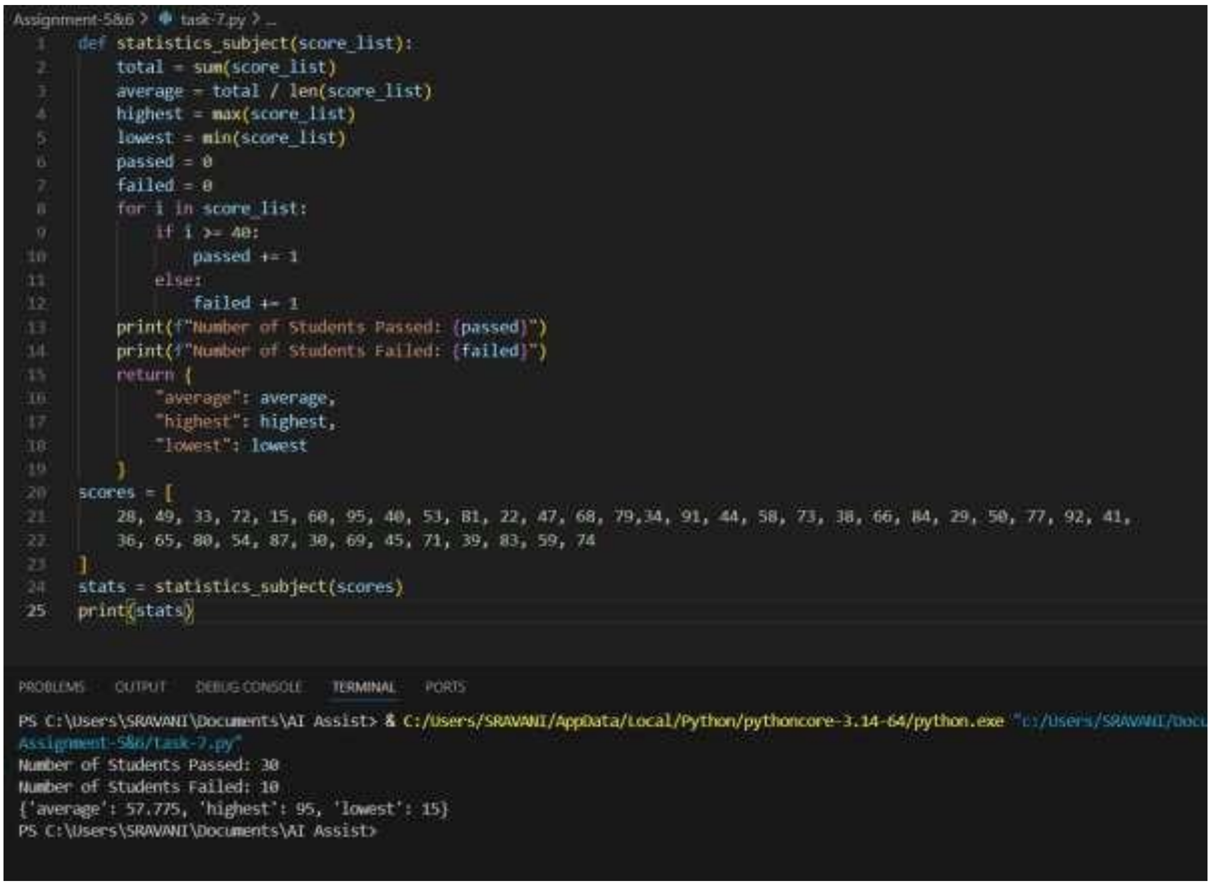
Statistics Subject Performance - Create a Python function `statistics_subject(scores_list)` that accepts a list of 60 student scores and computes key performance statistics. The function should return the following:

- 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



```python
def statistics_subject(score_list):
    total = sum(score_list)
    average = total / len(score_list)
    highest = max(score_list)
    lowest = min(score_list)
    passed = 0
    failed = 0
    for i in score_list:
        if i >= 40:
            passed += 1
        else:
            failed += 1
    print(f"Number of Students Passed: {passed}")
    print(f"Number of Students Failed: {failed}")
    return {
        "average": average,
        "highest": highest,
        "lowest": lowest
    }
scores = [
    28, 49, 33, 72, 15, 60, 95, 40, 53, 81, 22, 47, 68, 79,34, 91, 44, 58, 73, 38, 66, 84, 29, 50, 77, 92, 41,
    36, 65, 80, 54, 87, 30, 69, 45, 71, 39, 83, 59, 74
]
stats = statistics_subject(scores)
print(stats)
```

```
PS C:\Users\SRAVANI\Documents\AI Assist> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/SRAVANI/Docu
Assignment-5&6/task-7.py"
Number of Students Passed: 30
Number of Students Failed: 10
{'average': 57.775, 'highest': 95, 'lowest': 15}
PS C:\Users\SRAVANI\Documents\AI Assist>
```

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

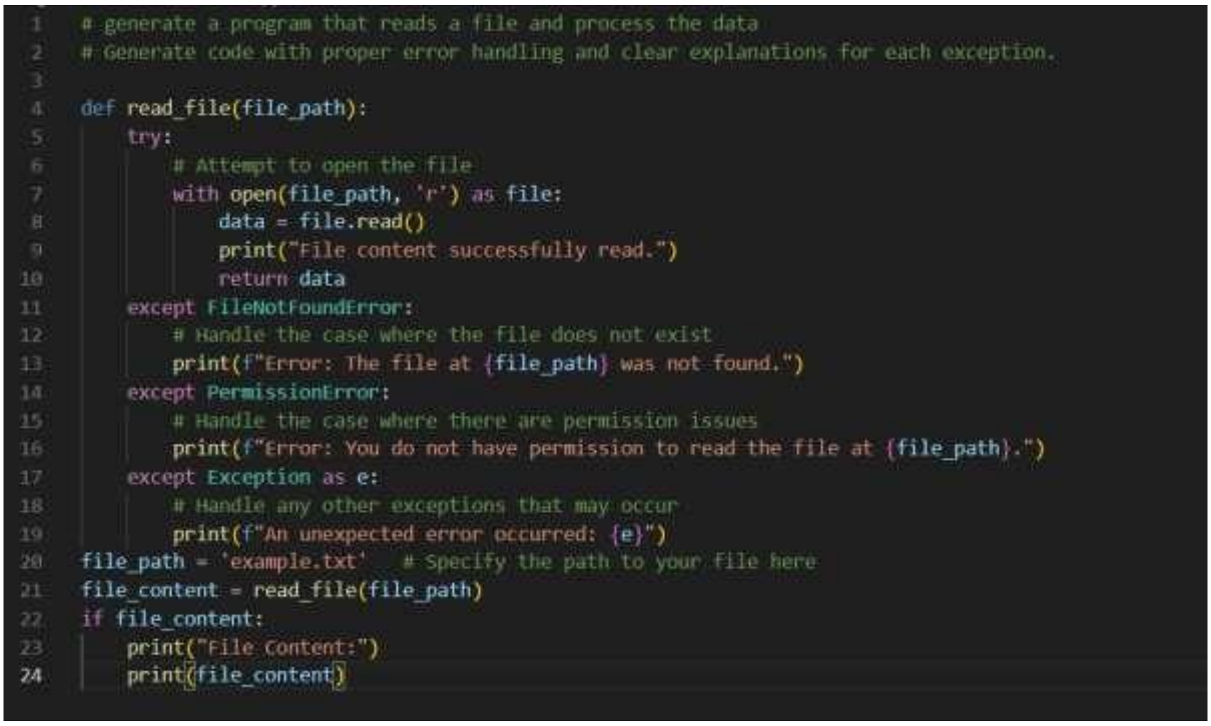
how the optimized version improves performance.”

Expected Output:

• Code for both methods.

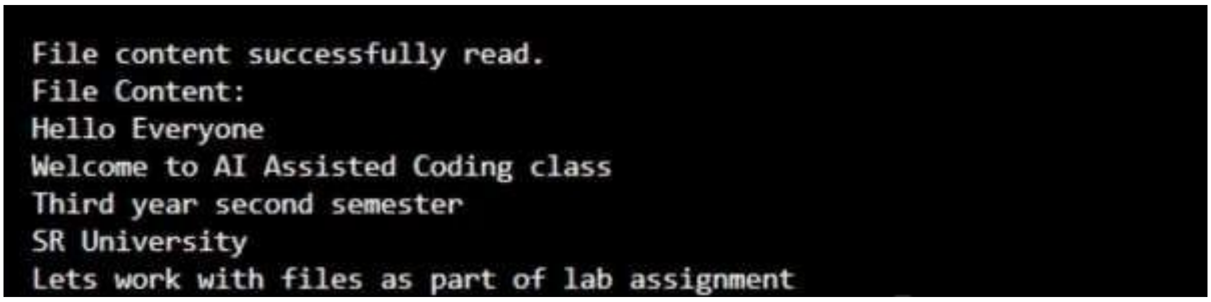
• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.



```python
# generate a program that reads a file and process the data
# Generate code with proper error handling and clear explanations for each exception.

def read_file(file_path):
    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.read()
            print("File content successfully read.")
            return data
    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file at {file_path} was not found.")
    except PermissionError:
        # Handle the case where there are permission issues
        print(f"Error: You do not have permission to read the file at {file_path}.")
    except Exception as e:
        # Handle any other exceptions that may occur
        print(f"An unexpected error occurred: {e}")
file_path = 'example.txt'   # Specify the path to your file here
file_content = read_file(file_path)
if file_content:
    print("File Content:")
    print(file_content)
```



```
File content successfully read.
File Content:
Hello Everyone
Welcome to AI Assisted Coding class
Third year second semester
SR University
Lets work with files as part of lab assignment
```

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.

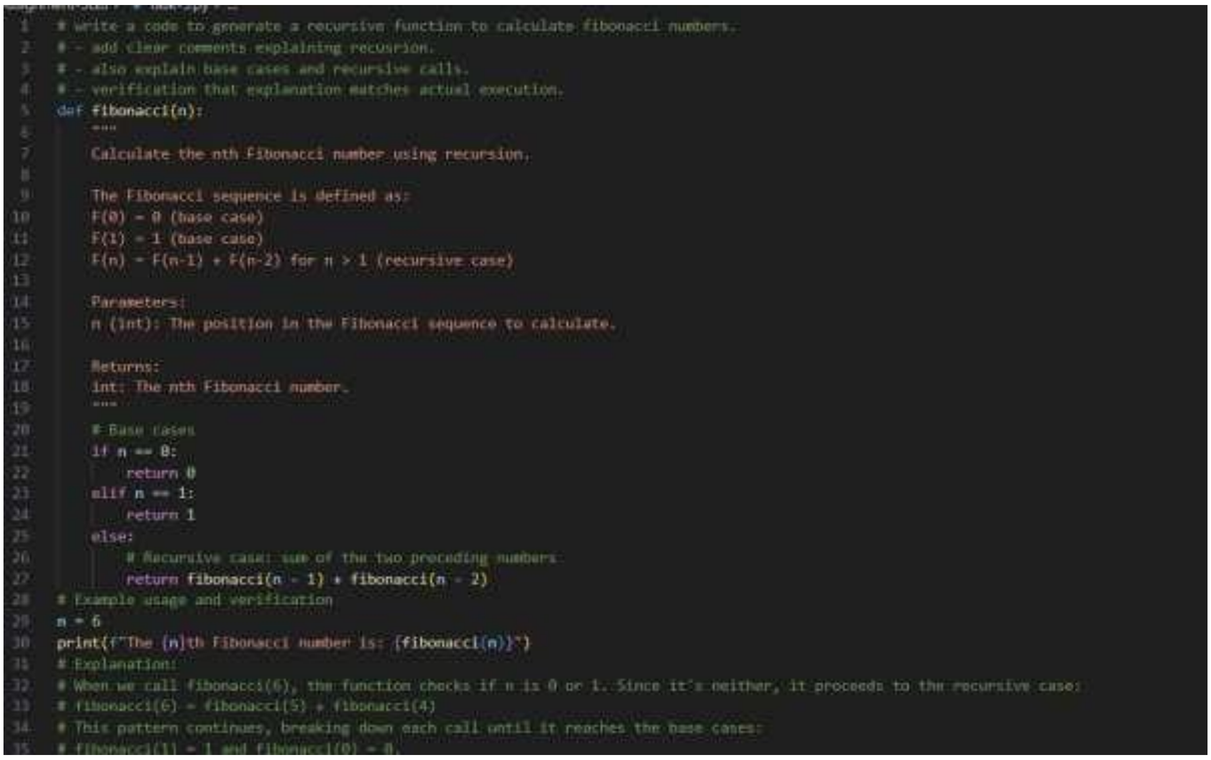
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.

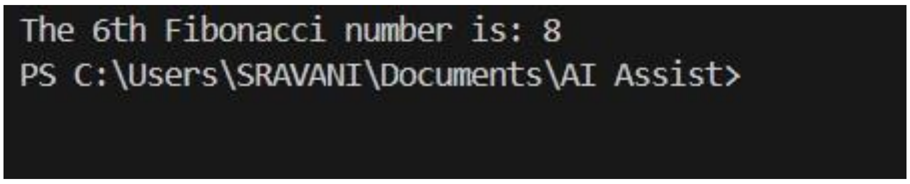


```python
# write a code to generate a recursive function to calculate fibonacci numbers.
# - add clear comments explaining recursion.
# - also explain base cases and recursive calls.
# - verification that explanation matches actual execution.
def fibonacci(n):
    """
    Calculate the nth Fibonacci number using recursion.

    The Fibonacci sequence is defined as:
    F(0) = 0 (base case)
    F(1) = 1 (base case)
    F(n) = F(n-1) + F(n-2) for n > 1 (recursive case)

    Parameters:
    n (int): The position in the Fibonacci sequence to calculate.

    Returns:
    int: The nth Fibonacci number.
    """
    # Base cases
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        # Recursive case: sum of the two preceding numbers
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage and verification
n = 6
print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
# Explanation:
# When we call fibonacci(6), the function checks if n is 0 or 1. Since it's neither, it proceeds to the recursive case:
# fibonacci(6) = fibonacci(5) + fibonacci(4)
# This pattern continues, breaking down each call until it reaches the base cases:
# fibonacci(1) = 1 and fibonacci(0) = 0.
```



```
The 6th Fibonacci number is: 8
PS C:\Users\SRAVANI\Documents\AI Assist>
```

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

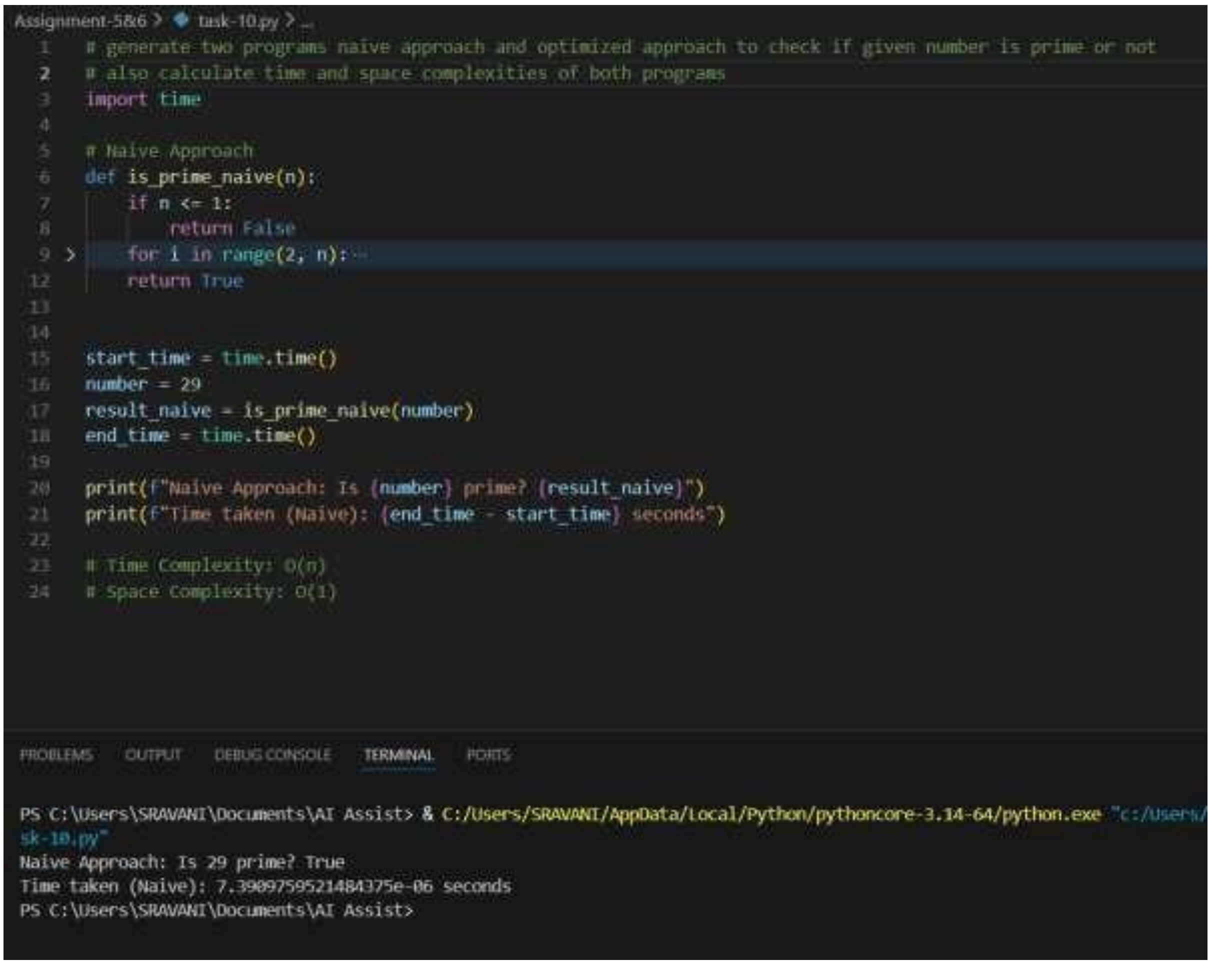
each exception.”

Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.



```python
# generate two programs naive approach and optimized approach to check if given number is prime or not
# also calculate time and space complexities of both programs
import time

# Naive Approach
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n): ...
    return True


start_time = time.time()
number = 29
result_naive = is_prime_naive(number)
end_time = time.time()

print(f"Naive Approach: Is {number} prime? {result_naive}")
print(f"Time taken (Naive): {end_time - start_time} seconds")

# Time Complexity: O(n)
# Space Complexity: O(1)
```

```
PS C:\Users\SRAVANI\Documents\AI Assist> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/
sk-10.py"
Naive Approach: Is 29 prime? True
Time taken (Naive): 7.3909759521484375e-06 seconds
PS C:\Users\SRAVANI\Documents\AI Assist>
```