LAB ASSIGNMENT 5.1 - 6

NAME : Aravind Reddy

Hall Ticket No : 2303a51027

Batch No : 01

Course Title: AI Assisted Coding

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

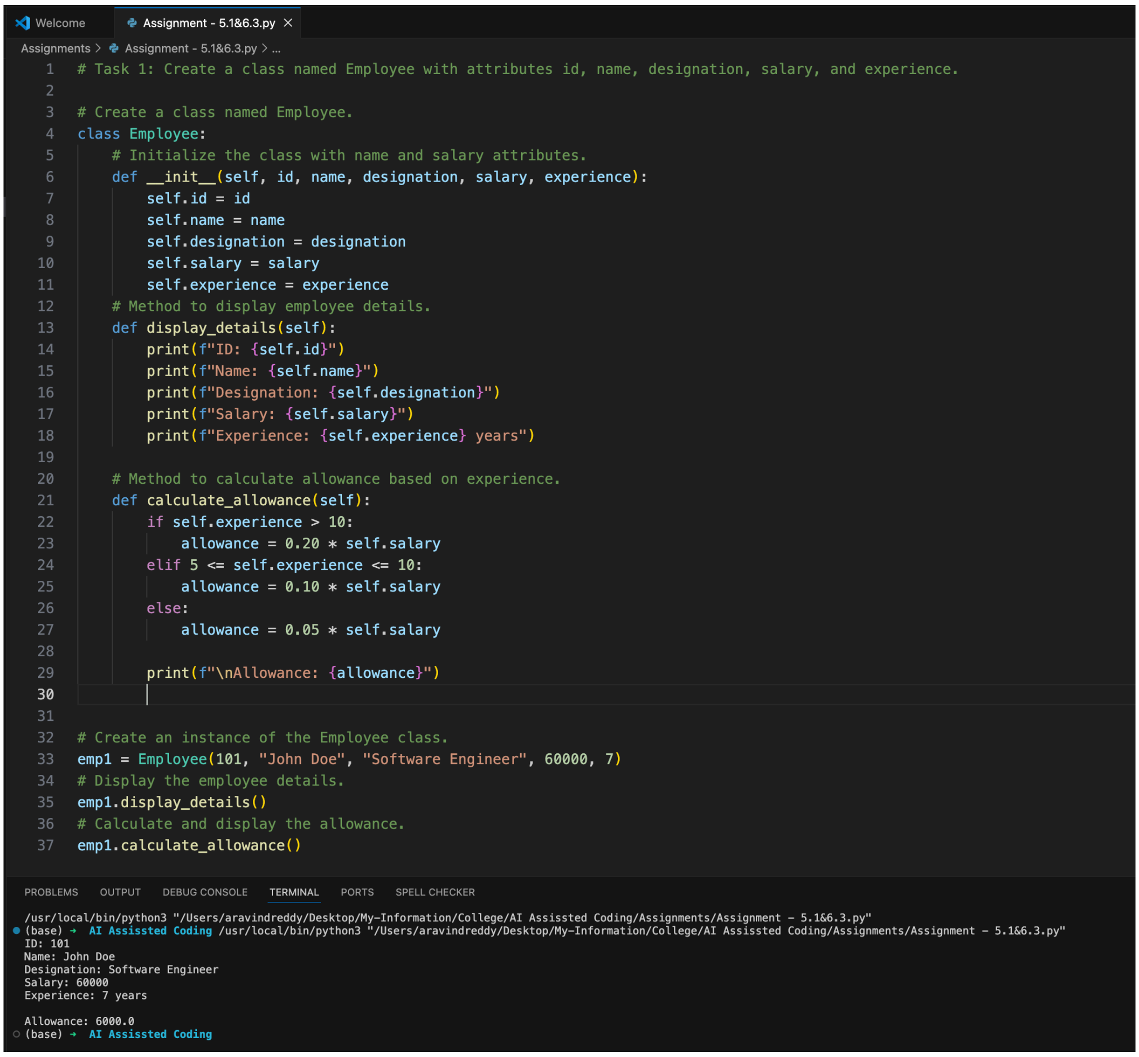
based on experience:

- 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.



Welcome | Assignment - 5.1&6.3.py

Assignments > Assignment - 5.1&6.3.py > ...

```python
# Task 1: Create a class named Employee with attributes id, name, designation, salary, and experience.

# Create a class named Employee.
class Employee:
    # Initialize the class with name and salary attributes.
    def __init__(self, id, name, designation, salary, experience):
        self.id = id
        self.name = name
        self.designation = designation
        self.salary = salary
        self.experience = experience
    # Method to display employee details.
    def display_details(self):
        print(f"ID: {self.id}")
        print(f"Name: {self.name}")
        print(f"Designation: {self.designation}")
        print(f"Salary: {self.salary}")
        print(f"Experience: {self.experience} years")

    # Method to calculate allowance based on experience.
    def calculate_allowance(self):
        if self.experience > 10:
            allowance = 0.20 * self.salary
        elif 5 <= self.experience <= 10:
            allowance = 0.10 * self.salary
        else:
            allowance = 0.05 * self.salary

        print(f"\nAllowance: {allowance}")


# Create an instance of the Employee class.
emp1 = Employee(101, "John Doe", "Software Engineer", 60000, 7)
# Display the employee details.
emp1.display_details()
# Calculate and display the allowance.
emp1.calculate_allowance()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.3.py"
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.3.py"
ID: 101
Name: John Doe
Designation: Software Engineer
Salary: 60000
Experience: 7 years

Allowance: 6000.0
○ (base) → AI Assissted Coding
```

Task 2 :

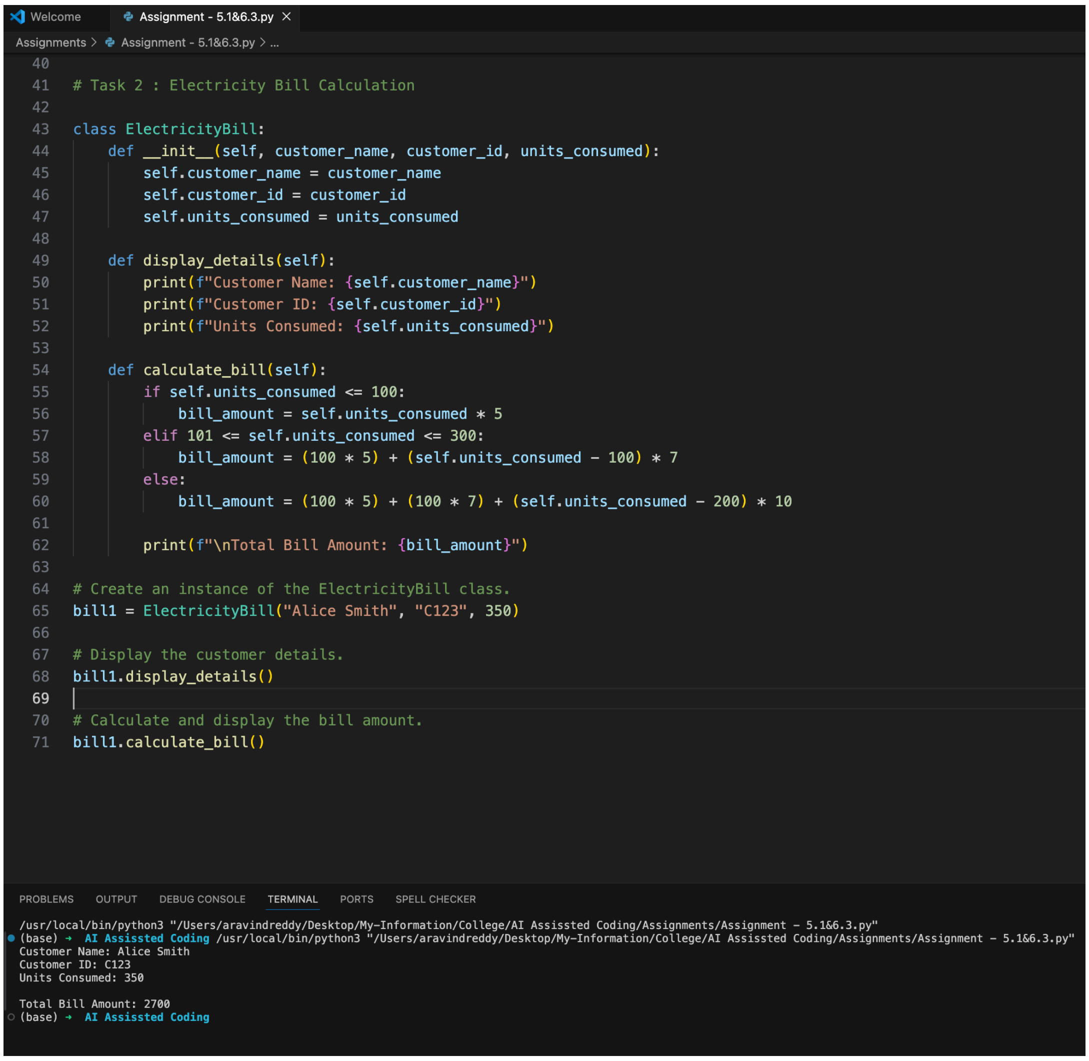
Electricity Bill Calculation- Create Python code that defines a class named `ElectricityBill` with attributes: `customer_id`, `name`, and `units_consumed`. Implement a method `display_details()` to print customer details, and a method `calculate_bill()` where:

- 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.



```python

# Task 2 : Electricity Bill Calculation

class ElectricityBill:
    def __init__(self, customer_name, customer_id, units_consumed):
        self.customer_name = customer_name
        self.customer_id = customer_id
        self.units_consumed = units_consumed

    def display_details(self):
        print(f"Customer Name: {self.customer_name}")
        print(f"Customer ID: {self.customer_id}")
        print(f"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) + (100 * 7) + (self.units_consumed - 200) * 10

        print(f"\nTotal Bill Amount: {bill_amount}")

# Create an instance of the ElectricityBill class.
bill1 = ElectricityBill("Alice Smith", "C123", 350)

# Display the customer details.
bill1.display_details()

# Calculate and display the bill amount.
bill1.calculate_bill()
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.3.py"
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.3.py"
Customer Name: Alice Smith
Customer ID: C123
Units Consumed: 350

Total Bill Amount: 2700
(base) → AI Assissted Coding
```

Task 3:

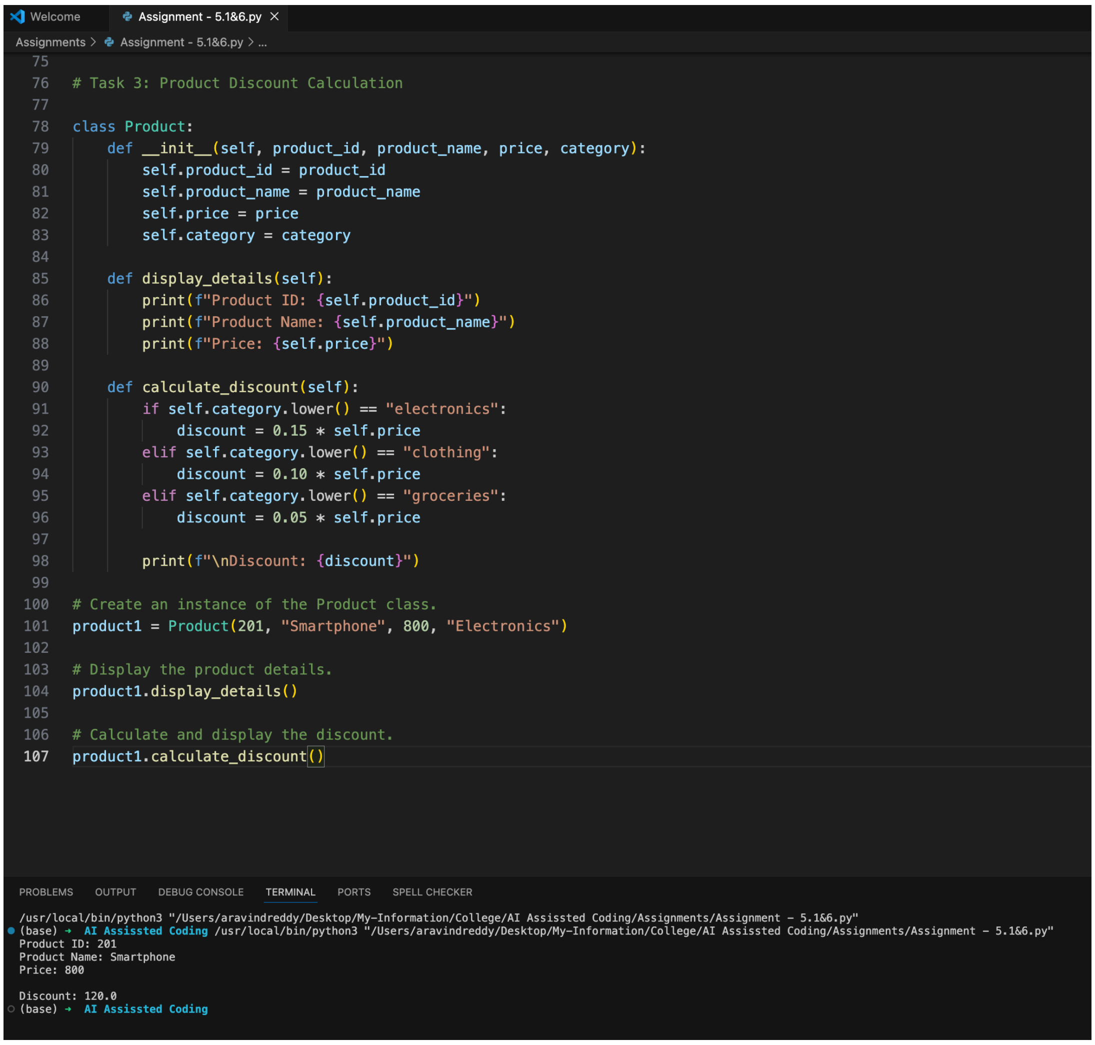
Product Discount Calculation- Create Python code that defines a class named `Product` with attributes: `product_id`, `product_name`, `price`, and `category`. Implement a method `display_details()` to print product details. Implement another method `calculate_discount()` where:

- Electronics → 10% discount

- Clothing → 15% discount

- Grocery → 5% discount

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



```python

# Task 3: Product Discount Calculation

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}")

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

        print(f"\nDiscount: {discount}")

# Create an instance of the Product class.
product1 = Product(201, "Smartphone", 800, "Electronics")

# Display the product details.
product1.display_details()

# Calculate and display the discount.
product1.calculate_discount()
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
Product ID: 201
Product Name: Smartphone
Price: 800

Discount: 120.0
(base) → AI Assissted Coding
```

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()`

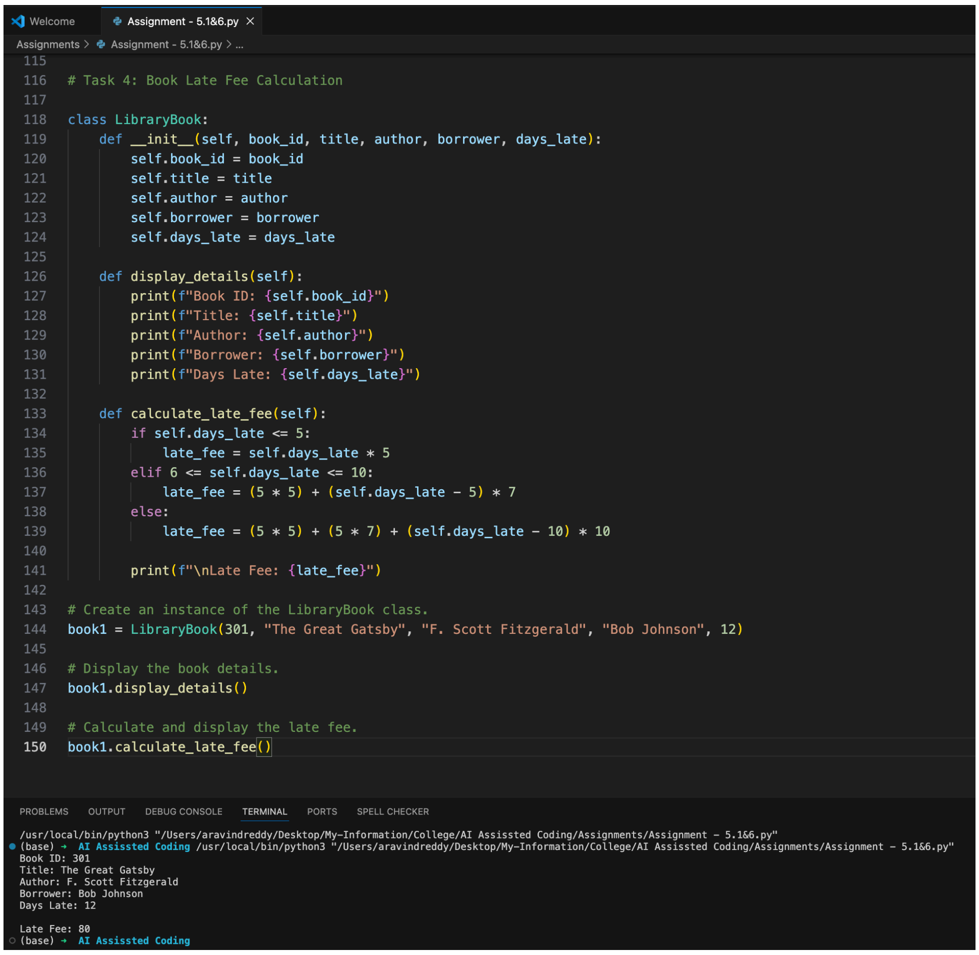
to print book details, and a method `calculate_late_fee()` where:

- 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.



```python

# Task 4: Book Late Fee Calculation

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 = (5 * 5) + (self.days_late - 5) * 7
        else:
            late_fee = (5 * 5) + (5 * 7) + (self.days_late - 10) * 10

        print(f"\nLate Fee: {late_fee}")

# Create an instance of the LibraryBook class.
book1 = LibraryBook(301, "The Great Gatsby", "F. Scott Fitzgerald", "Bob Johnson", 12)

# Display the book details.
book1.display_details()

# Calculate and display the late fee.
book1.calculate_late_fee()
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
Book ID: 301
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Bob Johnson
Days Late: 12

Late Fee: 80
○ (base) → AI Assissted Coding
```

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:

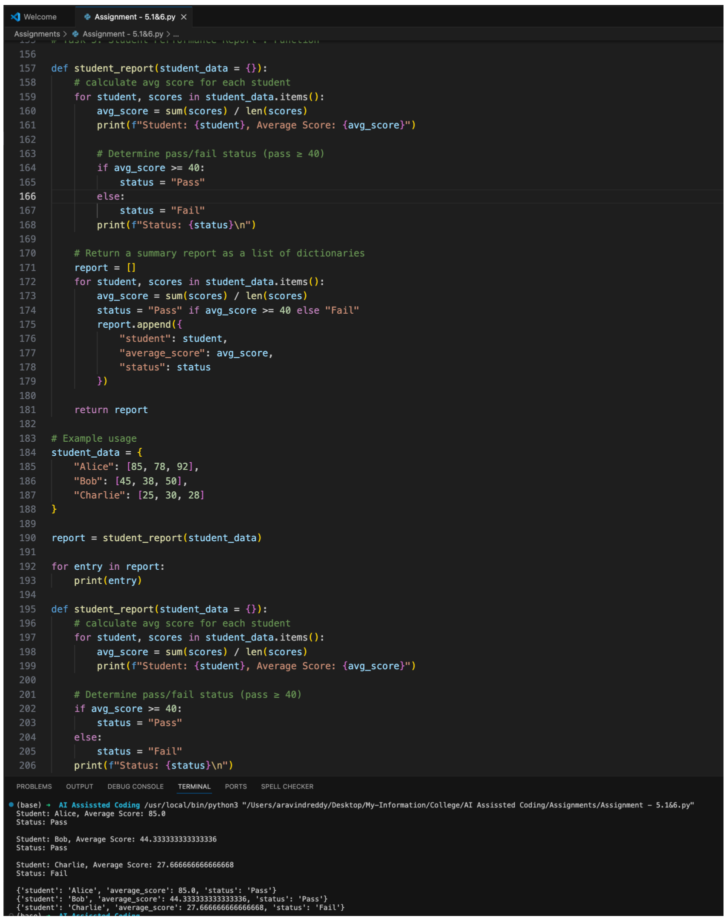
- Calculate the average score for each student

- 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.



```python

def student_report(student_data = {}):
    # calculate avg score for each student
    for student, scores in student_data.items():
        avg_score = sum(scores) / len(scores)
        print(f"Student: {student}, Average Score: {avg_score}")

        # Determine pass/fail status (pass ≥ 40)
        if avg_score >= 40:
            status = "Pass"
        else:
            status = "Fail"
        print(f"Status: {status}\n")

    # Return a summary report as a list of dictionaries
    report = []
    for student, scores in student_data.items():
        avg_score = sum(scores) / len(scores)
        status = "Pass" if avg_score >= 40 else "Fail"
        report.append({
            "student": student,
            "average_score": avg_score,
            "status": status
        })

    return report

# Example usage
student_data = {
    "Alice": [85, 78, 92],
    "Bob": [45, 38, 50],
    "Charlie": [25, 30, 28]
}

report = student_report(student_data)

for entry in report:
    print(entry)

def student_report(student_data = {}):
    # calculate avg score for each student
    for student, scores in student_data.items():
        avg_score = sum(scores) / len(scores)
        print(f"Student: {student}, Average Score: {avg_score}")

    # Determine pass/fail status (pass ≥ 40)
    if avg_score >= 40:
        status = "Pass"
    else:
        status = "Fail"
    print(f"Status: {status}\n")
```

```
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
Student: Alice, Average Score: 85.0
Status: Pass

Student: Bob, Average Score: 44.333333333333336
Status: Pass

Student: Charlie, Average Score: 27.666666666666668
Status: Fail

{'student': 'Alice', 'average_score': 85.0, 'status': 'Pass'}
{'student': 'Bob', 'average_score': 44.333333333333336, 'status': 'Pass'}
{'student': 'Charlie', 'average_score': 27.666666666666668, 'status': 'Fail'}
```

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:

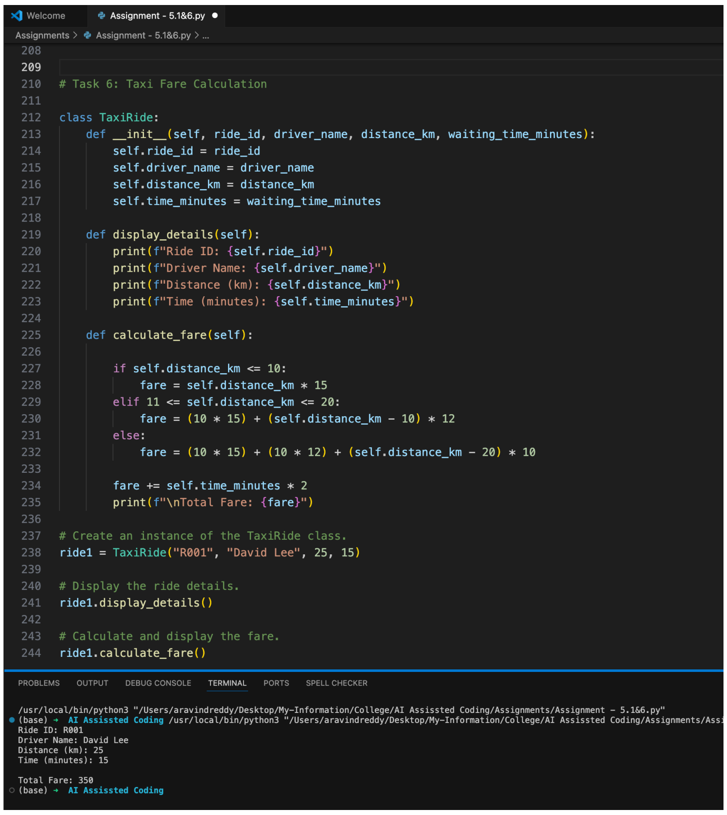
- ₹15 per km for the first 10 km

- ₹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

# Task 6: Taxi Fare Calculation

class TaxiRide:
    def __init__(self, ride_id, driver_name, distance_km, waiting_time_minutes):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.time_minutes = waiting_time_minutes

    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"Time (minutes): {self.time_minutes}")

    def calculate_fare(self):

        if self.distance_km <= 10:
            fare = self.distance_km * 15
        elif 11 <= self.distance_km <= 20:
            fare = (10 * 15) + (self.distance_km - 10) * 12
        else:
            fare = (10 * 15) + (10 * 12) + (self.distance_km - 20) * 10

        fare += self.time_minutes * 2
        print(f"\nTotal Fare: {fare}")

# Create an instance of the TaxiRide class.
ride1 = TaxiRide("R001", "David Lee", 25, 15)

# Display the ride details.
ride1.display_details()

# Calculate and display the fare.
ride1.calculate_fare()
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assi
Ride ID: R001
Driver Name: David Lee
Distance (km): 25
Time (minutes): 15

Total Fare: 350
(base) → AI Assissted Coding
```

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

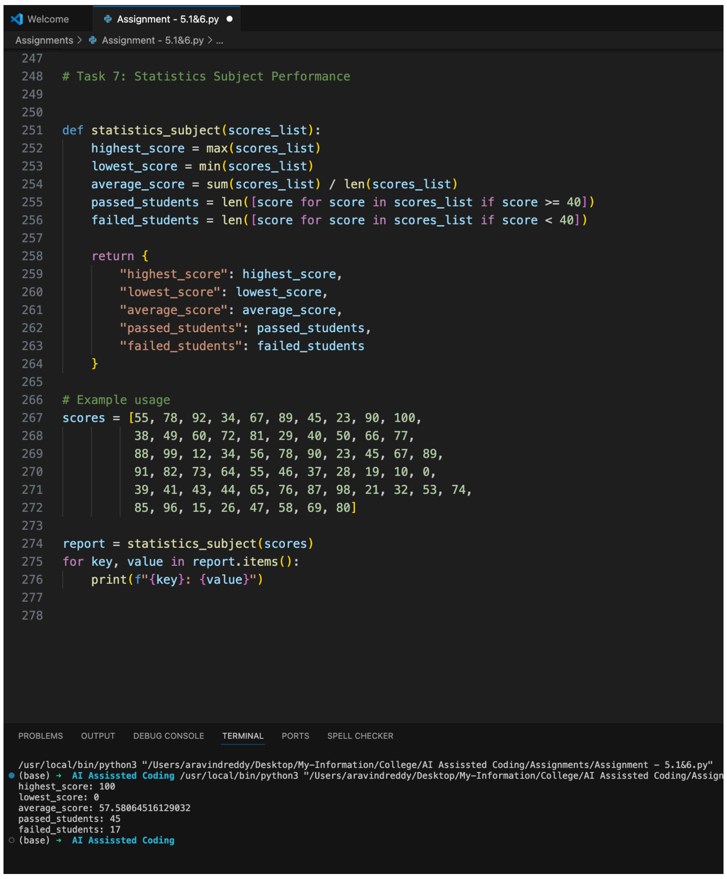
- Lowest score in the class

- Class average score

- Number of students passed (score ≥ 40)

- Number of students failed (score < 40)

Allow Copilot to assist with aggregations and logic



```python

# Task 7: Statistics Subject Performance


def statistics_subject(scores_list):
    highest_score = max(scores_list)
    lowest_score = min(scores_list)
    average_score = sum(scores_list) / len(scores_list)
    passed_students = len([score for score in scores_list if score >= 40])
    failed_students = len([score for score in scores_list if score < 40])

    return {
        "highest_score": highest_score,
        "lowest_score": lowest_score,
        "average_score": average_score,
        "passed_students": passed_students,
        "failed_students": failed_students
    }

# Example usage
scores = [55, 78, 92, 34, 67, 89, 45, 23, 90, 100,
          38, 49, 60, 72, 81, 29, 40, 50, 66, 77,
          88, 99, 12, 34, 56, 78, 90, 23, 45, 67, 89,
          91, 82, 73, 64, 55, 46, 37, 28, 19, 10, 0,
          39, 41, 43, 44, 65, 76, 87, 98, 21, 32, 53, 74,
          85, 96, 15, 26, 47, 58, 69, 80]

report = statistics_subject(scores)
for key, value in report.items():
    print(f"{key}: {value}")


```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assign
highest_score: 100
lowest_score: 0
average_score: 57.58064516129032
passed_students: 45
failed_students: 17
○ (base) → AI Assissted Coding
```

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

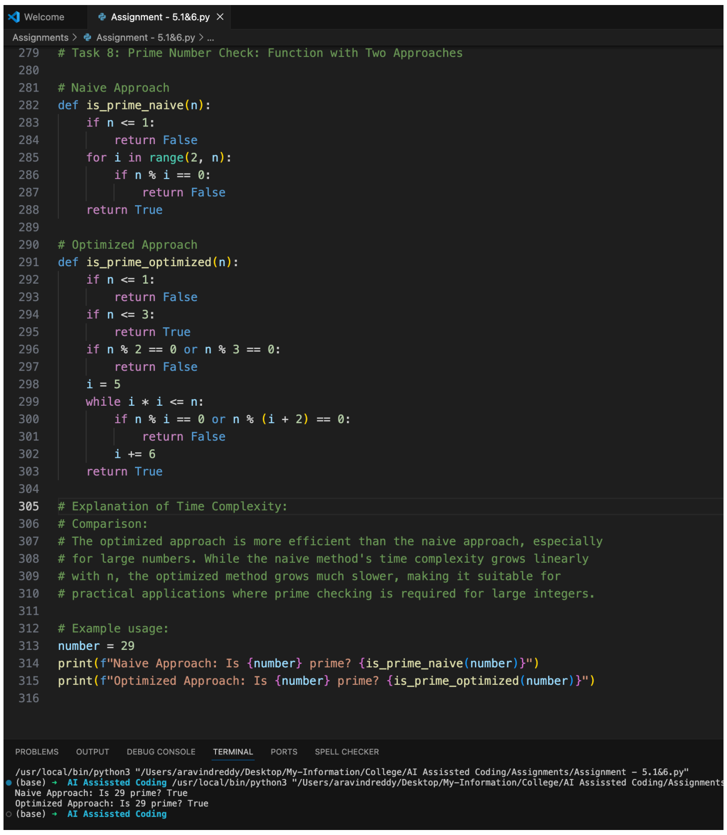
how the optimized version improves performance.”

Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.



```python
# Task 8: Prime Number Check: Function with Two Approaches

# Naive Approach
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Optimized Approach
def is_prime_optimized(n):
    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

# Explanation of Time Complexity:
# Comparison:
# The optimized approach is more efficient than the naive approach, especially
# for large numbers. While the naive method's time complexity grows linearly
# with n, the optimized method grows much slower, making it suitable for
# practical applications where prime checking is required for large integers.

# Example usage:
number = 29
print(f"Naive Approach: Is {number} prime? {is_prime_naive(number)}")
print(f"Optimized Approach: Is {number} prime? {is_prime_optimized(number)}")
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments
Naive Approach: Is 29 prime? True
Optimized Approach: Is 29 prime? True
○ (base) → AI Assissted Coding
```

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.

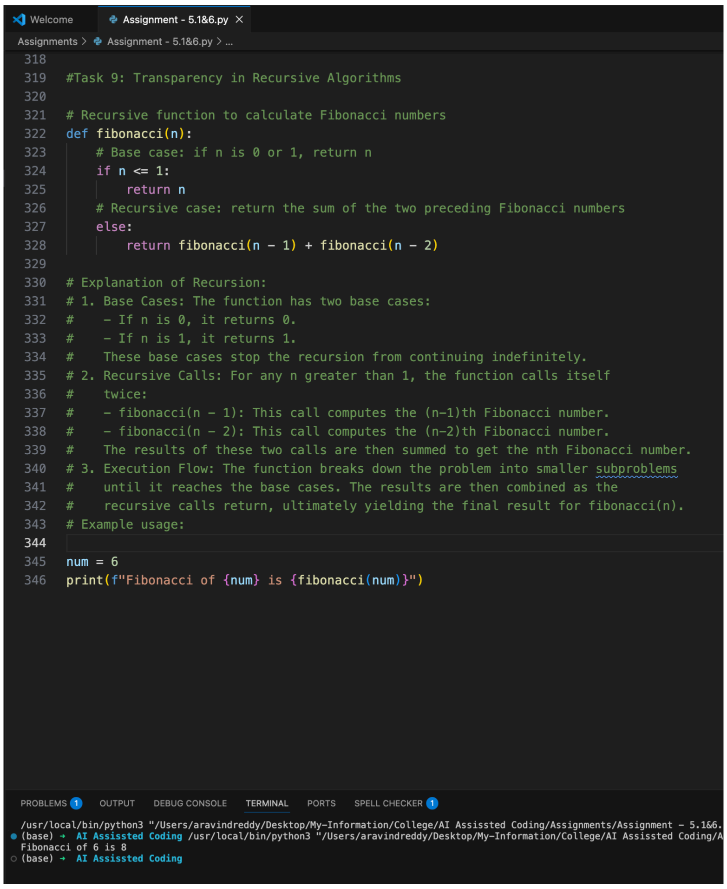
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.



```python

#Task 9: Transparency in Recursive Algorithms

# Recursive function to calculate Fibonacci numbers
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 1:
        return n
    # Recursive case: return the sum of the two preceding Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Explanation of Recursion:
# 1. Base Cases: The function has two base cases:
#    - If n is 0, it returns 0.
#    - If n is 1, it returns 1.
#    These base cases stop the recursion from continuing indefinitely.
# 2. Recursive Calls: For any n greater than 1, the function calls itself
#    twice:
#    - fibonacci(n - 1): This call computes the (n-1)th Fibonacci number.
#    - fibonacci(n - 2): This call computes the (n-2)th Fibonacci number.
#    The results of these two calls are then summed to get the nth Fibonacci number.
# 3. Execution Flow: The function breaks down the problem into smaller subproblems
#    until it reaches the base cases. The results are then combined as the
#    recursive calls return, ultimately yielding the final result for fibonacci(n).
# Example usage:

num = 6
print(f"Fibonacci of {num} is {fibonacci(num)}")
```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/A
Fibonacci of 6 is 8
○ (base) → AI Assissted Coding
```

Task Description #10 (Transparency in Error Handling)

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

Prompt:

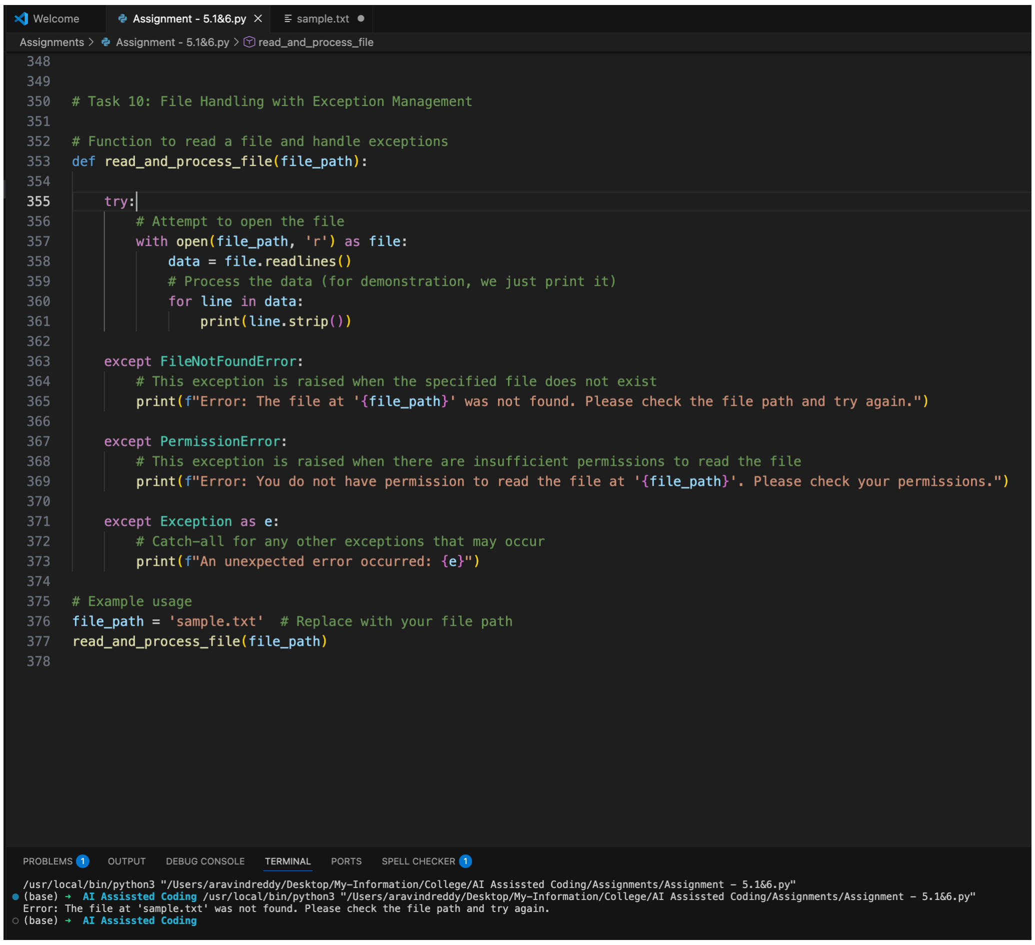
“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

• Validation that explanations align with runtime behavior.



```python


# Task 10: File Handling with Exception Management

# Function to read a file and handle exceptions
def read_and_process_file(file_path):

    try:
        # Attempt to open the file
        with open(file_path, 'r') as file:
            data = file.readlines()
            # Process the data (for demonstration, we just print it)
            for line in data:
                print(line.strip())

    except FileNotFoundError:
        # This exception is raised when the specified file does not exist
        print(f"Error: The file at '{file_path}' was not found. Please check the file path and try again.")

    except PermissionError:
        # This exception is raised when there are insufficient permissions to read the file
        print(f"Error: You do not have permission to read the file at '{file_path}'. Please check your permissions.")

    except Exception as e:
        # Catch-all for any other exceptions that may occur
        print(f"An unexpected error occurred: {e}")

# Example usage
file_path = 'sample.txt'  # Replace with your file path
read_and_process_file(file_path)

```

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 5.1&6.py"
Error: The file at 'sample.txt' was not found. Please check the file path and try again.
(base) → AI Assissted Coding
```