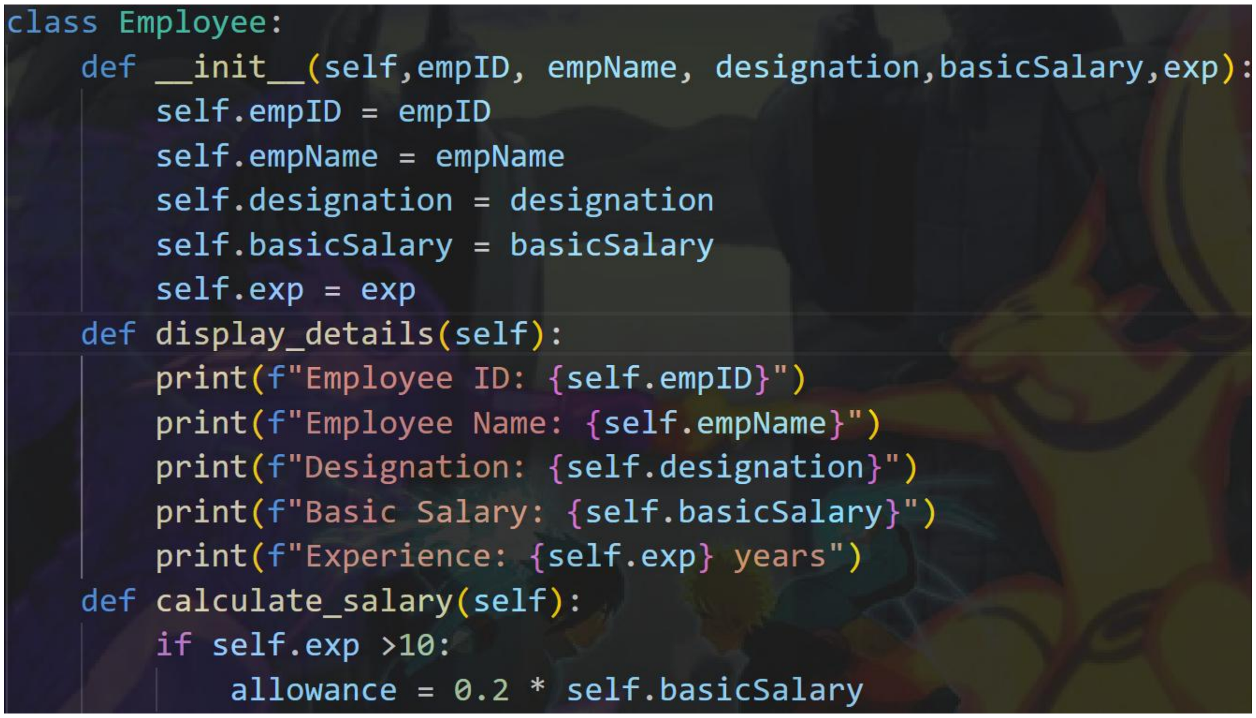
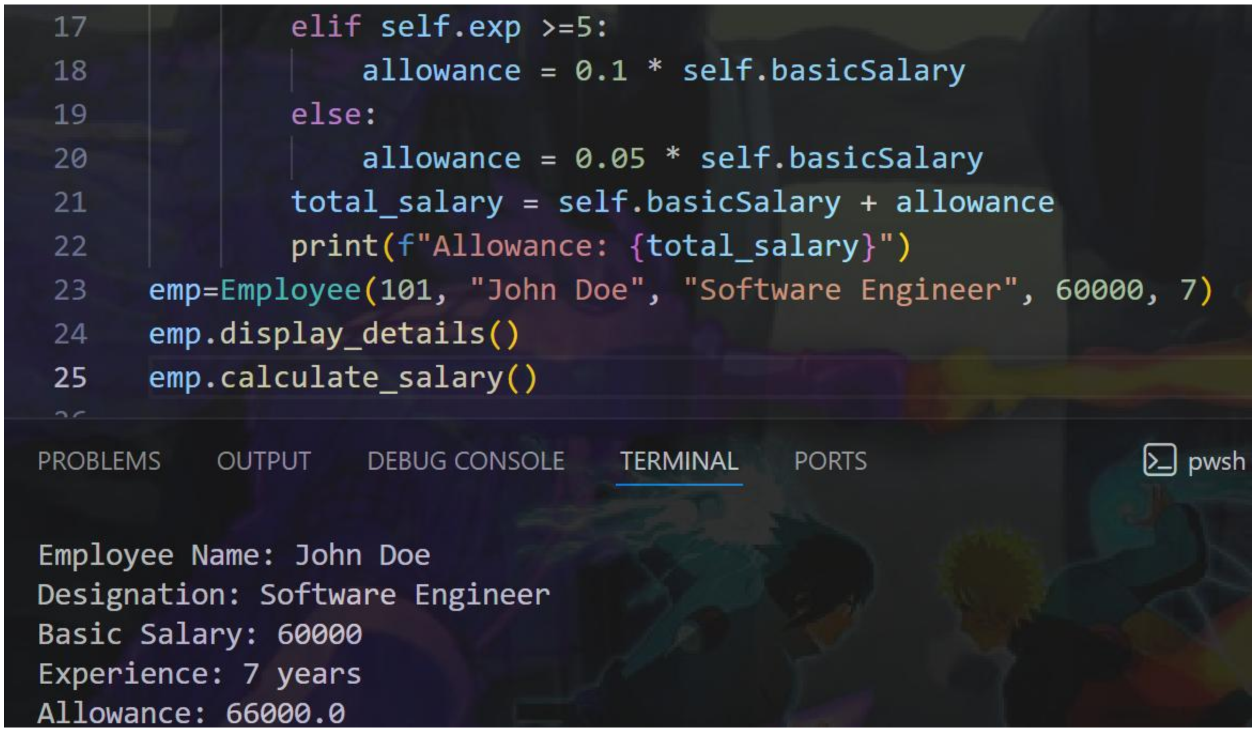
Name: B. Srikar Batch: 02
Course: AI ASSISSTED CODING
Semester: Even

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.

CODE AND OUTPUT:



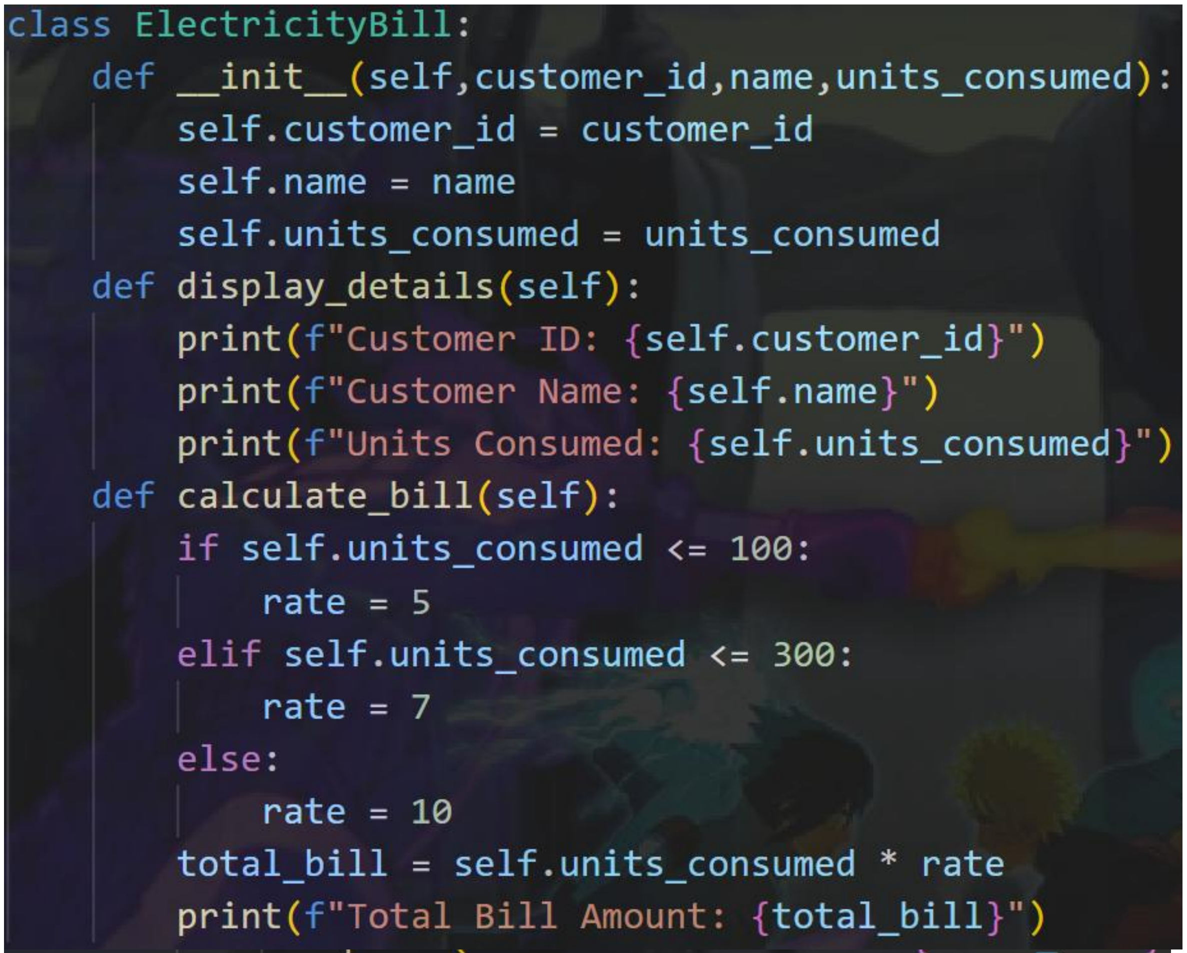
```
class Employee:
    def __init__(self,empID, empName, designation,basicSalary,exp):
        self.empID = empID
        self.empName = empName
        self.designation = designation
        self.basicSalary = basicSalary
        self.exp = exp
    def display_details(self):
        print(f"Employee ID: {self.empID}")
        print(f"Employee Name: {self.empName}")
        print(f"Designation: {self.designation}")
        print(f"Basic Salary: {self.basicSalary}")
        print(f"Experience: {self.exp} years")
    def calculate_salary(self):
        if self.exp >10:
            allowance = 0.2 * self.basicSalary
```


```python
            elif self.exp >=5:
                allowance = 0.1 * self.basicSalary
            else:
                allowance = 0.05 * self.basicSalary
            total_salary = self.basicSalary + allowance
            print(f"Allowance: {total_salary}")
emp=Employee(101, "John Doe", "Software Engineer", 60000, 7)
emp.display_details()
emp.calculate_salary()
```

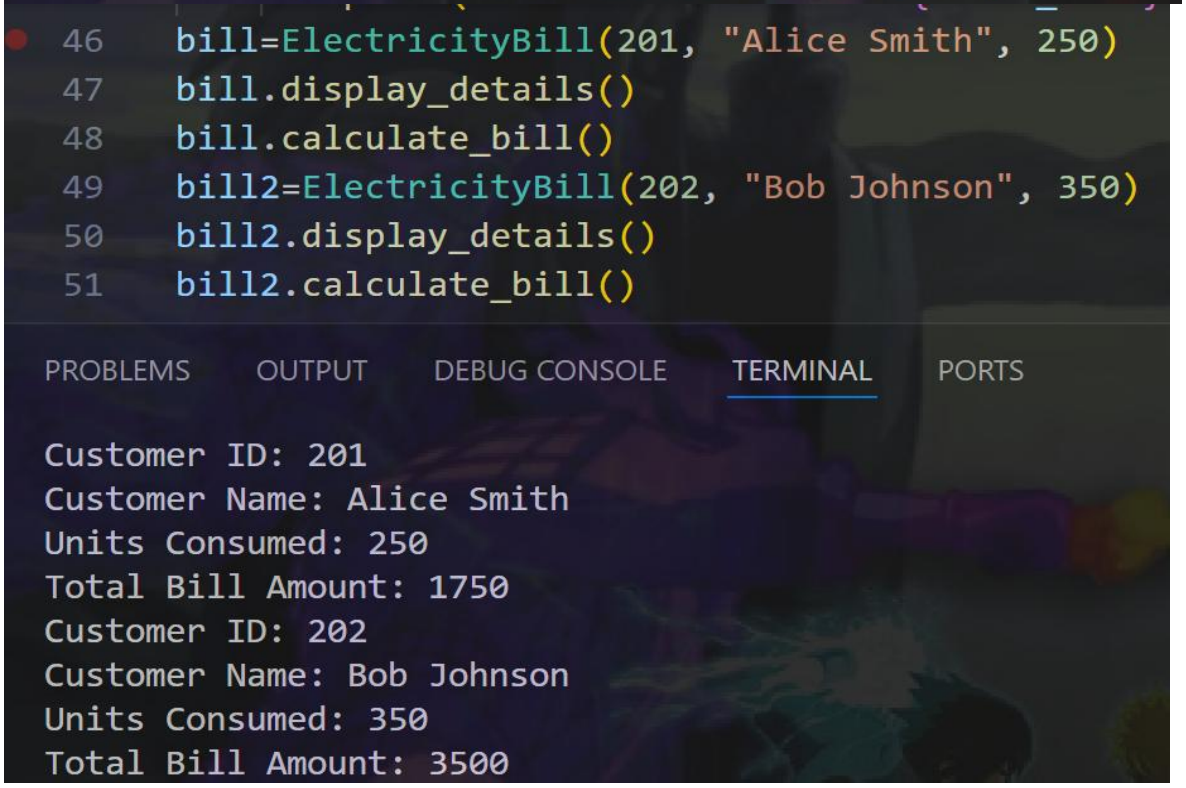
```
Employee Name: John Doe
Designation: Software Engineer
Basic Salary: 60000
Experience: 7 years
Allowance: 66000.0
```

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.

<u>CODE AND OUTPUT:</u>



```python
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(f"Customer ID: {self.customer_id}")
        print(f"Customer Name: {self.name}")
        print(f"Units Consumed: {self.units_consumed}")
    def calculate_bill(self):
        if self.units_consumed <= 100:
            rate = 5
        elif self.units_consumed <= 300:
            rate = 7
        else:
            rate = 10
        total_bill = self.units_consumed * rate
        print(f"Total Bill Amount: {total_bill}")
```

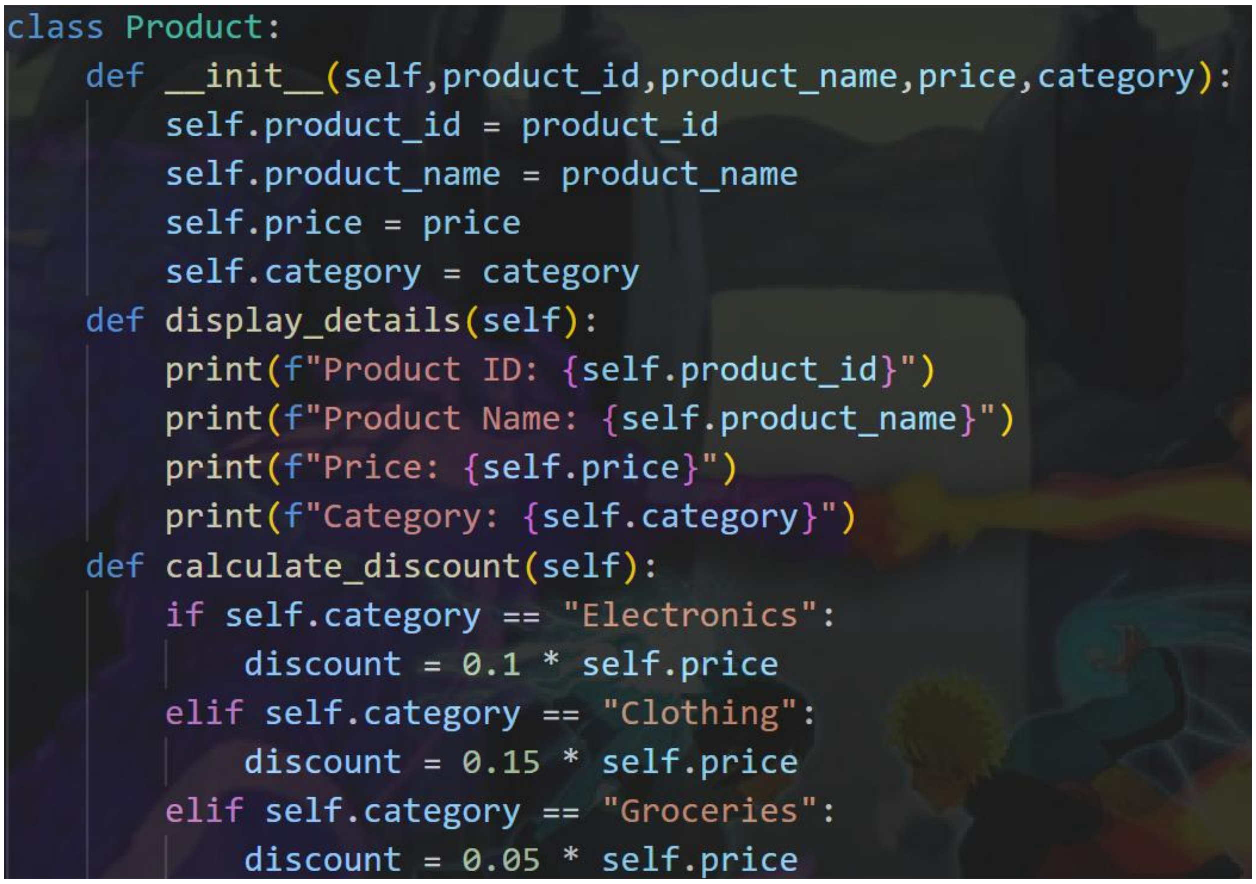


```python
bill=ElectricityBill(201, "Alice Smith", 250)
bill.display_details()
bill.calculate_bill()
bill2=ElectricityBill(202, "Bob Johnson", 350)
bill2.display_details()
bill2.calculate_bill()
```

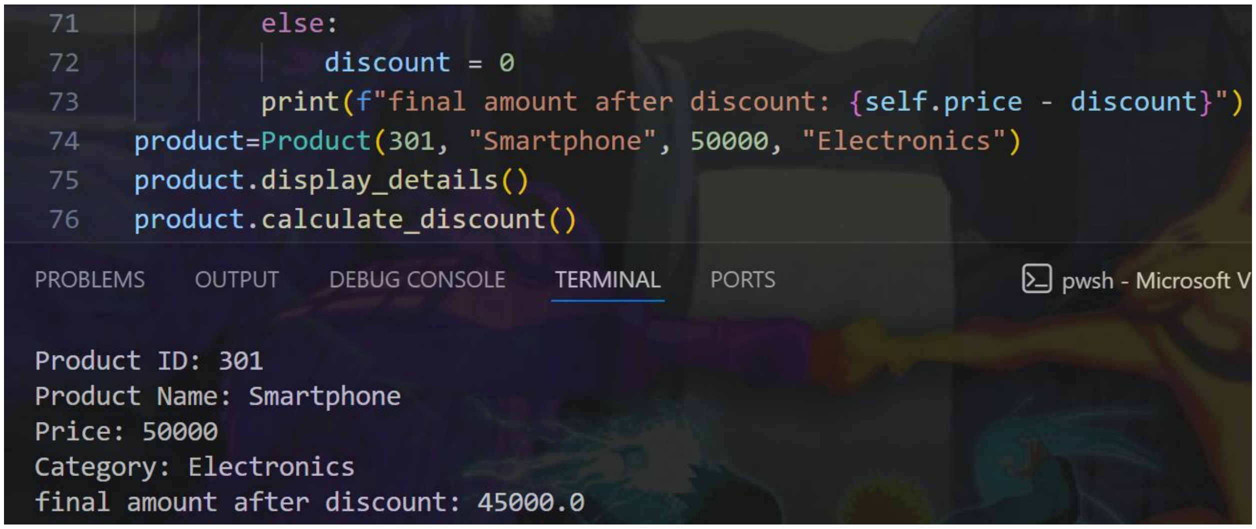
```
Customer ID: 201
Customer Name: Alice Smith
Units Consumed: 250
Total Bill Amount: 1750
Customer ID: 202
Customer Name: Bob Johnson
Units Consumed: 350
Total Bill Amount: 3500
```

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.

<u>**CODE AND OUTPUT:**</u>



```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 == "Electronics":
            discount = 0.1 * self.price
        elif self.category == "Clothing":
            discount = 0.15 * self.price
        elif self.category == "Groceries":
            discount = 0.05 * self.price
```

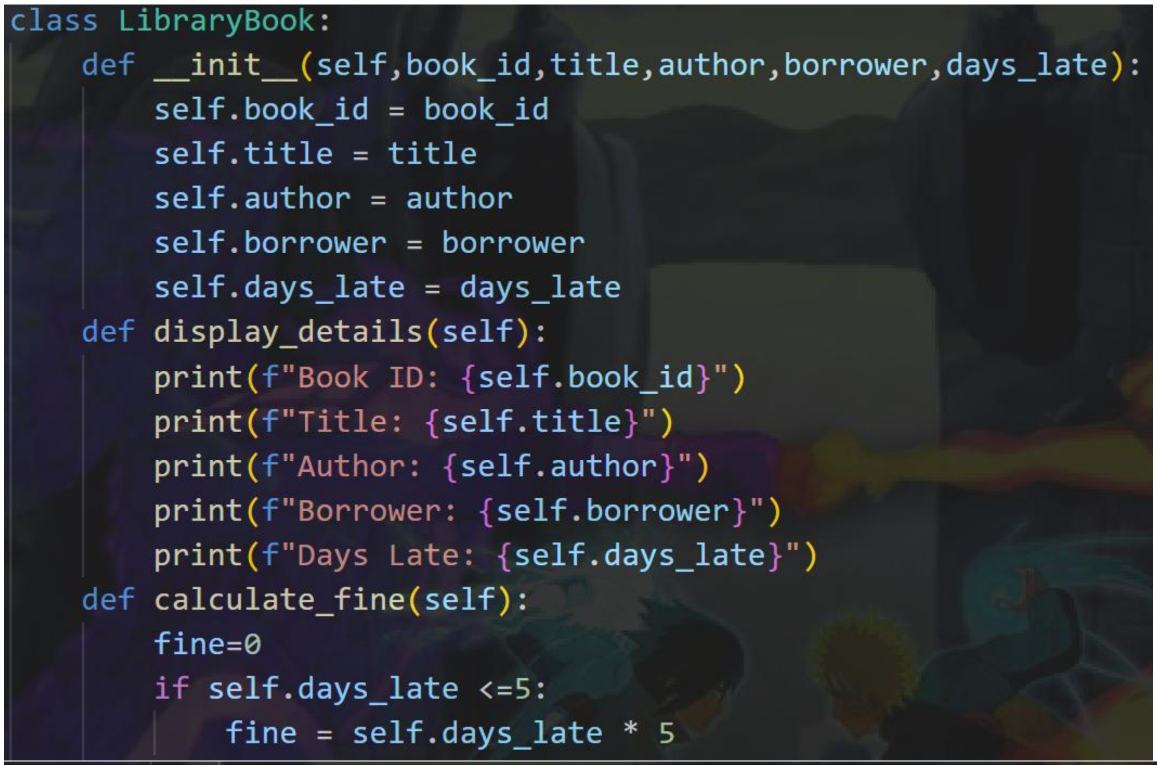


```
            else:
                discount = 0
            print(f"final amount after discount: {self.price - discount}")
product=Product(301, "Smartphone", 50000, "Electronics")
product.display_details()
product.calculate_discount()
```

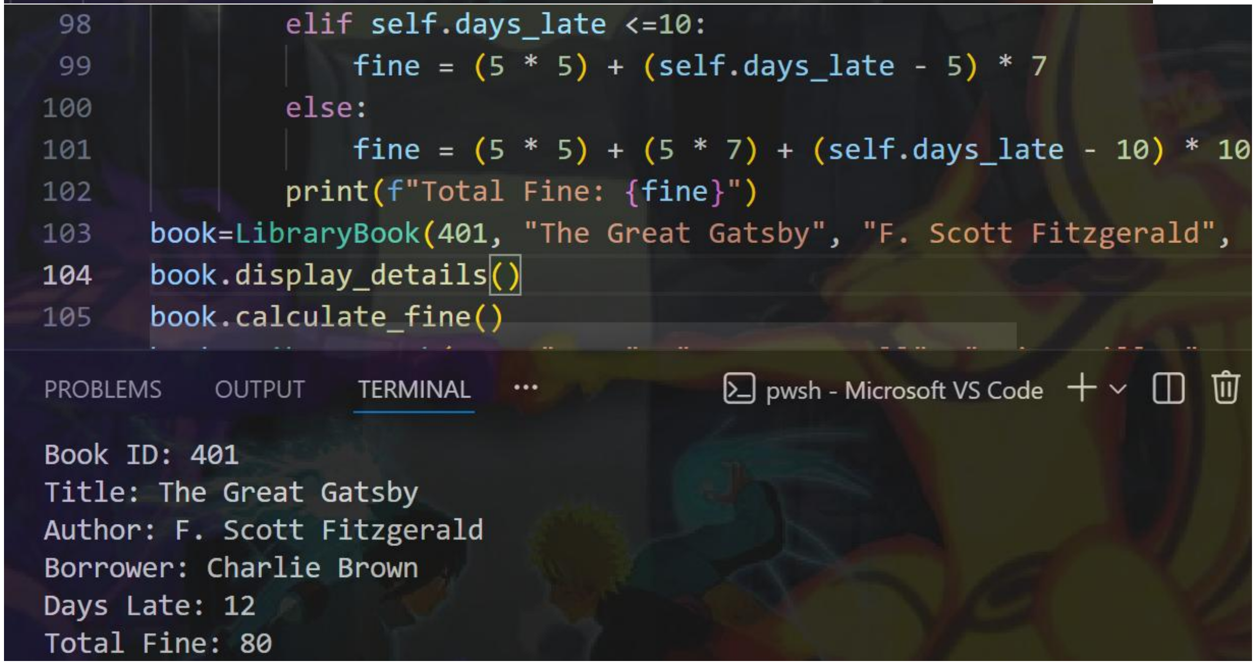
```
Product ID: 301
Product Name: Smartphone
Price: 50000
Category: Electronics
final amount after discount: 45000.0
```

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.

<u>CODE AND OUTPUT:</u>



```python
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_fine(self):
        fine=0
        if self.days_late <=5:
            fine = self.days_late * 5
```



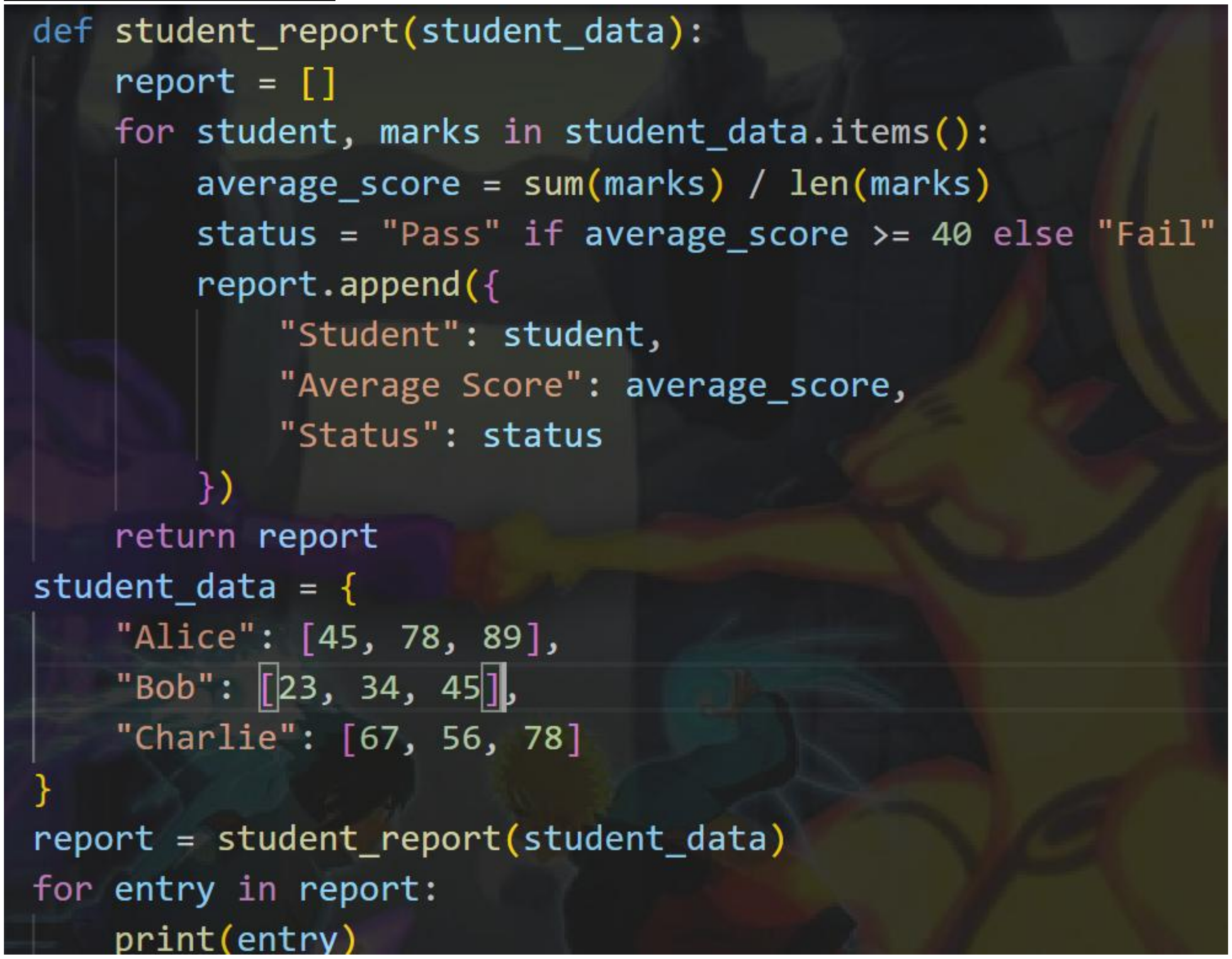
```python
        elif self.days_late <=10:
            fine = (5 * 5) + (self.days_late - 5) * 7
        else:
            fine = (5 * 5) + (5 * 7) + (self.days_late - 10) * 10
        print(f"Total Fine: {fine}")
book=LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald",
book.display_details()
book.calculate_fine()
```

```
Book ID: 401
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Charlie Brown
Days Late: 12
Total Fine: 80
```

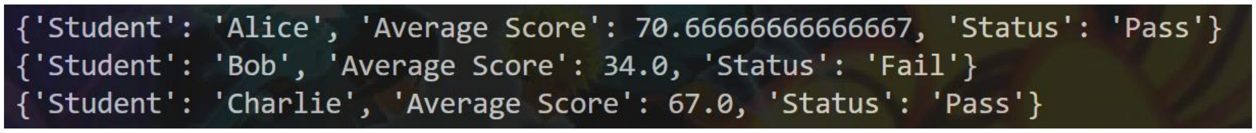
Task 5:
Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing

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.

<u>CODE AND OUTPUT:</u>



```python
def student_report(student_data):
    report = []
    for student, marks in student_data.items():
        average_score = sum(marks) / len(marks)
        status = "Pass" if average_score >= 40 else "Fail"
        report.append({
            "Student": student,
            "Average Score": average_score,
            "Status": status
        })
    return report
student_data = {
    "Alice": [45, 78, 89],
    "Bob": [23, 34, 45],
    "Charlie": [67, 56, 78]
}
report = student_report(student_data)
for entry in report:
    print(entry)
```

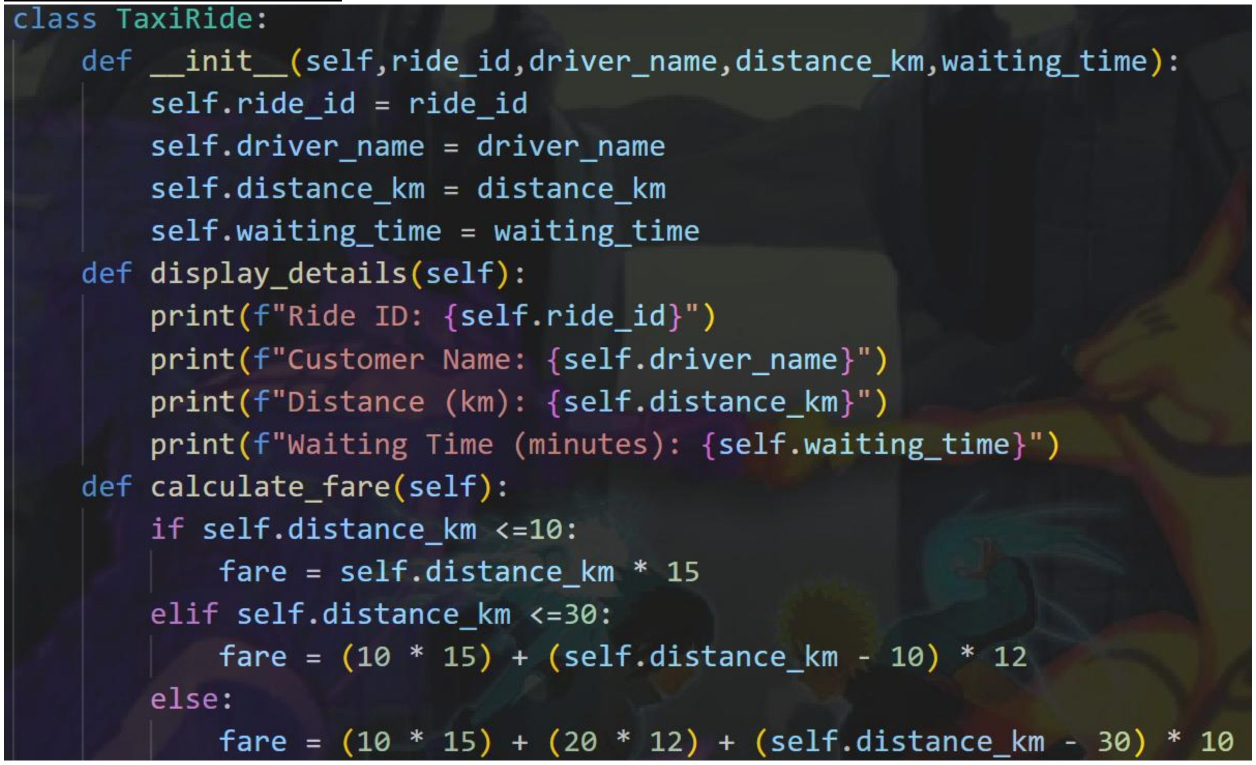


```
{'Student': 'Alice', 'Average Score': 70.66666666666667, 'Status': 'Pass'}
{'Student': 'Bob', 'Average Score': 34.0, 'Status': 'Fail'}
{'Student': 'Charlie', 'Average Score': 67.0, 'Status': 'Pass'}
```

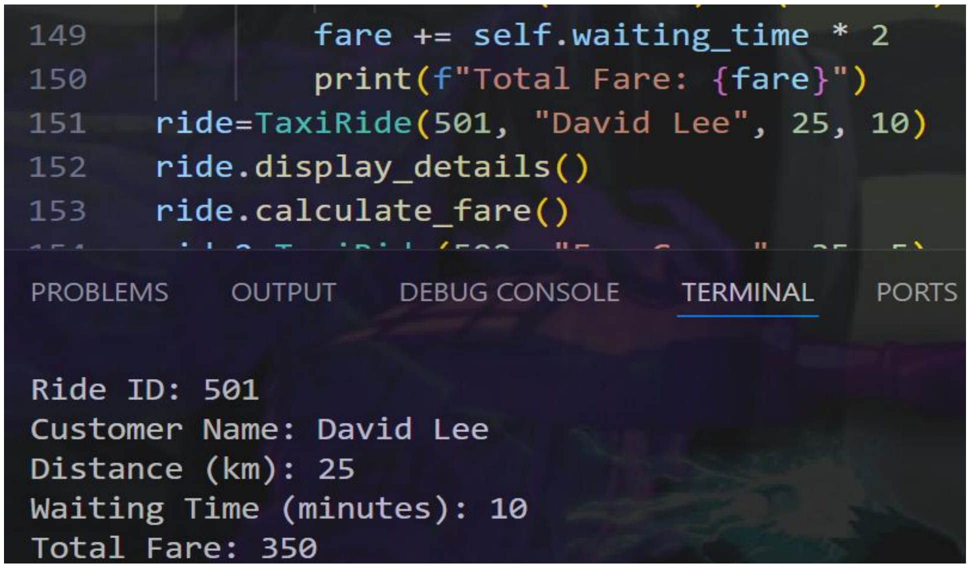
Task 6:
Taxi Fare Calculation-Create Python code that defines a class named

`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.

CODE AND OUTPUT:



```python
class TaxiRide:
    def __init__(self,ride_id,driver_name,distance_km,waiting_time):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.waiting_time = waiting_time
    def display_details(self):
        print(f"Ride ID: {self.ride_id}")
        print(f"Customer Name: {self.driver_name}")
        print(f"Distance (km): {self.distance_km}")
        print(f"Waiting Time (minutes): {self.waiting_time}")
    def calculate_fare(self):
        if self.distance_km <=10:
            fare = self.distance_km * 15
        elif self.distance_km <=30:
            fare = (10 * 15) + (self.distance_km - 10) * 12
        else:
            fare = (10 * 15) + (20 * 12) + (self.distance_km - 30) * 10
```



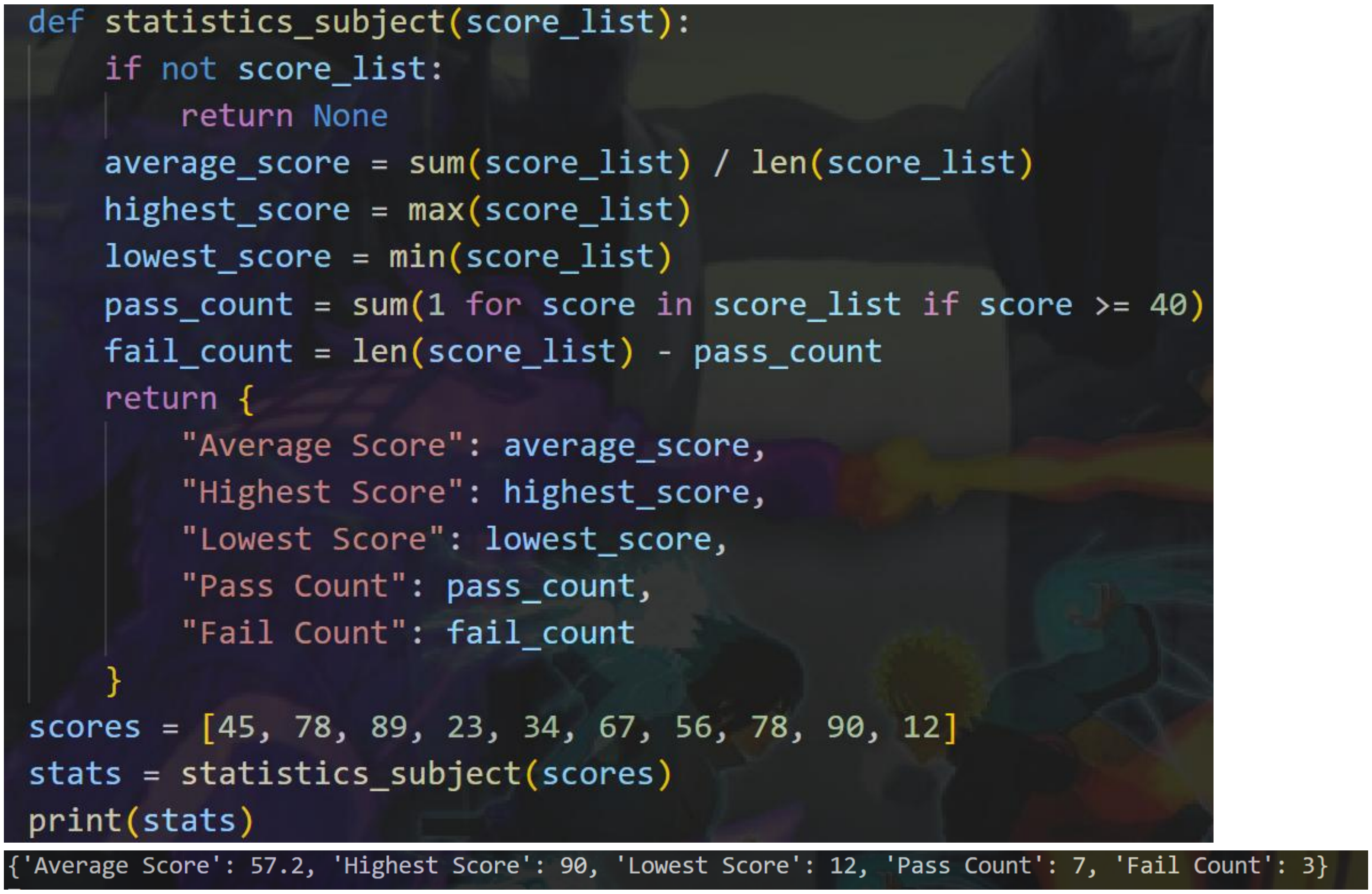
```
            fare += self.waiting_time * 2
            print(f"Total Fare: {fare}")
ride=TaxiRide(501, "David Lee", 25, 10)
ride.display_details()
ride.calculate_fare()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Ride ID: 501
Customer Name: David Lee
Distance (km): 25
Waiting Time (minutes): 10
Total Fare: 350
```

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

CODE AND OUTPUT:



```python
def statistics_subject(score_list):
    if not score_list:
        return None
    average_score = sum(score_list) / len(score_list)
    highest_score = max(score_list)
    lowest_score = min(score_list)
    pass_count = sum(1 for score in score_list if score >= 40)
    fail_count = len(score_list) - pass_count
    return {
        "Average Score": average_score,
        "Highest Score": highest_score,
        "Lowest Score": lowest_score,
        "Pass Count": pass_count,
        "Fail Count": fail_count
    }
scores = [45, 78, 89, 23, 34, 67, 56, 78, 90, 12]
stats = statistics_subject(scores)
print(stats)
```

```
{'Average Score': 57.2, 'Highest Score': 90, 'Lowest Score': 12, 'Pass Count': 7, 'Fail Count': 3}
```

Task Description #8 (Transparency in Algorithm Optimization)

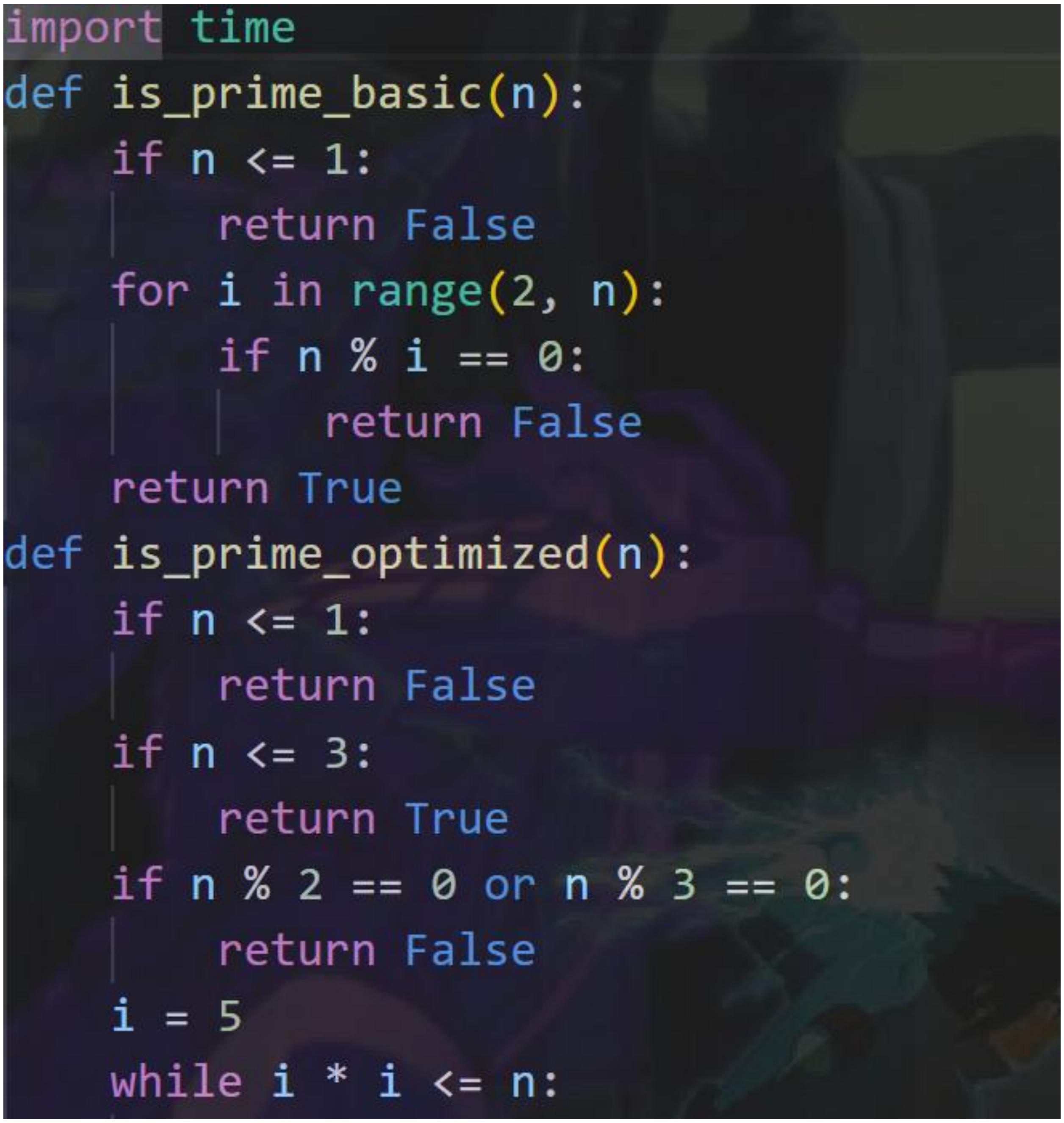
Task: Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

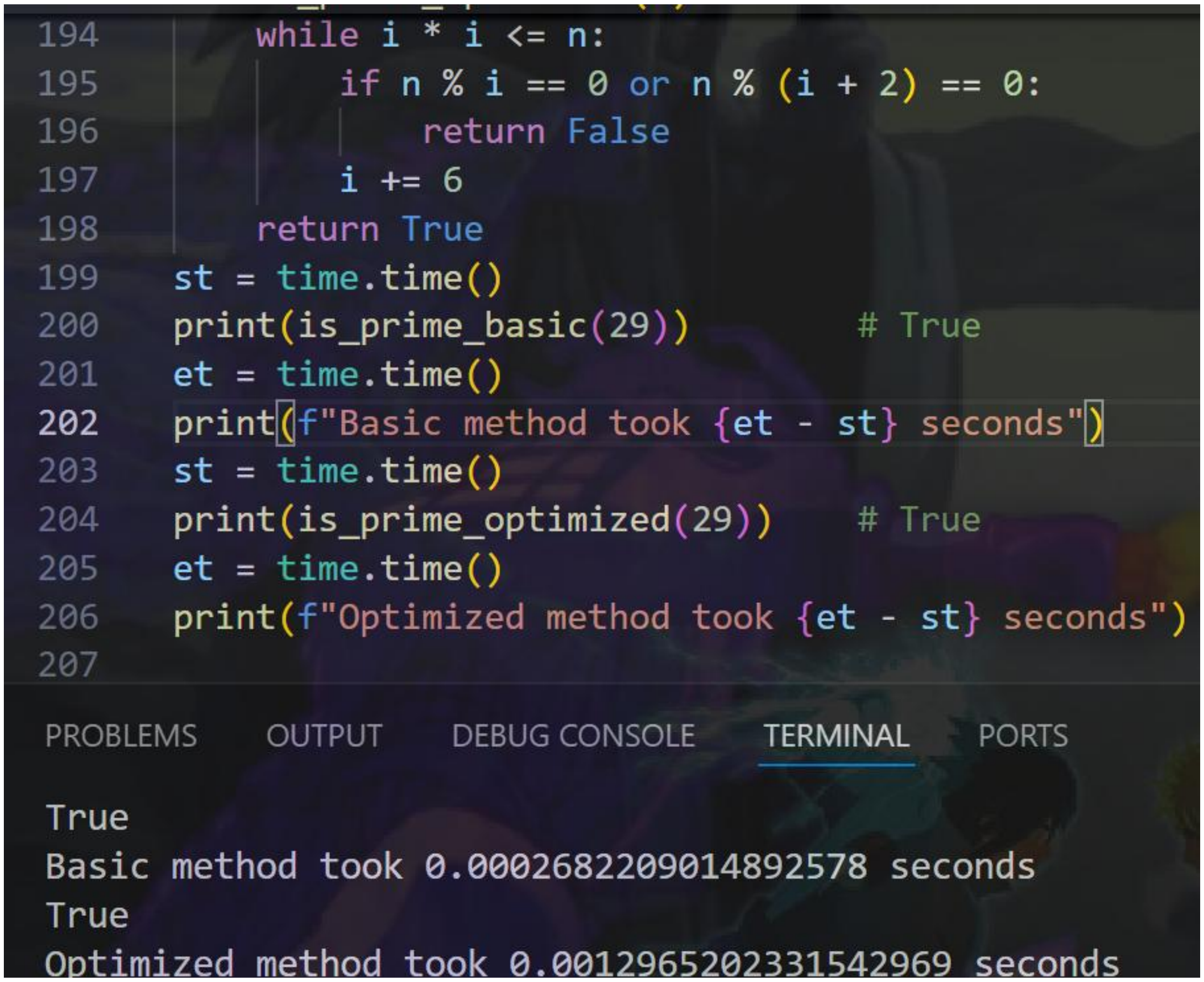
Prompt:

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

Code and Output:



```python
import time
def is_prime_basic(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
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:
```



```python
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
st = time.time()
print(is_prime_basic(29))         # True
et = time.time()
print(f"Basic method took {et - st} seconds")
st = time.time()
print(is_prime_optimized(29))     # True
et = time.time()
print(f"Optimized method took {et - st} seconds")
```

```
True
Basic method took 0.0002682209014892578 seconds
True
Optimized method took 0.0012965202331542969 seconds
```

Explanation:
The basic method checks for factors from 2 to n-1, which can be slow for large n.
The optimized method reduces the number of checks by:
1. Eliminating even numbers and multiples of 3 right away.
2. Only checking up to the square root of n, since a larger factor of n must be a multiple of a smaller factor that has already been checked.
3. Checking only numbers of the form 6k ± 1, as all primes greater than 3 can be expressed in this form. This significantly reduces the number of iterations needed to determine primality.

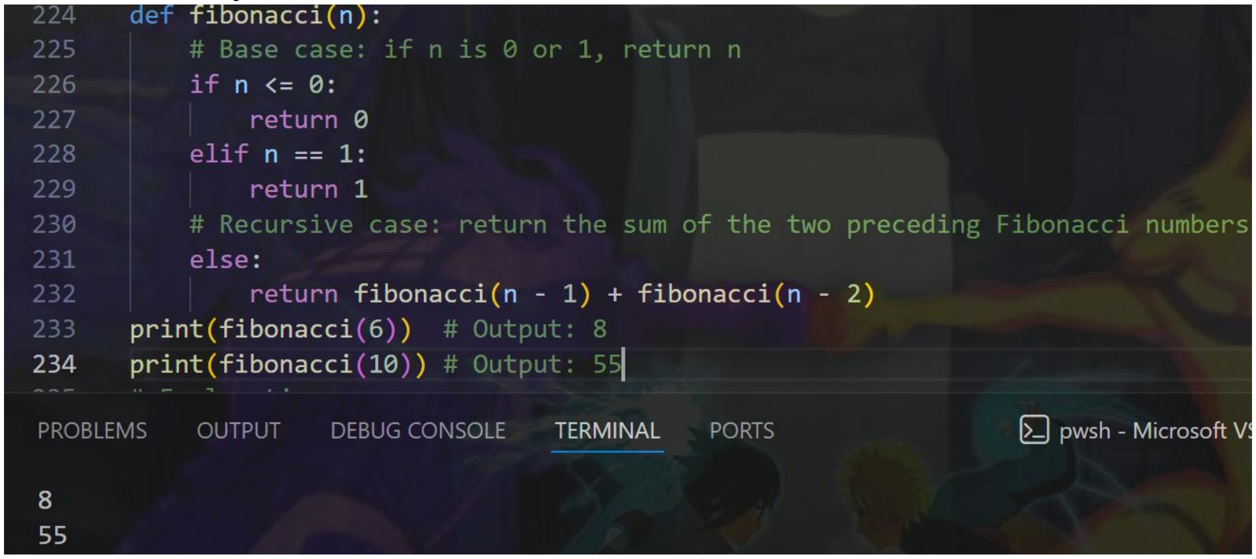
Task Description #9 (Transparency in Recursive Algorithms)

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

Prompt:

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

Code and Output:



```python
def fibonacci(n):
    # Base case: if n is 0 or 1, return n
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    # Recursive case: return the sum of the two preceding Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
print(fibonacci(6))  # Output: 8
print(fibonacci(10)) # Output: 55
```

```
8
55
```

Explanation:
In this recursive function, we define two base cases:
1. If n is 0, we return 0.
2. If n is 1, we return 1.
For any other value of n, we make two recursive calls:
1. fibonacci(n - 1): This call computes the (n-1)th Fibonacci number.
2. fibonacci(n - 2): This call computes the (n-2)th Fibonacci number.
The function then returns the sum of these two calls, effectively building up the Fibonacci sequence from the base cases upward.

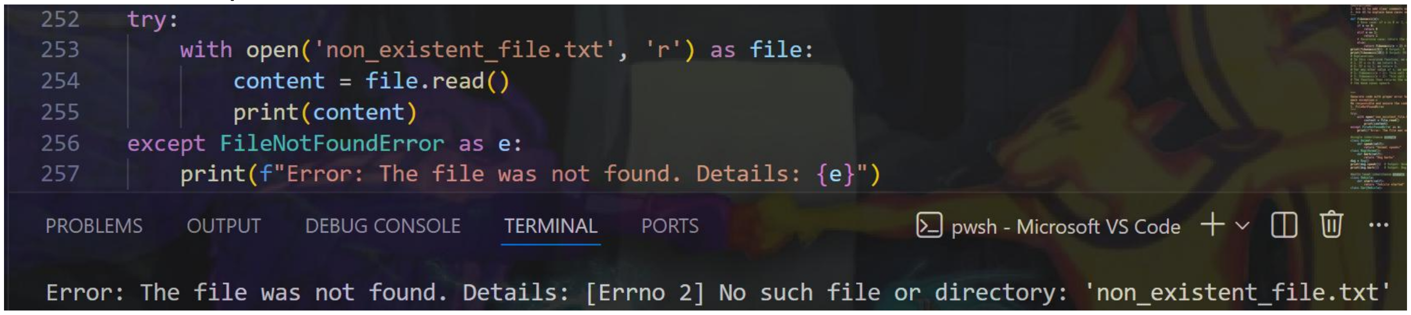
Task Description #10 (Transparency in Error Handling)

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.”

Code and Output:



```python
try:
    with open('non_existent_file.txt', 'r') as file:
        content = file.read()
        print(content)
except FileNotFoundError as e:
    print(f"Error: The file was not found. Details: {e}")
```

```
Error: The file was not found. Details: [Errno 2] No such file or directory: 'non_existent_file.txt'
```