Assignment- 6.1

Hall Ticket: 2303A51630

Name: K. Nithin Kumar

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:

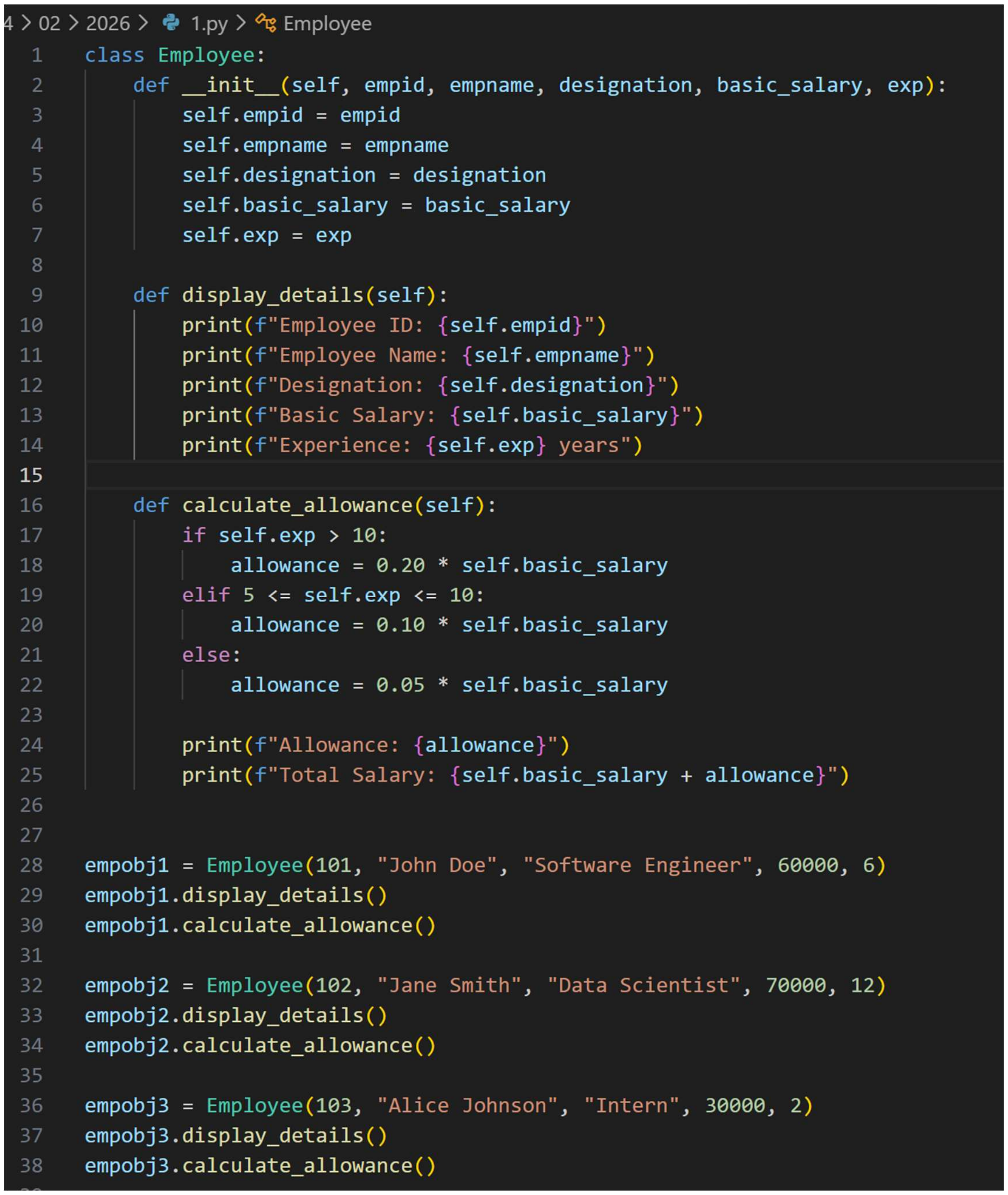
- If `exp > 10 years` → allowance = 20% of `basic_salary`

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

- If `exp < 5 years` → allowance = 5% of `basic_salary`

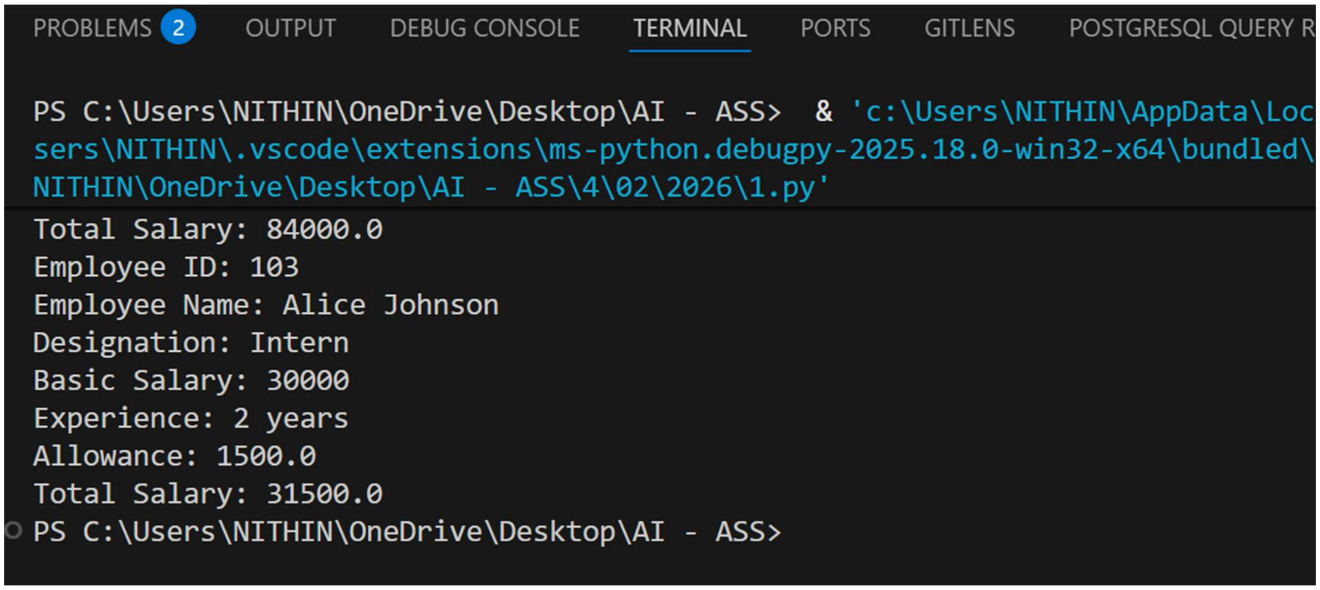
Finally, create at least one instance of the `Employee` class, call the

`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, "John Doe", "Software Engineer", 60000, 6)
empobj1.display_details()
empobj1.calculate_allowance()

empobj2 = Employee(102, "Jane Smith", "Data Scientist", 70000, 12)
empobj2.display_details()
empobj2.calculate_allowance()

empobj3 = Employee(103, "Alice Johnson", "Intern", 30000, 2)
empobj3.display_details()
empobj3.calculate_allowance()
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  & 'c:\Users\NITHIN\AppData\Loc
sers\NITHIN\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\
NITHIN\OneDrive\Desktop\AI - ASS\4\02\2026\1.py'
Total Salary: 84000.0
Employee ID: 103
Employee Name: Alice Johnson
Designation: Intern
Basic Salary: 30000
Experience: 2 years
Allowance: 1500.0
Total Salary: 31500.0
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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

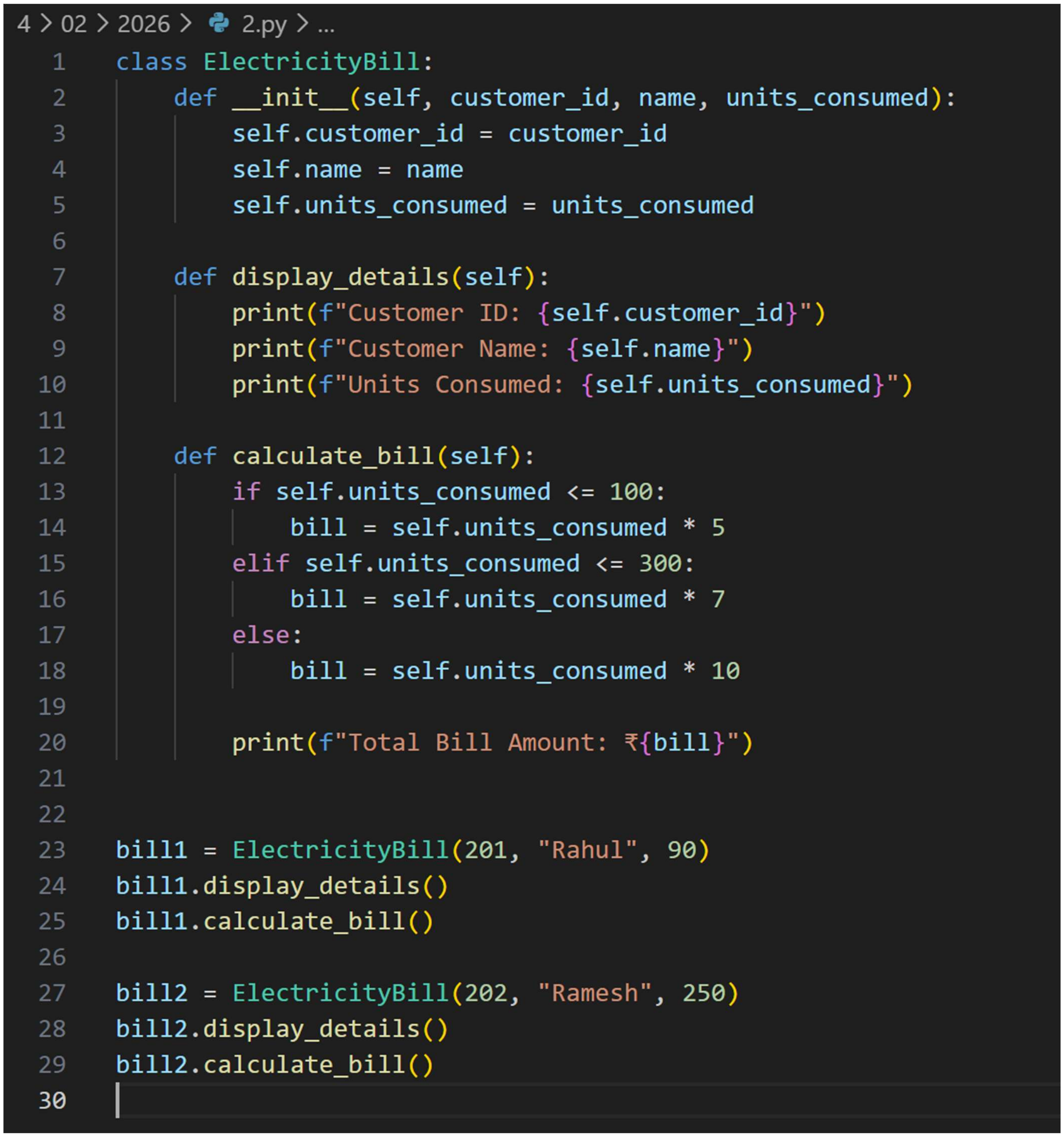
customer details, and a method `calculate_bill()` where:

- Units ≤ 100 → ₹5 per unit

- 101 to 300 units → ₹7 per unit

- More than 300 units → ₹10 per unit

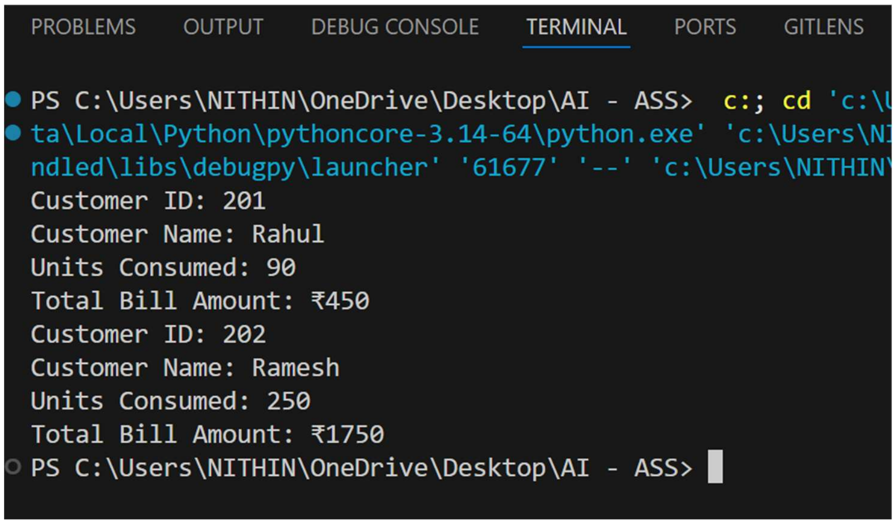
Create a bill object, display details, and print the total bill amount.



4 > 02 > 2026 > 2.py > ...

```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:
            bill = self.units_consumed * 5
        elif self.units_consumed <= 300:
            bill = self.units_consumed * 7
        else:
            bill = self.units_consumed * 10

        print(f"Total Bill Amount: ₹{bill}")


bill1 = ElectricityBill(201, "Rahul", 90)
bill1.display_details()
bill1.calculate_bill()

bill2 = ElectricityBill(202, "Ramesh", 250)
bill2.display_details()
bill2.calculate_bill()

```

OUPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\U
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NI
ndled\libs\debugpy\launcher' '61677' '--' 'c:\Users\NITHIN\
Customer ID: 201
Customer Name: Rahul
Units Consumed: 90
Total Bill Amount: ₹450
Customer ID: 202
Customer Name: Ramesh
Units Consumed: 250
Total Bill Amount: ₹1750
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```

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

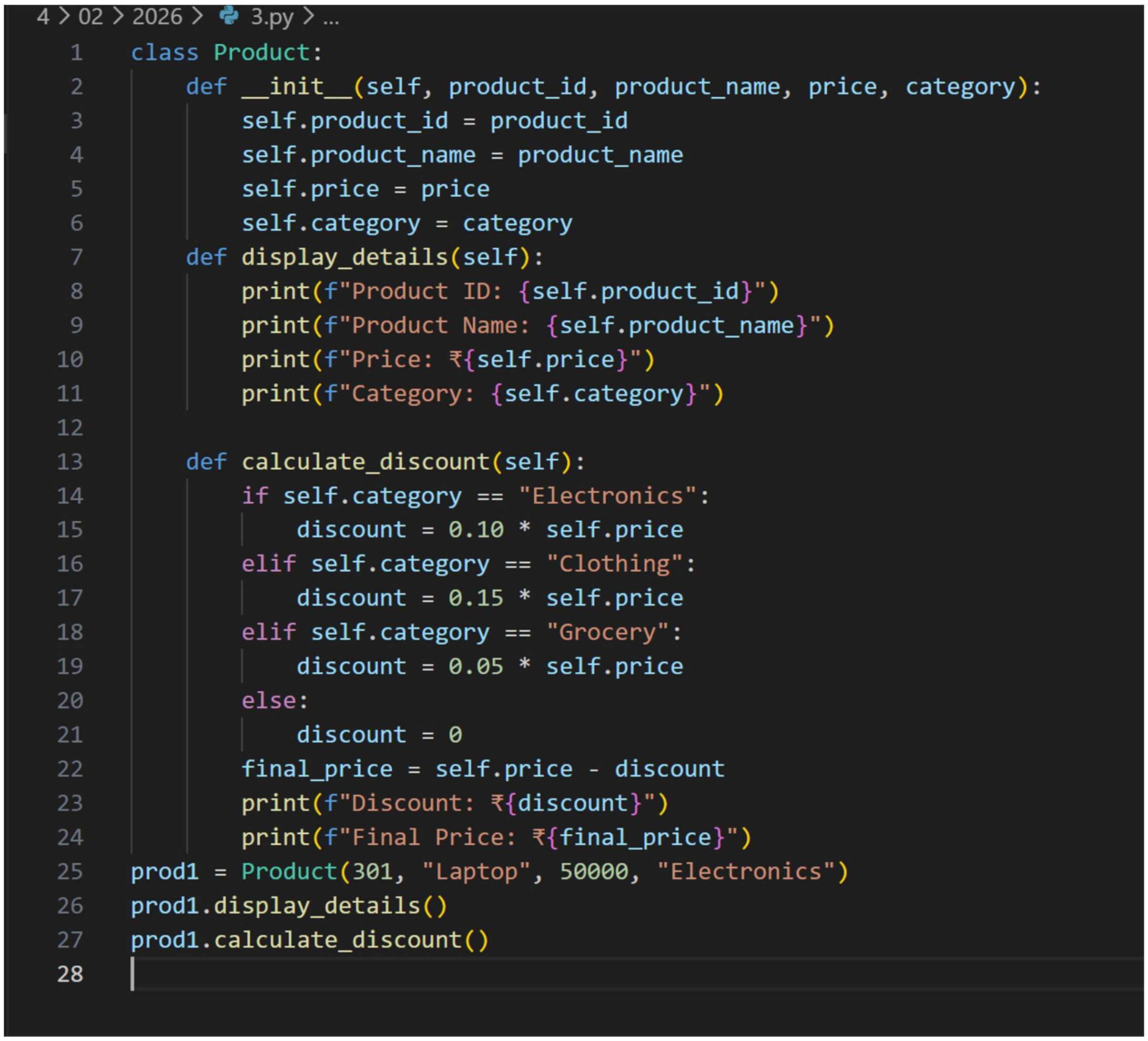
print product details. Implement another method `calculate_discount()` where:

- Electronics → 10% discount

- Clothing → 15% discount

- Grocery → 5% discount

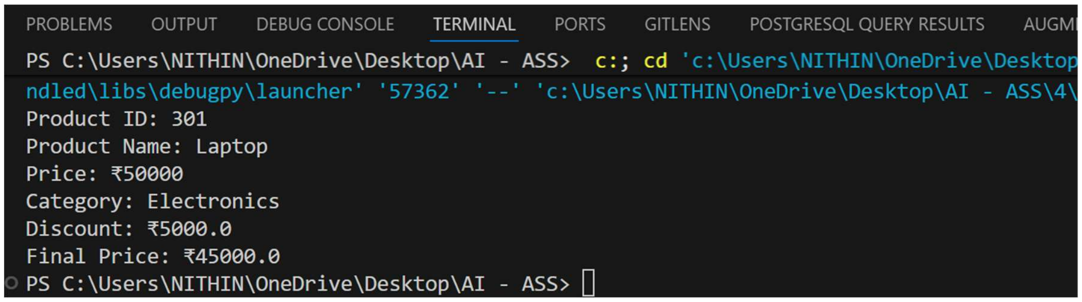
Create at least one product object, display details, and print the final price after discount.



4 > 02 > 2026 > 3.py > ...

```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 = 0.10 * self.price
        elif self.category == "Clothing":
            discount = 0.15 * self.price
        elif self.category == "Grocery":
            discount = 0.05 * self.price
        else:
            discount = 0
        final_price = self.price - discount
        print(f"Discount: ₹{discount}")
        print(f"Final Price: ₹{final_price}")
prod1 = Product(301, "Laptop", 50000, "Electronics")
prod1.display_details()
prod1.calculate_discount()

```

OUT PUT:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTGRESQL QUERY RESULTS AUGM

```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\Desktop
ndled\libs\debugpy\launcher' '57362' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI - ASS\4\
Product ID: 301
Product Name: Laptop
Price: ₹50000
Category: Electronics
Discount: ₹5000.0
Final Price: ₹45000.0
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```

Task 4:

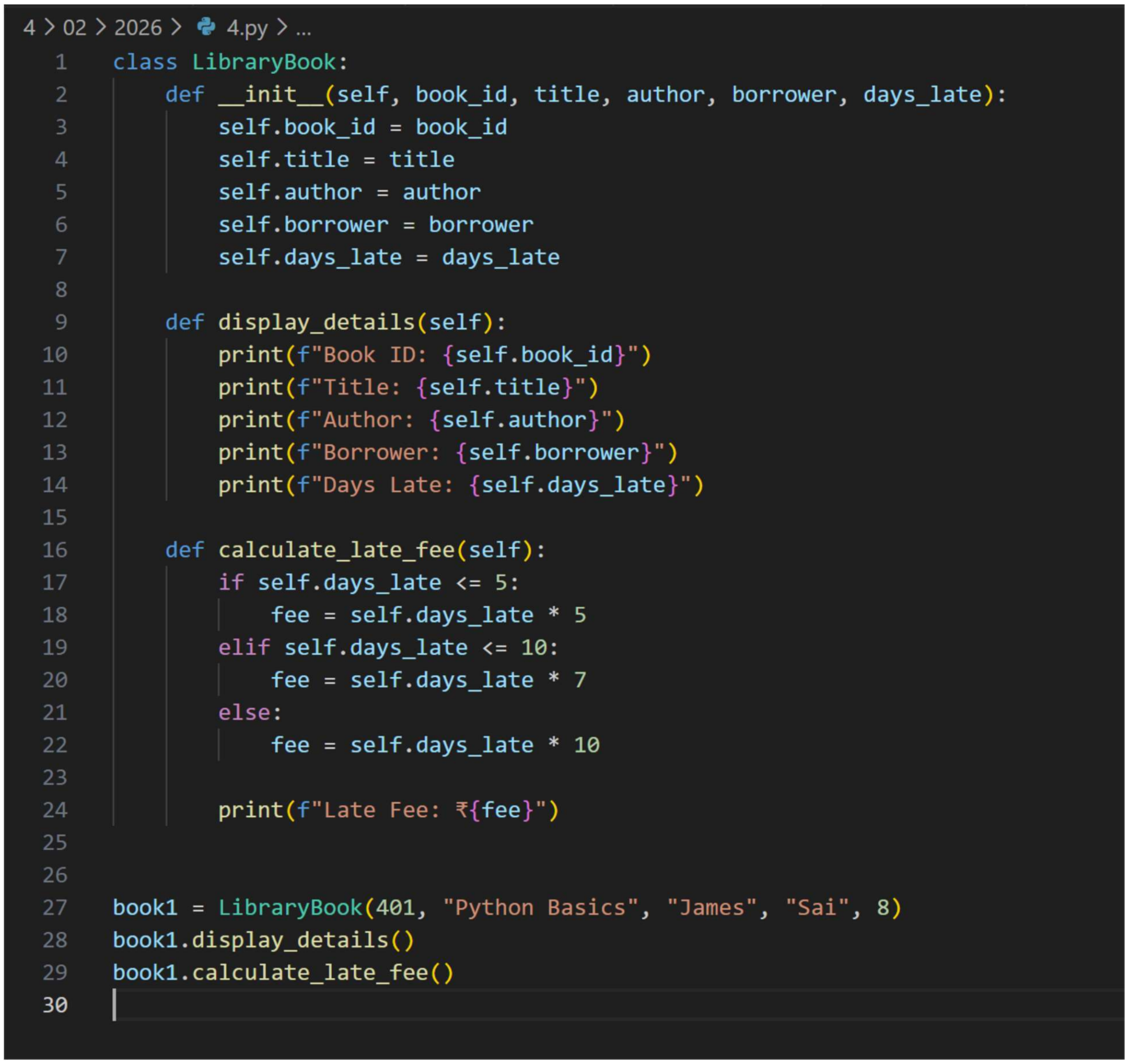
Book Late Fee Calculation- Create Python code that defines a class named `LibraryBook` with attributes: `book_id`, `title`, `author`, `borrower`, and `days_late`. Implement a method `display_details()` to print book details, and a method `calculate_late_fee()` where:

- Days late ≤ 5 → ₹5 per day

- 6 to 10 days late → ₹7 per day

- More than 10 days late → ₹10 per day

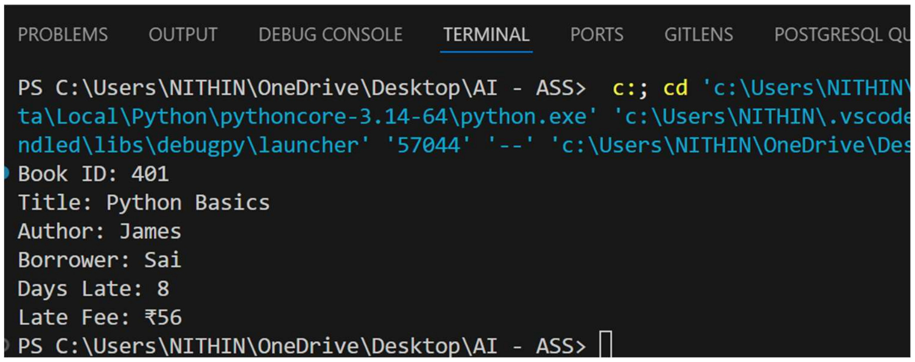
Create a book object, display details, and print the late fee.



4 > 02 > 2026 > 4.py > ...

```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_late_fee(self):
        if self.days_late <= 5:
            fee = self.days_late * 5
        elif self.days_late <= 10:
            fee = self.days_late * 7
        else:
            fee = self.days_late * 10

        print(f"Late Fee: ₹{fee}")


book1 = LibraryBook(401, "Python Basics", "James", "Sai", 8)
book1.display_details()
book1.calculate_late_fee()

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode
ndled\libs\debugpy\launcher' '57044' '--' 'c:\Users\NITHIN\OneDrive\Des
Book ID: 401
Title: Python Basics
Author: James
Borrower: Sai
Days Late: 8
Late Fee: ₹56
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```

Task 5:

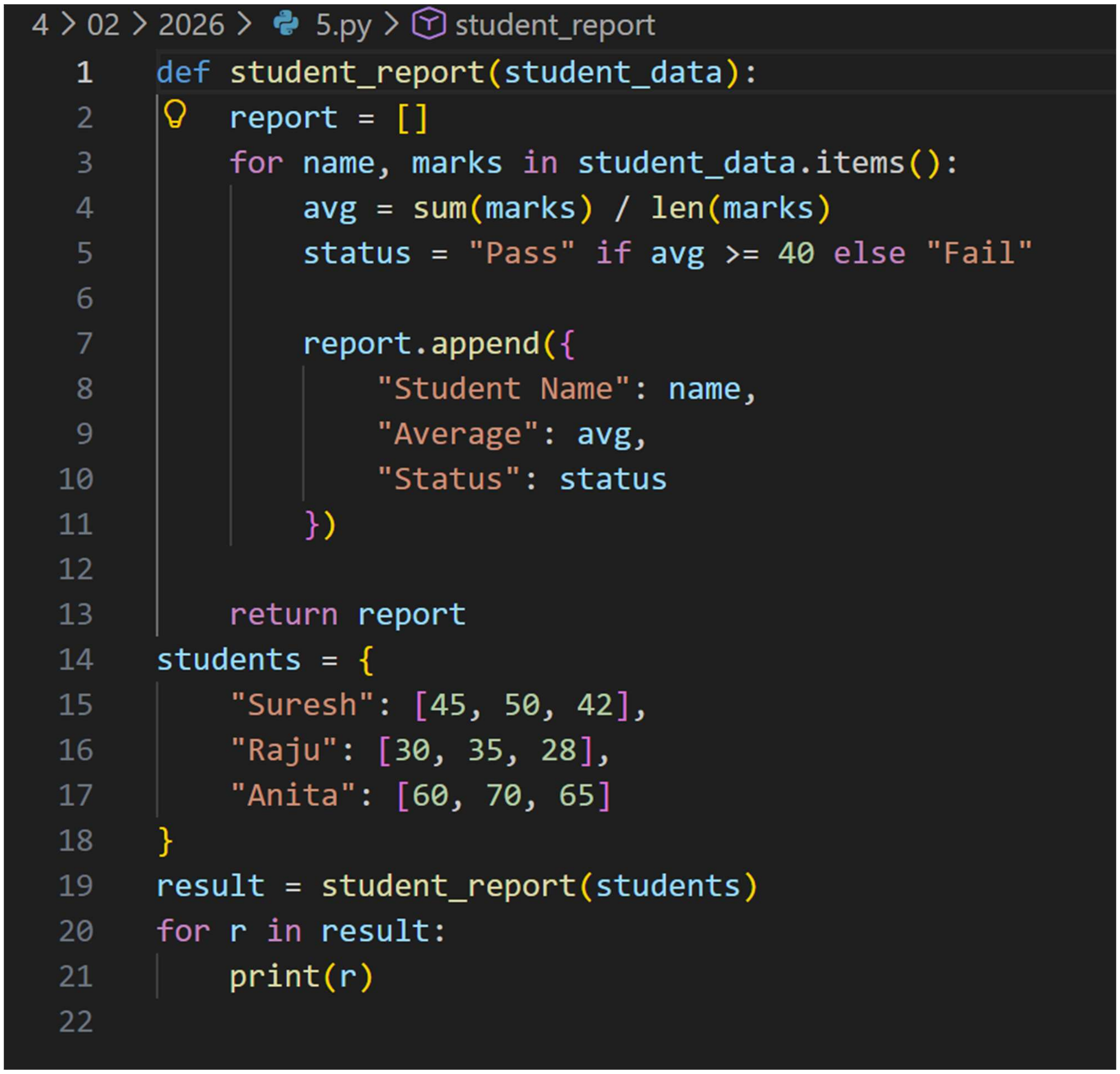
Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks. The function should:

- Calculate the average score for each student

- Determine pass/fail status (pass ≥ 40)

- Return a summary report as a list of dictionaries

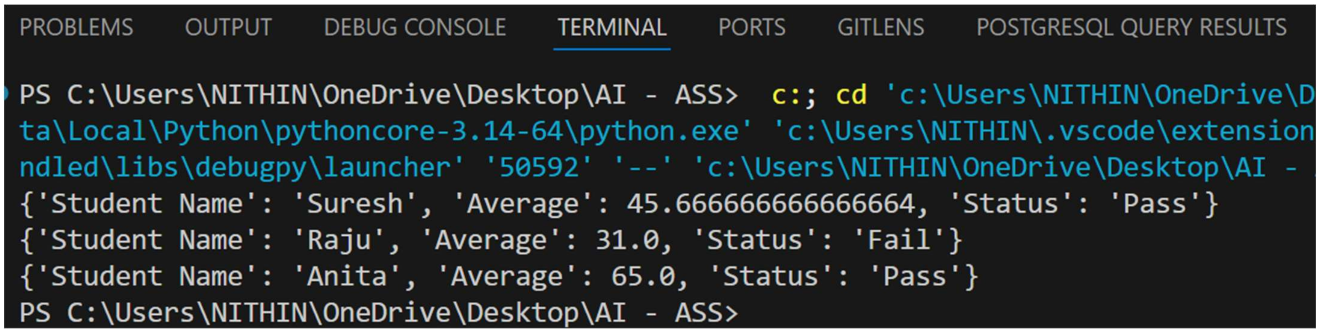
Use Copilot suggestions as you build the function and format the output.



```
4 > 02 > 2026 > 5.py > student_report
```

```python
def student_report(student_data):
    report = []
    for name, marks in student_data.items():
        avg = sum(marks) / len(marks)
        status = "Pass" if avg >= 40 else "Fail"

        report.append({
            "Student Name": name,
            "Average": avg,
            "Status": status
        })

    return report
students = {
    "Suresh": [45, 50, 42],
    "Raju": [30, 35, 28],
    "Anita": [60, 70, 65]
}
result = student_report(students)
for r in result:
    print(r)

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\D
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\extension
ndled\libs\debugpy\launcher' '50592' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI -
{'Student Name': 'Suresh', 'Average': 45.666666666666664, 'Status': 'Pass'}
{'Student Name': 'Raju', 'Average': 31.0, 'Status': 'Fail'}
{'Student Name': 'Anita', 'Average': 65.0, 'Status': 'Pass'}
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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:

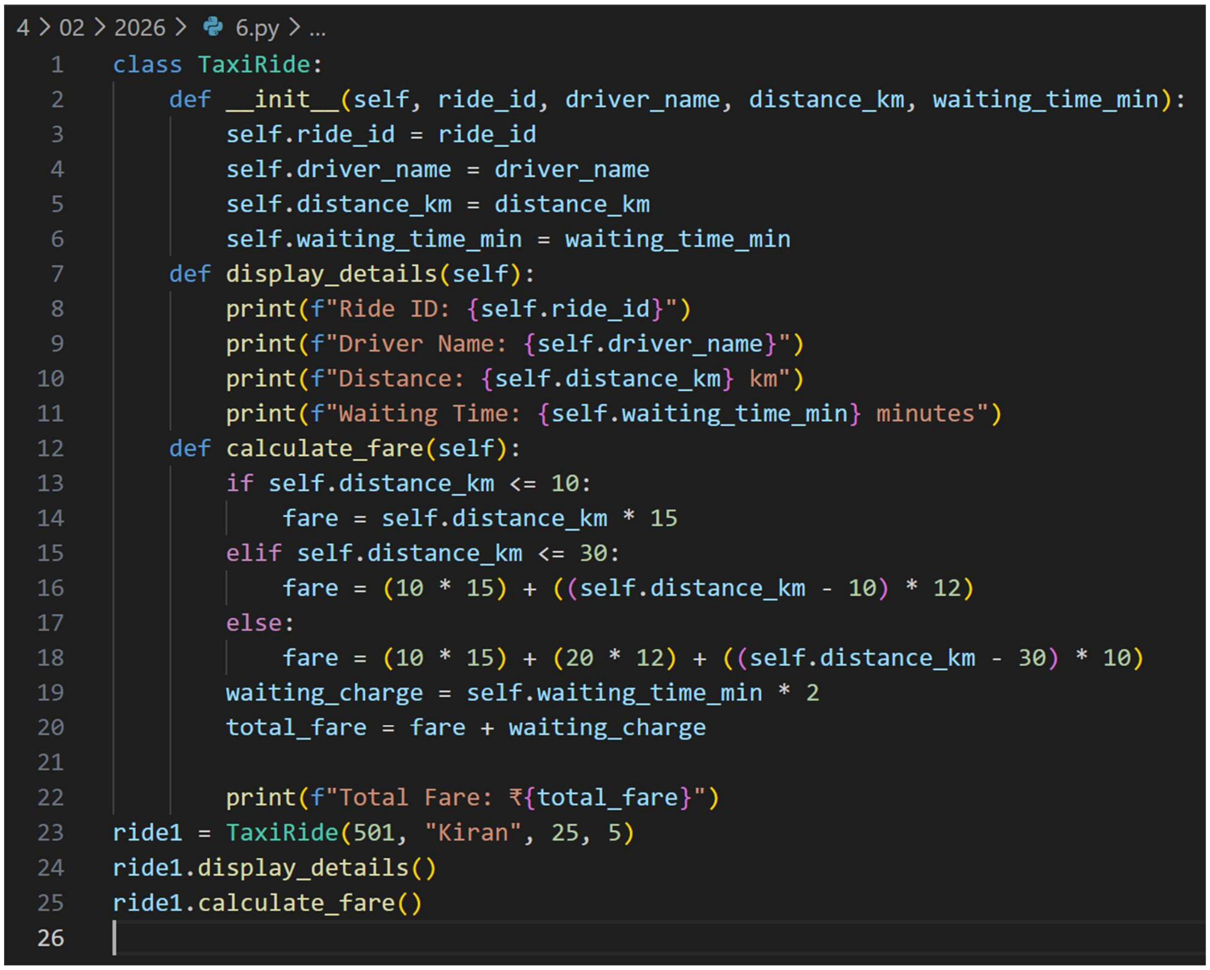
- ₹15 per km for the first 10 km

- ₹12 per km for the next 20 km

- ₹10 per km above 30 km

- Waiting charge: ₹2 per minute

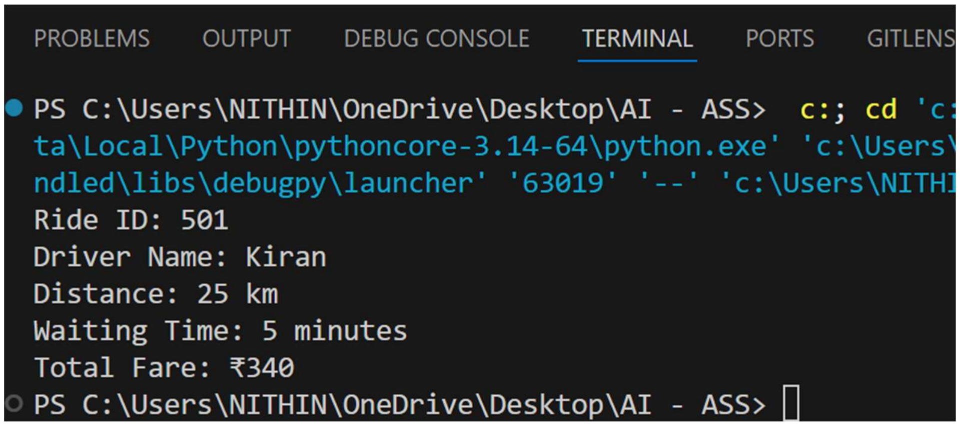
Create a ride object, display details, and print the total fare.



4 > 02 > 2026 > 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: {self.distance_km} km")
        print(f"Waiting Time: {self.waiting_time_min} minutes")
    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)
        waiting_charge = self.waiting_time_min * 2
        total_fare = fare + waiting_charge

        print(f"Total Fare: ₹{total_fare}")
ride1 = TaxiRide(501, "Kiran", 25, 5)
ride1.display_details()
ride1.calculate_fare()

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\
ndled\libs\debugpy\launcher' '63019' '--' 'c:\Users\NITHI
Ride ID: 501
Driver Name: Kiran
Distance: 25 km
Waiting Time: 5 minutes
Total Fare: ₹340
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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 returnthe following:

- Highest score in the class

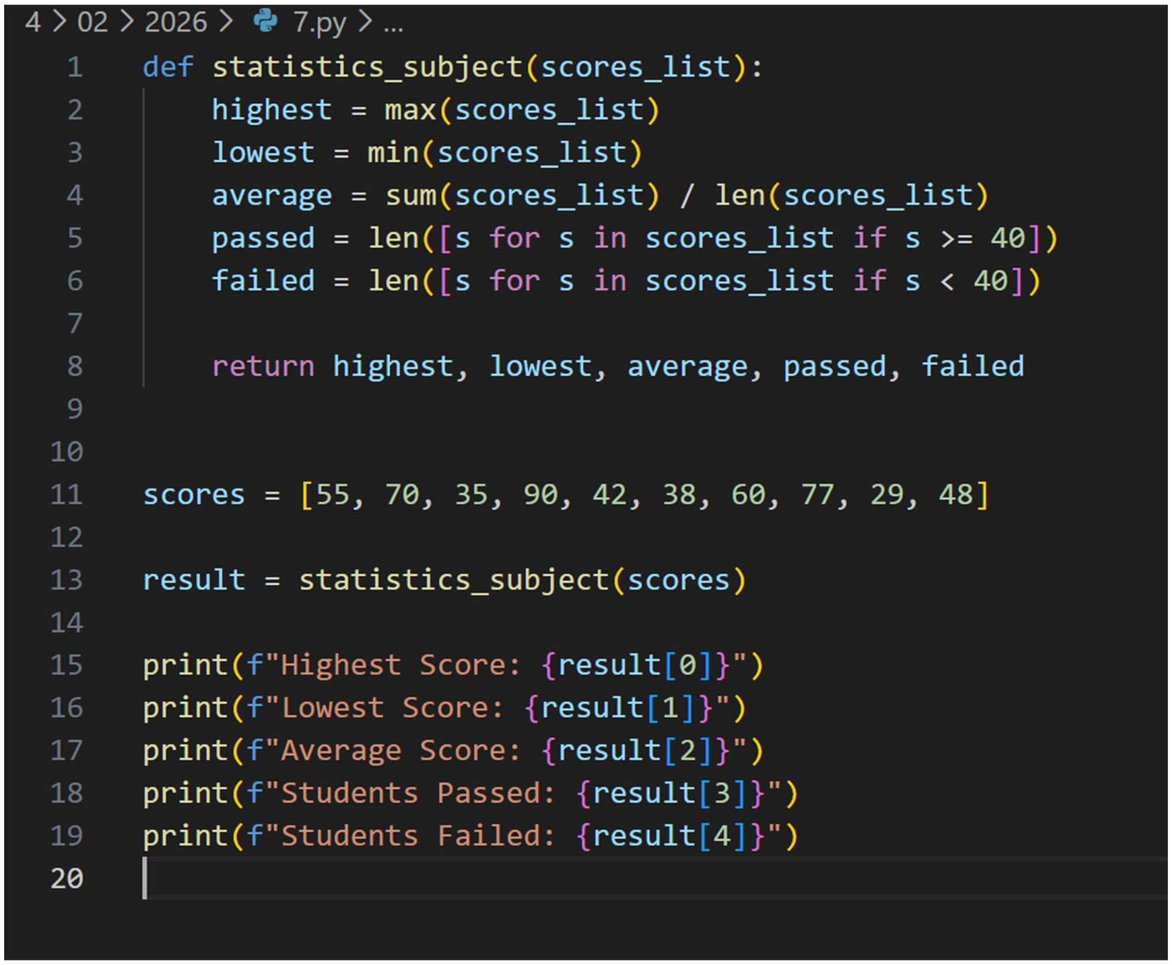
- Lowest score in the class

- Class average score

- Number of students passed (score ≥ 40)

- Number of students failed (score < 40)

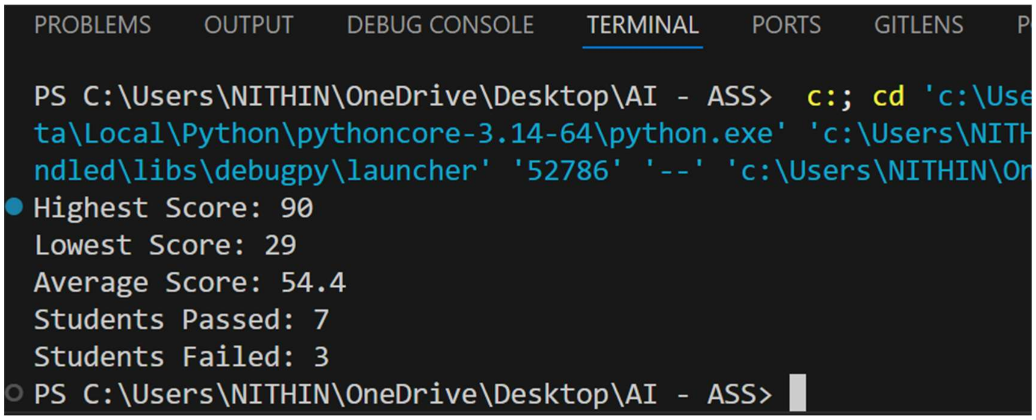
Allow Copilot to assist with aggregations and logic



4 > 02 > 2026 > 7.py > ...

```python
def statistics_subject(scores_list):
    highest = max(scores_list)
    lowest = min(scores_list)
    average = sum(scores_list) / len(scores_list)
    passed = len([s for s in scores_list if s >= 40])
    failed = len([s for s in scores_list if s < 40])

    return highest, lowest, average, passed, failed


scores = [55, 70, 35, 90, 42, 38, 60, 77, 29, 48]

result = statistics_subject(scores)

print(f"Highest Score: {result[0]}")
print(f"Lowest Score: {result[1]}")
print(f"Average Score: {result[2]}")
print(f"Students Passed: {result[3]}")
print(f"Students Failed: {result[4]}")

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Use
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITH
ndled\libs\debugpy\launcher' '52786' '--' 'c:\Users\NITHIN\Or
Highest Score: 90
Lowest Score: 29
Average Score: 54.4
Students Passed: 7
Students Failed: 3
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

Task Description #8 (Transparency in Algorithm Optimization)

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

• Naive approach(basic)

• Optimized approach

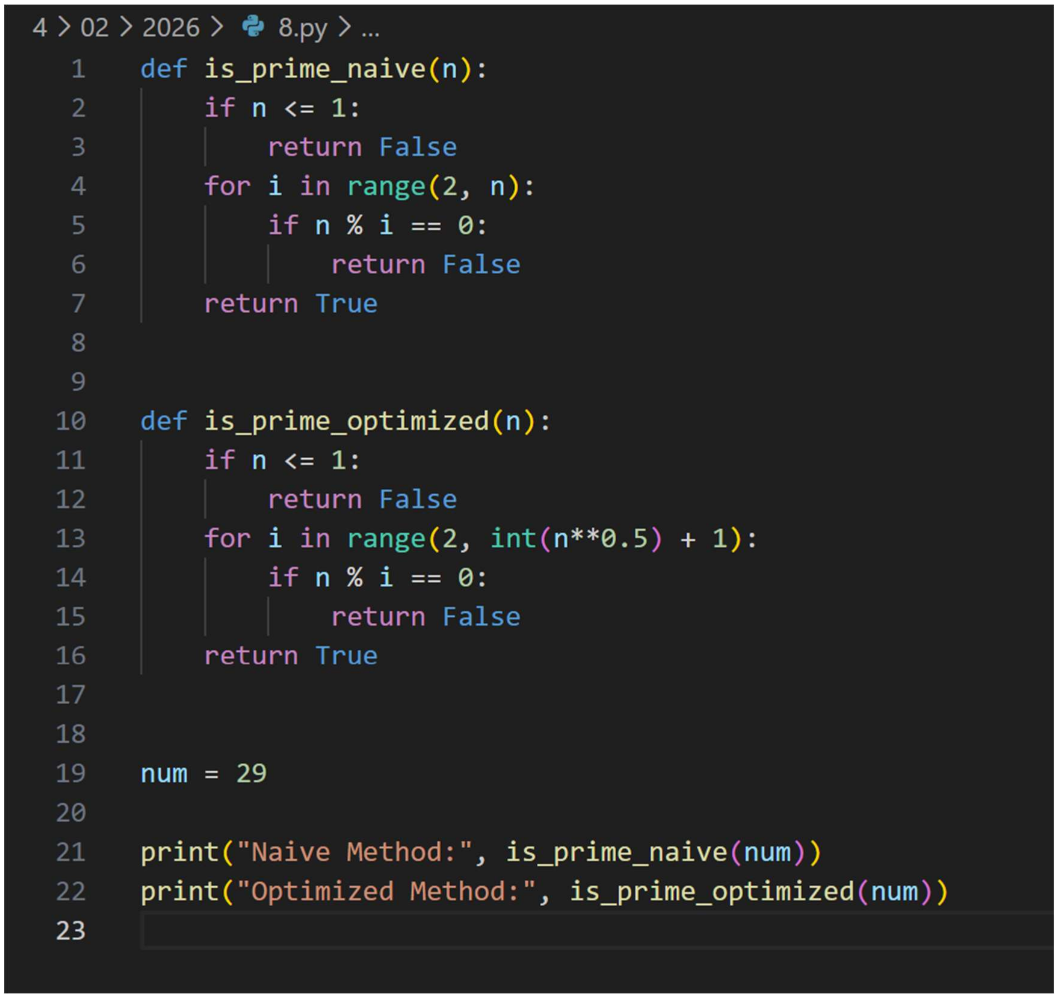
Prompt:

“Generate Python code for two prime-checking methods and explainhow the optimized version improves performance.” Expected Output:

• Code for both methods.

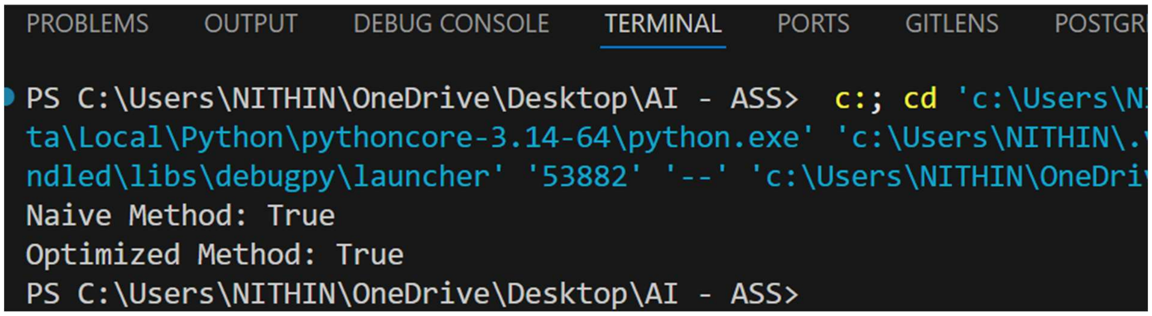
• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.



```python
def is_prime_naive(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
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True


num = 29

print("Naive Method:", is_prime_naive(num))
print("Optimized Method:", is_prime_optimized(num))

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\N
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.
ndled\libs\debugpy\launcher' '53882' '--' 'c:\Users\NITHIN\OneDri
Naive Method: True
Optimized Method: True
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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.

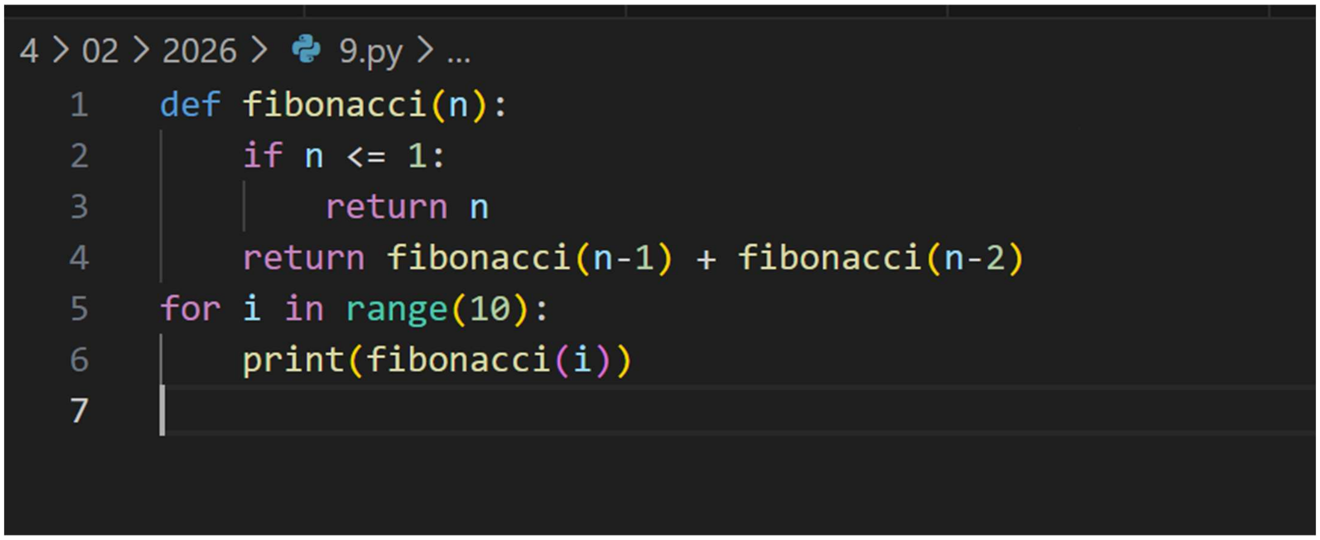
2. Ask AI to explain base cases and recursive calls.

Expected Output:

• Well-commented recursive code.

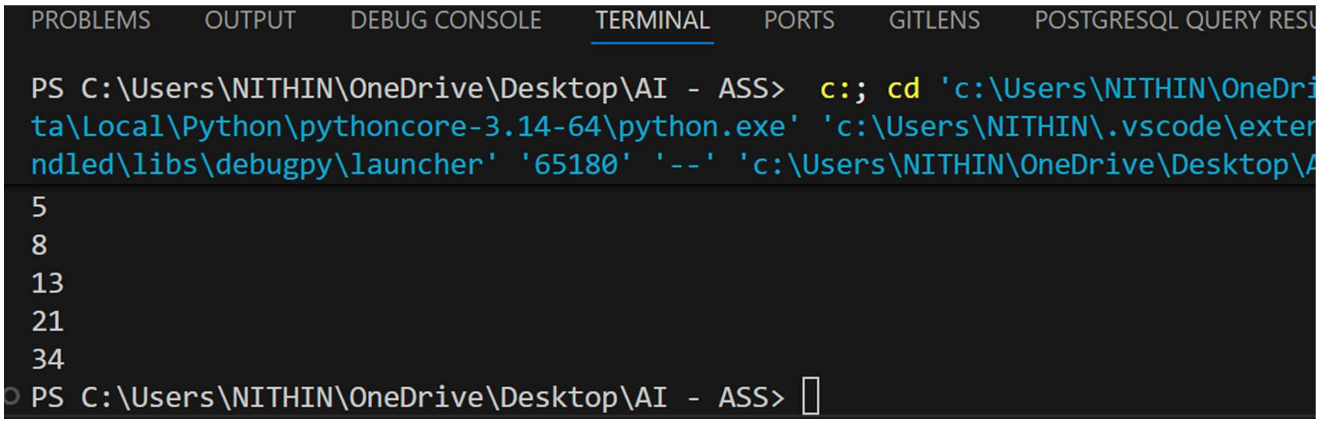
• Clear explanation of how recursion works.

• Verification that explanation matches actual execution.



4 > 02 > 2026 > 9.py > ...

```python
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
for i in range(10):
    print(fibonacci(i))

```



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDri
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\exter
ndled\libs\debugpy\launcher' '65180' '--' 'c:\Users\NITHIN\OneDrive\Desktop\A
5
8
13
21
34
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```

Task Description #10 (Transparency in Error Handling)

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

Prompt:

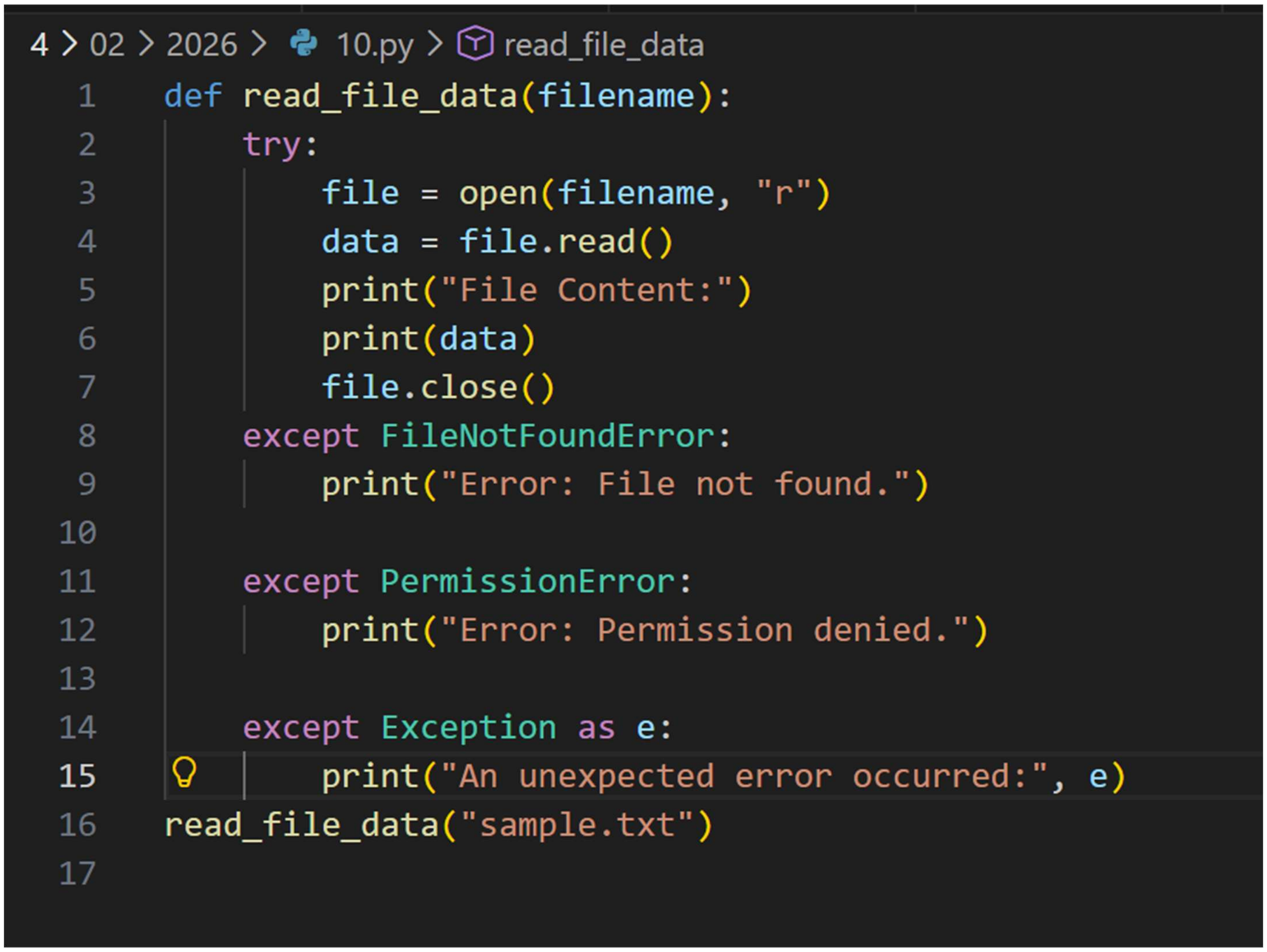
“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

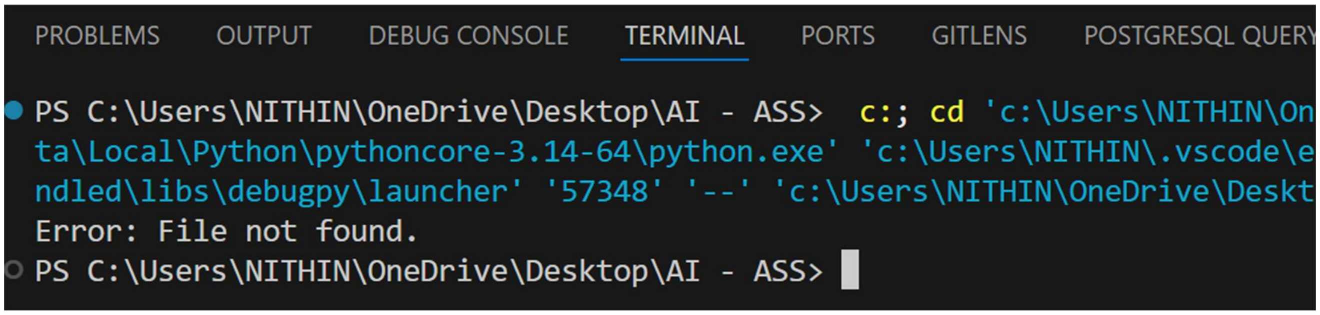
• Validation that explanations align with runtime behavior.



4 > 02 > 2026 > 10.py > read_file_data

```python
def read_file_data(filename):
    try:
        file = open(filename, "r")
        data = file.read()
        print("File Content:")
        print(data)
        file.close()
    except FileNotFoundError:
        print("Error: File not found.")

    except PermissionError:
        print("Error: Permission denied.")

    except Exception as e:
        print("An unexpected error occurred:", e)
read_file_data("sample.txt")
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\On
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\e
ndled\libs\debugpy\launcher' '57348' '--' 'c:\Users\NITHIN\OneDrive\Deskt
Error: File not found.
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```