AI Assistant Coding

Assignment-5.1&6.1

Name:K.Trinay Prasad

HT No:2303A52070

Batch:32

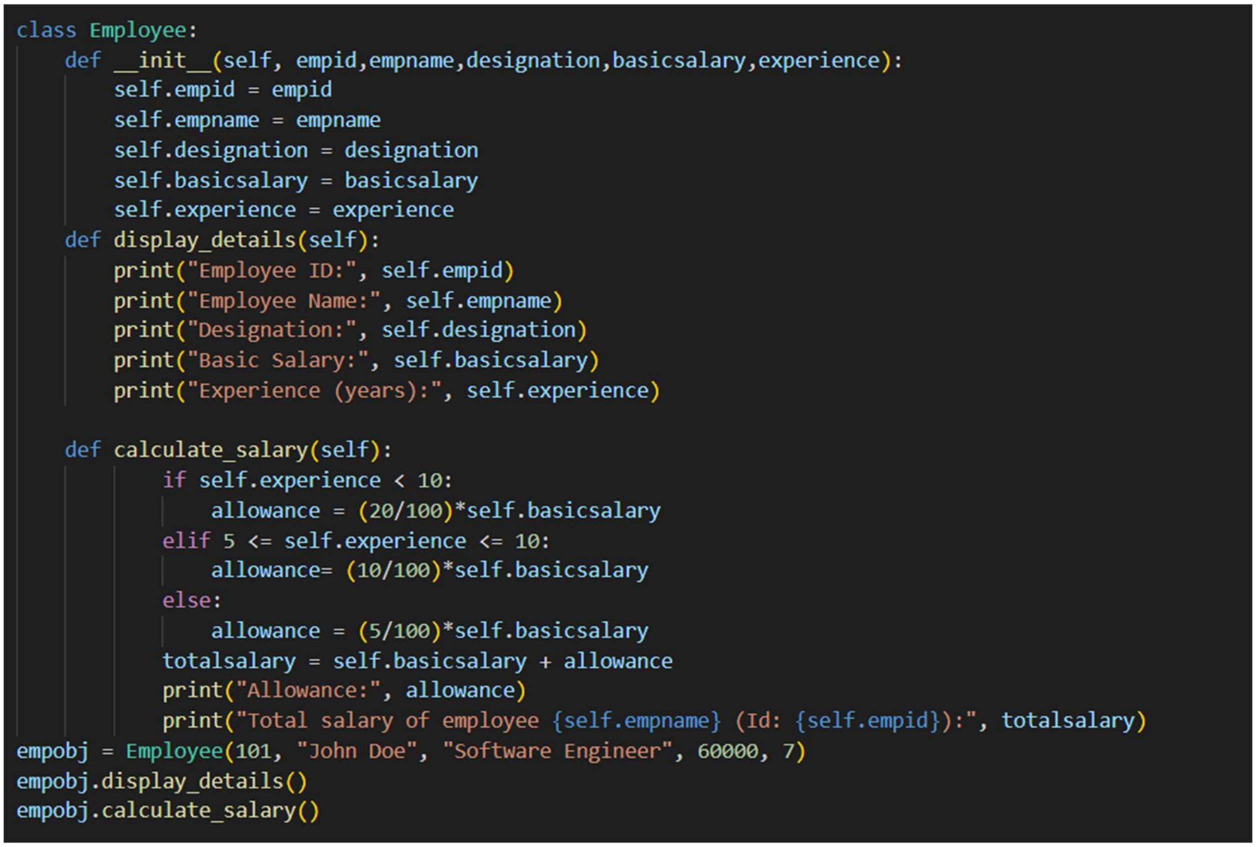
Task 1:

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\`

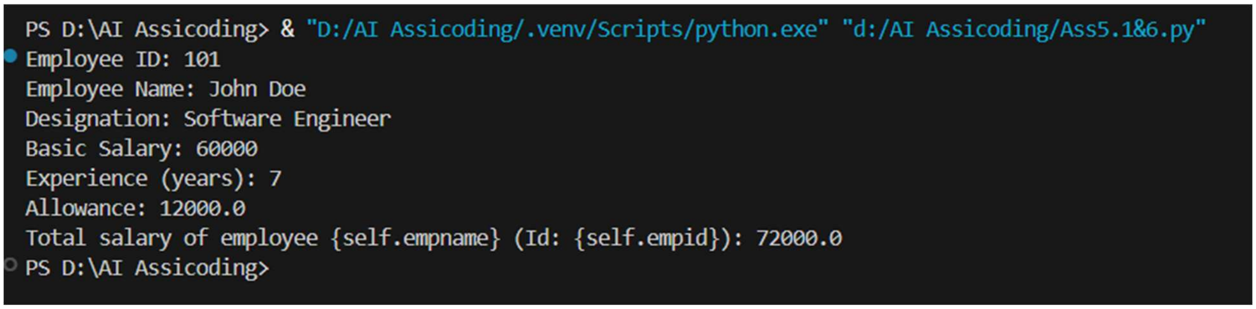
Finally, create at least one instance of the \`Employee\` class, call the \`display_details()\` method, and print the calculated allowance.

Code:



```
class Employee:
    def __init__(self, empid,empname,designation,basicsalary,experience):
        self.empid = empid
        self.empname = empname
        self.designation = designation
        self.basicsalary = basicsalary
        self.experience = experience
    def display_details(self):
        print("Employee ID:", self.empid)
        print("Employee Name:", self.empname)
        print("Designation:", self.designation)
        print("Basic Salary:", self.basicsalary)
        print("Experience (years):", self.experience)

    def calculate_salary(self):
            if self.experience < 10:
                allowance = (20/100)*self.basicsalary
            elif 5 <= self.experience <= 10:
                allowance= (10/100)*self.basicsalary
            else:
                allowance = (5/100)*self.basicsalary
            totalsalary = self.basicsalary + allowance
            print("Allowance:", allowance)
            print("Total salary of employee {self.empname} (Id: {self.empid}):", totalsalary)
empobj = Employee(101, "John Doe", "Software Engineer", 60000, 7)
empobj.display_details()
empobj.calculate_salary()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Employee ID: 101
Employee Name: John Doe
Designation: Software Engineer
Basic Salary: 60000
Experience (years): 7
Allowance: 12000.0
Total salary of employee {self.empname} (Id: {self.empid}): 72000.0
PS D:\AI Assicoding>
```

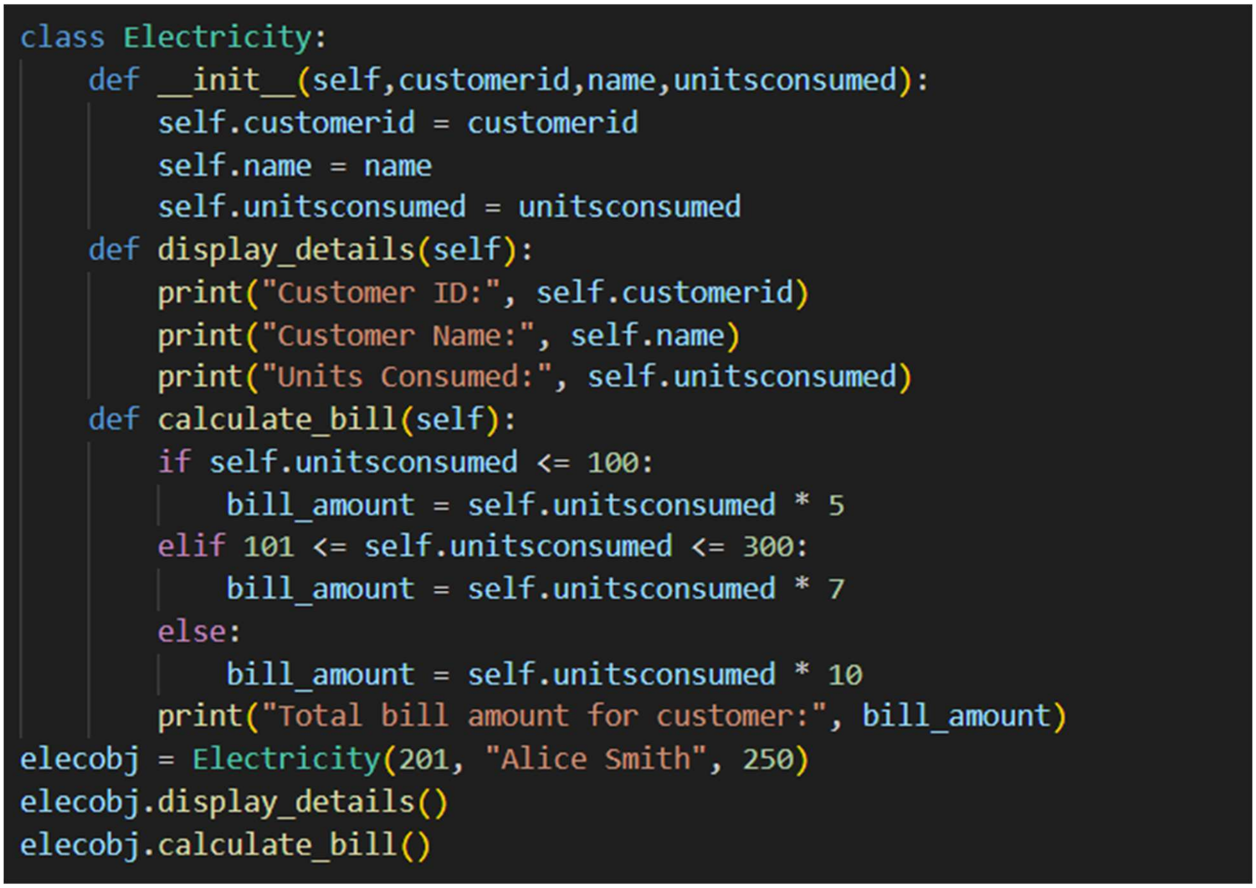
Task 2:

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

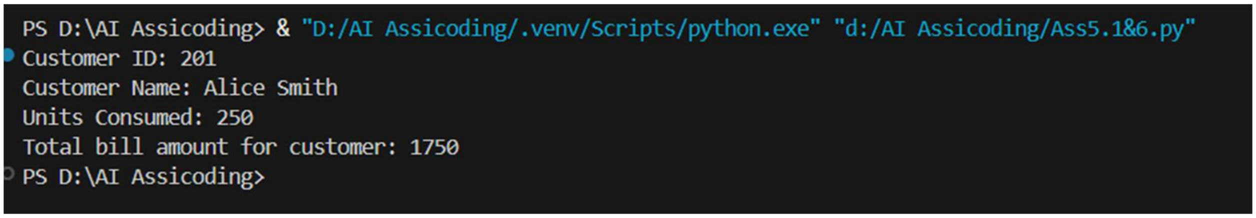
Create a bill object, display details, and print the total bill amount.

Code:



```python
class Electricity:
    def __init__(self,customerid,name,unitsconsumed):
        self.customerid = customerid
        self.name = name
        self.unitsconsumed = unitsconsumed
    def display_details(self):
        print("Customer ID:", self.customerid)
        print("Customer Name:", self.name)
        print("Units Consumed:", self.unitsconsumed)
    def calculate_bill(self):
        if self.unitsconsumed <= 100:
            bill_amount = self.unitsconsumed * 5
        elif 101 <= self.unitsconsumed <= 300:
            bill_amount = self.unitsconsumed * 7
        else:
            bill_amount = self.unitsconsumed * 10
        print("Total bill amount for customer:", bill_amount)
elecobj = Electricity(201, "Alice Smith", 250)
elecobj.display_details()
elecobj.calculate_bill()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Customer ID: 201
Customer Name: Alice Smith
Units Consumed: 250
Total bill amount for customer: 1750
PS D:\AI Assicoding>
```

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:

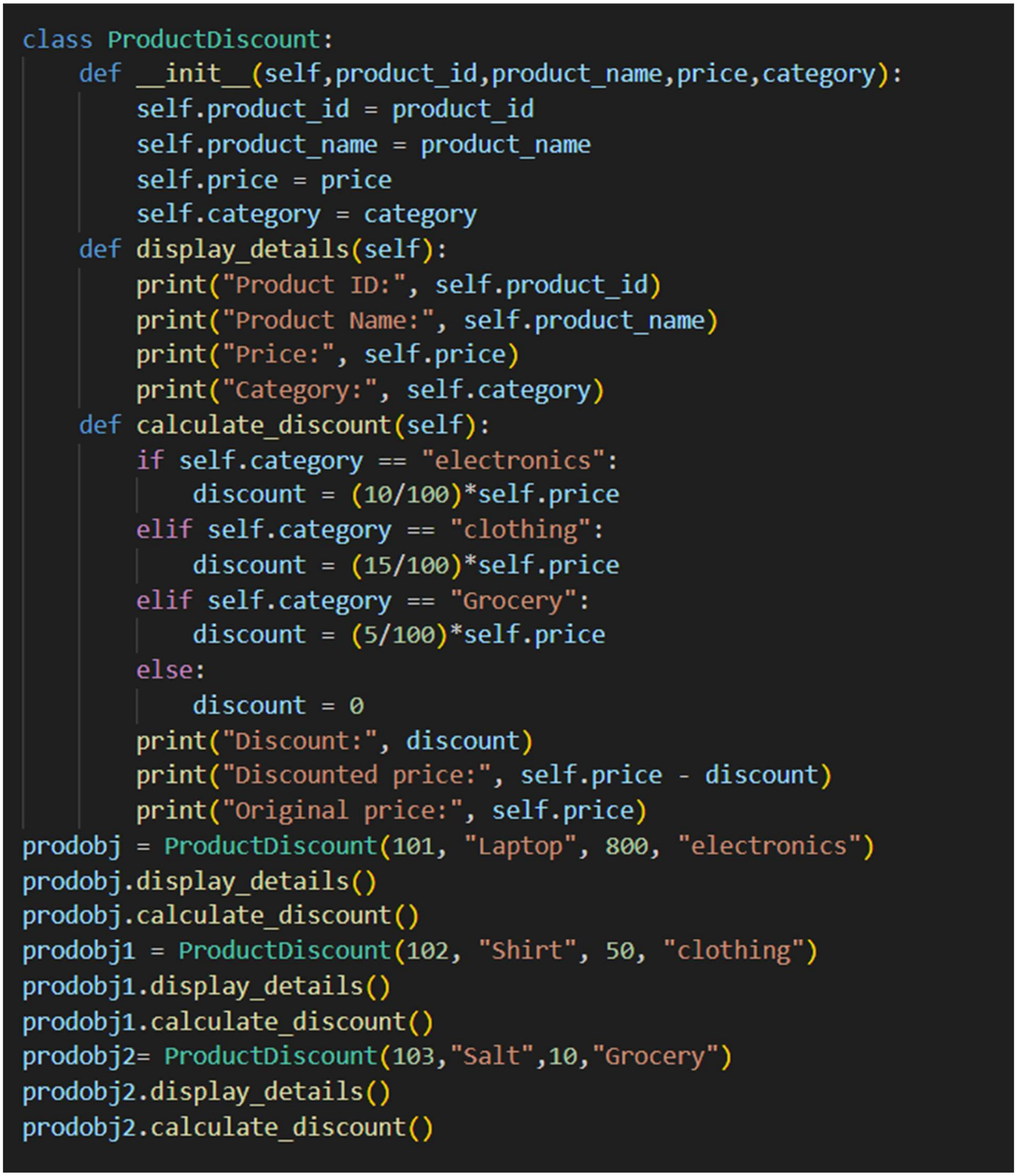
- Electronics → 10% discount

- Clothing → 15% discount

- Grocery → 5% discount

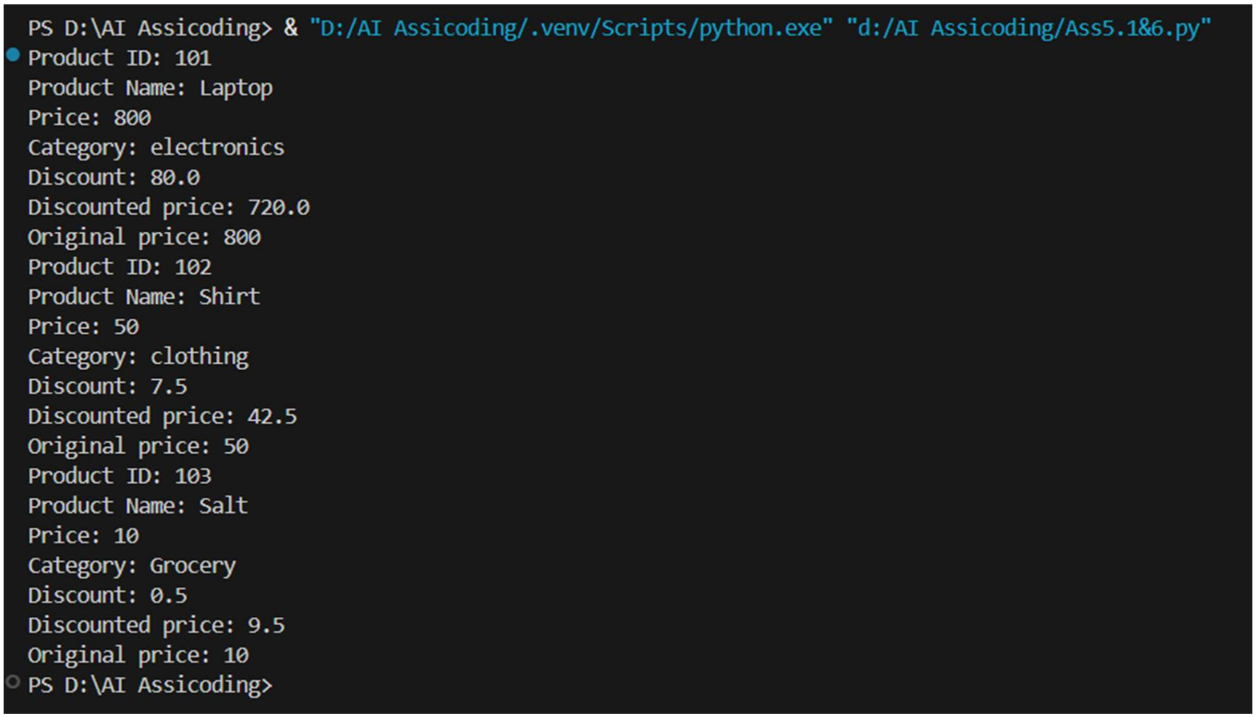
Create at least one product object, display details, and print the final price after discount.

Code:



```
class ProductDiscount:
    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("Product ID:", self.product_id)
        print("Product Name:", self.product_name)
        print("Price:", self.price)
        print("Category:", self.category)
    def calculate_discount(self):
        if self.category == "electronics":
            discount = (10/100)*self.price
        elif self.category == "clothing":
            discount = (15/100)*self.price
        elif self.category == "Grocery":
            discount = (5/100)*self.price
        else:
            discount = 0
        print("Discount:", discount)
        print("Discounted price:", self.price - discount)
        print("Original price:", self.price)
prodobj = ProductDiscount(101, "Laptop", 800, "electronics")
prodobj.display_details()
prodobj.calculate_discount()
prodobj1 = ProductDiscount(102, "Shirt", 50, "clothing")
prodobj1.display_details()
prodobj1.calculate_discount()
prodobj2= ProductDiscount(103,"Salt",10,"Grocery")
prodobj2.display_details()
prodobj2.calculate_discount()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Product ID: 101
Product Name: Laptop
Price: 800
Category: electronics
Discount: 80.0
Discounted price: 720.0
Original price: 800
Product ID: 102
Product Name: Shirt
Price: 50
Category: clothing
Discount: 7.5
Discounted price: 42.5
Original price: 50
Product ID: 103
Product Name: Salt
Price: 10
Category: Grocery
Discount: 0.5
Discounted price: 9.5
Original price: 10
PS D:\AI Assicoding>
```

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:

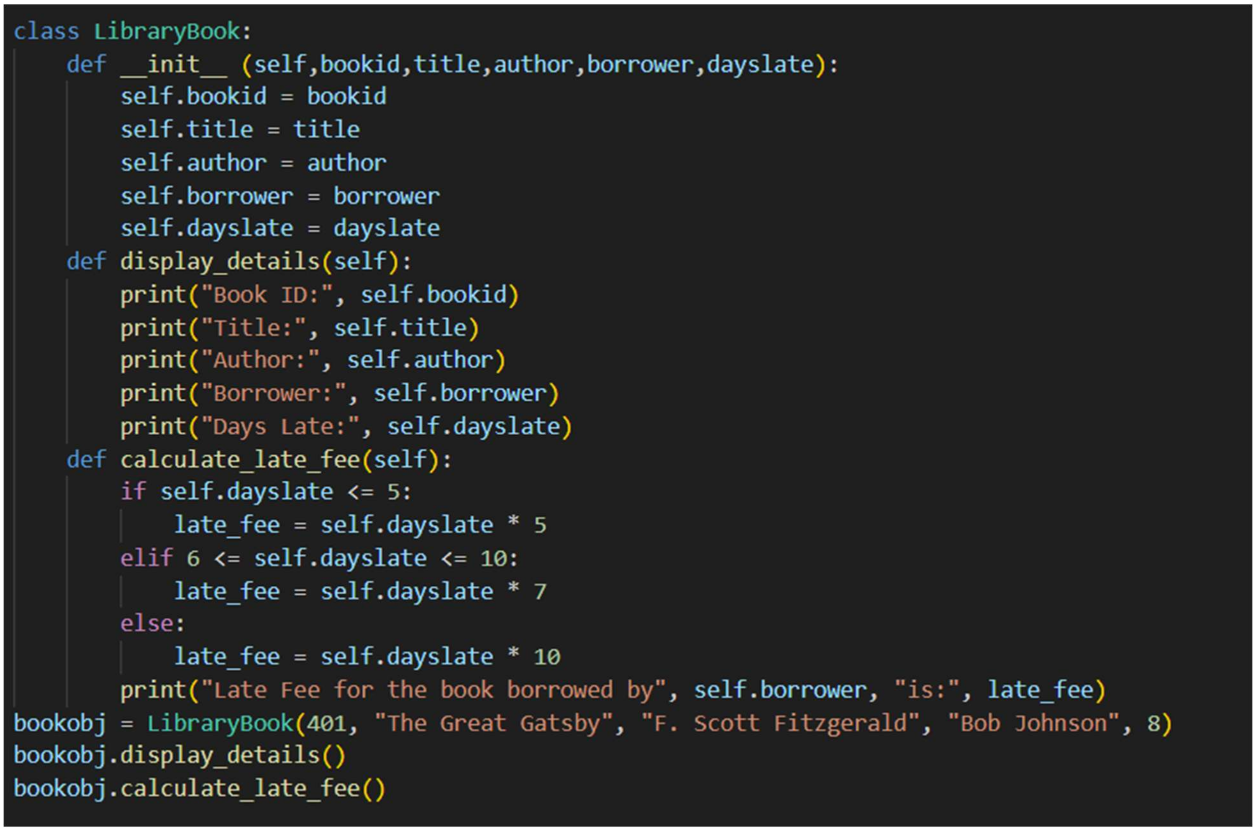
- Days late ≤ 5 → ₹5 per day

- 6 to 10 days late → ₹7 per day

- More than 10 days late → ₹10 per day

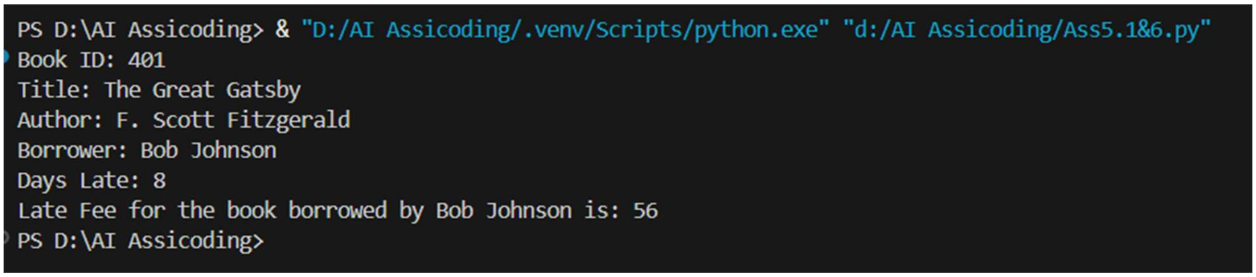
Create a book object, display details, and print the late fee.

Code:



```python
class LibraryBook:
    def __init__ (self,bookid,title,author,borrower,dayslate):
        self.bookid = bookid
        self.title = title
        self.author = author
        self.borrower = borrower
        self.dayslate = dayslate
    def display_details(self):
        print("Book ID:", self.bookid)
        print("Title:", self.title)
        print("Author:", self.author)
        print("Borrower:", self.borrower)
        print("Days Late:", self.dayslate)
    def calculate_late_fee(self):
        if self.dayslate <= 5:
            late_fee = self.dayslate * 5
        elif 6 <= self.dayslate <= 10:
            late_fee = self.dayslate * 7
        else:
            late_fee = self.dayslate * 10
        print("Late Fee for the book borrowed by", self.borrower, "is:", late_fee)
bookobj = LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald", "Bob Johnson", 8)
bookobj.display_details()
bookobj.calculate_late_fee()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Book ID: 401
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Bob Johnson
Days Late: 8
Late Fee for the book borrowed by Bob Johnson is: 56
PS D:\AI Assicoding>
```

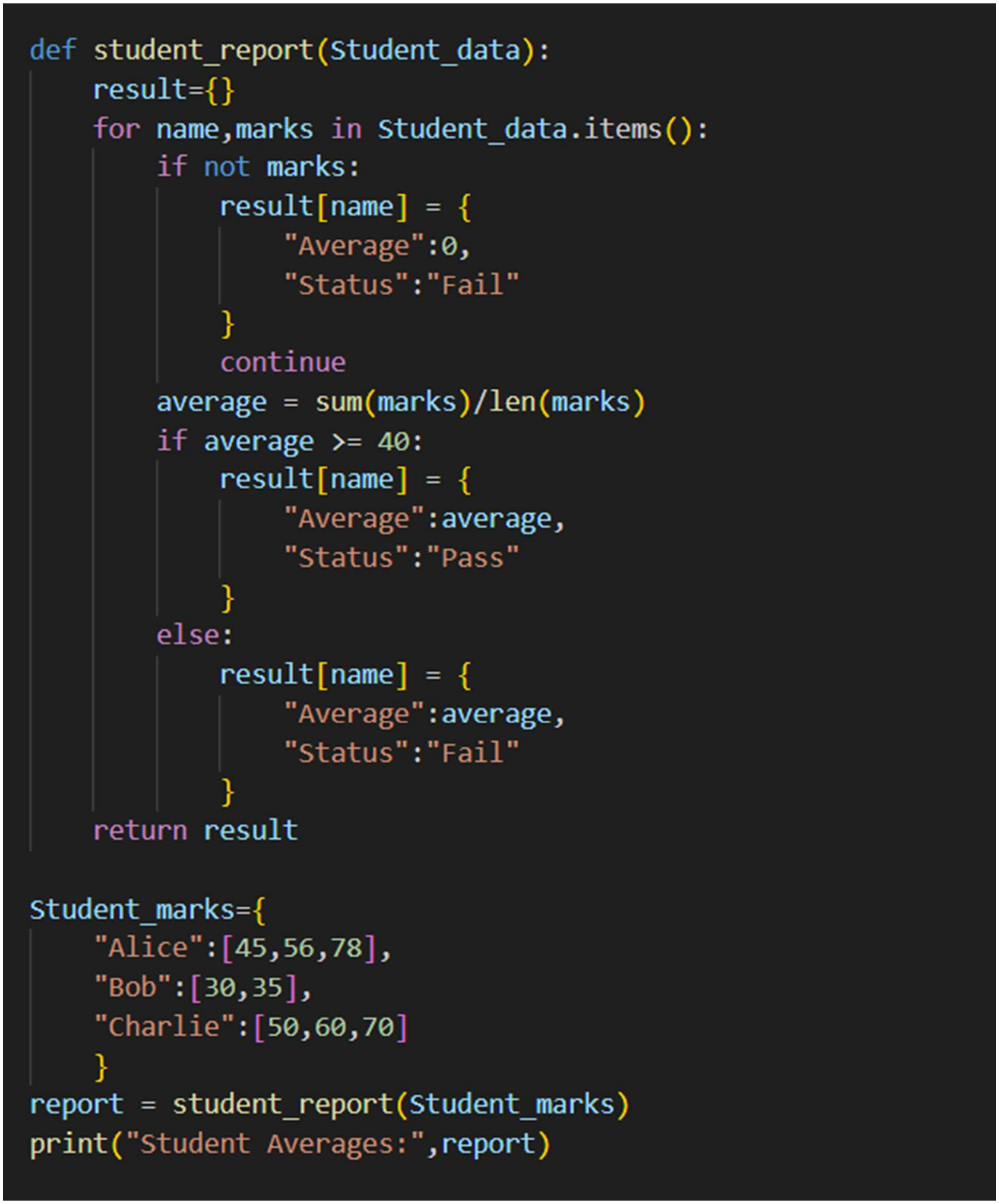
Task 5:

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

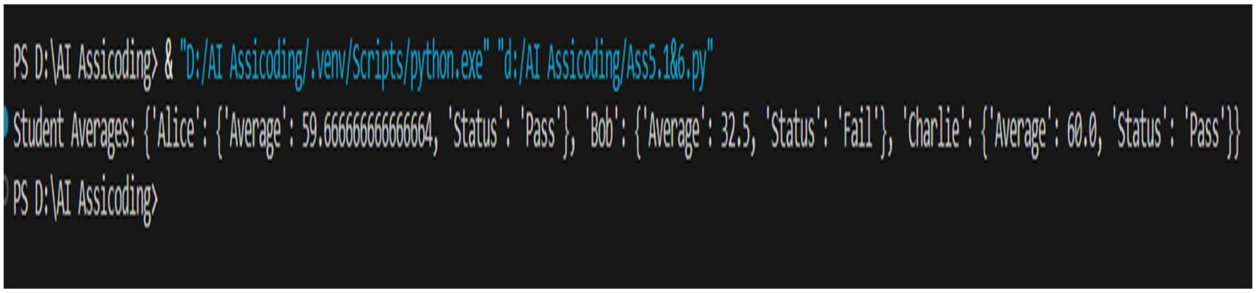
Use Copilot suggestions as you build the function and format the output.

Code:



```python
def student_report(Student_data):
    result={}
    for name,marks in Student_data.items():
        if not marks:
            result[name] = {
                "Average":0,
                "Status":"Fail"
            }
            continue
        average = sum(marks)/len(marks)
        if average >= 40:
            result[name] = {
                "Average":average,
                "Status":"Pass"
            }
        else:
            result[name] = {
                "Average":average,
                "Status":"Fail"
            }
    return result

Student_marks={
    "Alice":[45,56,78],
    "Bob":[30,35],
    "Charlie":[50,60,70]
    }
report = student_report(Student_marks)
print("Student Averages:",report)
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Student Averages: {'Alice': {'Average': 59.666666666666664, 'Status': 'Pass'}, 'Bob': {'Average': 32.5, 'Status': 'Fail'}, 'Charlie': {'Average': 60.0, 'Status': 'Pass'}}
PS D:\AI Assicoding>
```

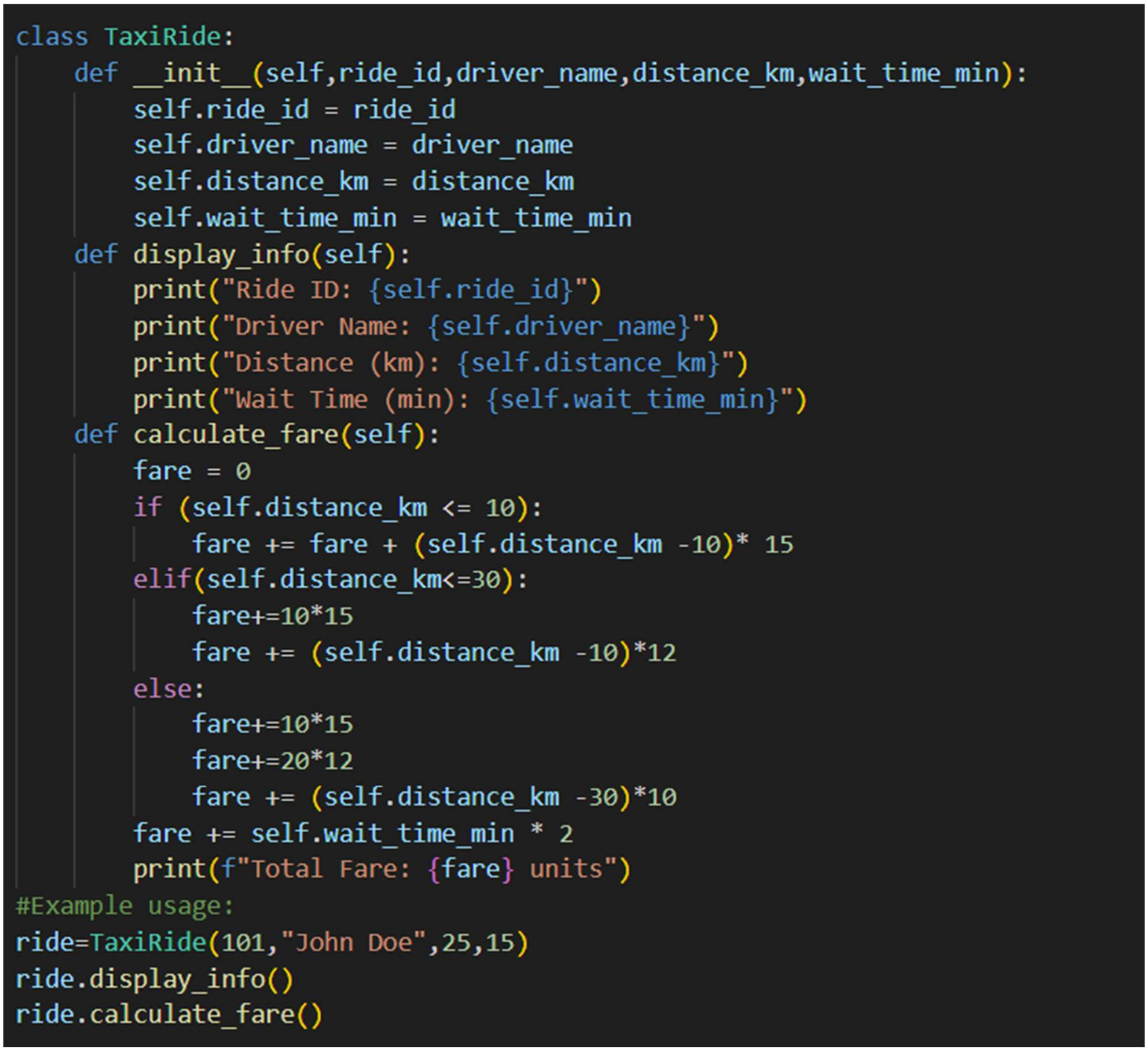
Task 6:

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

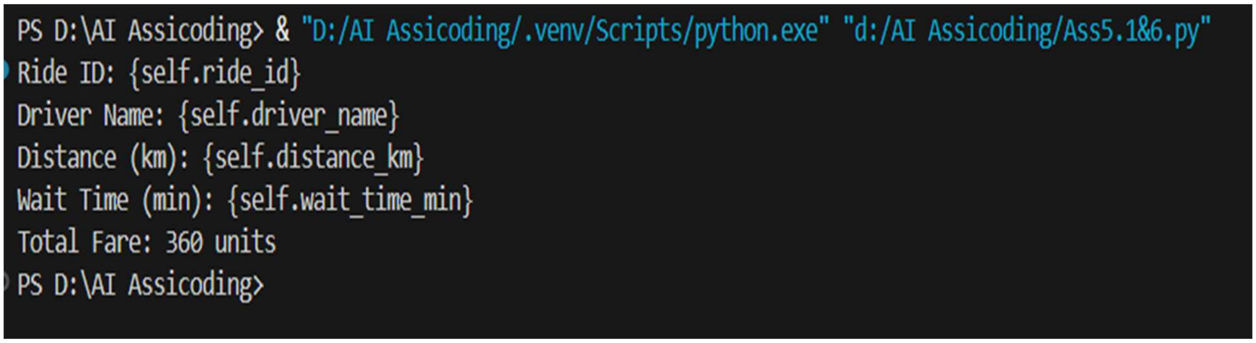
Create a ride object, display details, and print the total fare.

Code:



```python
class TaxiRide:
    def __init__(self,ride_id,driver_name,distance_km,wait_time_min):
        self.ride_id = ride_id
        self.driver_name = driver_name
        self.distance_km = distance_km
        self.wait_time_min = wait_time_min
    def display_info(self):
        print("Ride ID: {self.ride_id}")
        print("Driver Name: {self.driver_name}")
        print("Distance (km): {self.distance_km}")
        print("Wait Time (min): {self.wait_time_min}")
    def calculate_fare(self):
        fare = 0
        if (self.distance_km <= 10):
            fare += fare + (self.distance_km -10)* 15
        elif(self.distance_km<=30):
            fare+=10*15
            fare += (self.distance_km -10)*12
        else:
            fare+=10*15
            fare+=20*12
            fare += (self.distance_km -30)*10
        fare += self.wait_time_min * 2
        print(f"Total Fare: {fare} units")
#Example usage:
ride=TaxiRide(101,"John Doe",25,15)
ride.display_info()
ride.calculate_fare()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Ride ID: {self.ride_id}
Driver Name: {self.driver_name}
Distance (km): {self.distance_km}
Wait Time (min): {self.wait_time_min}
Total Fare: 360 units
PS D:\AI Assicoding>
```

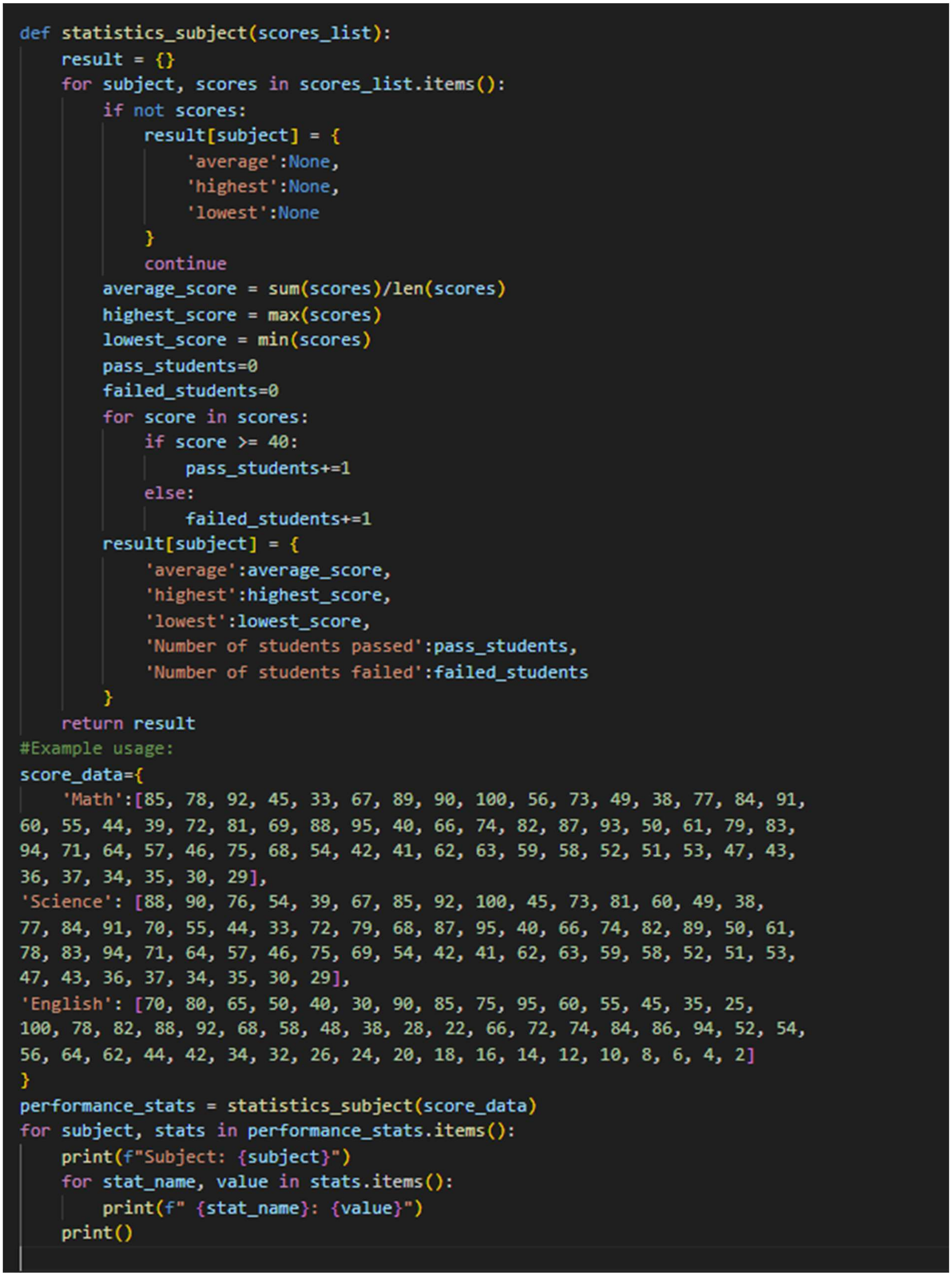
Task 7:

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)

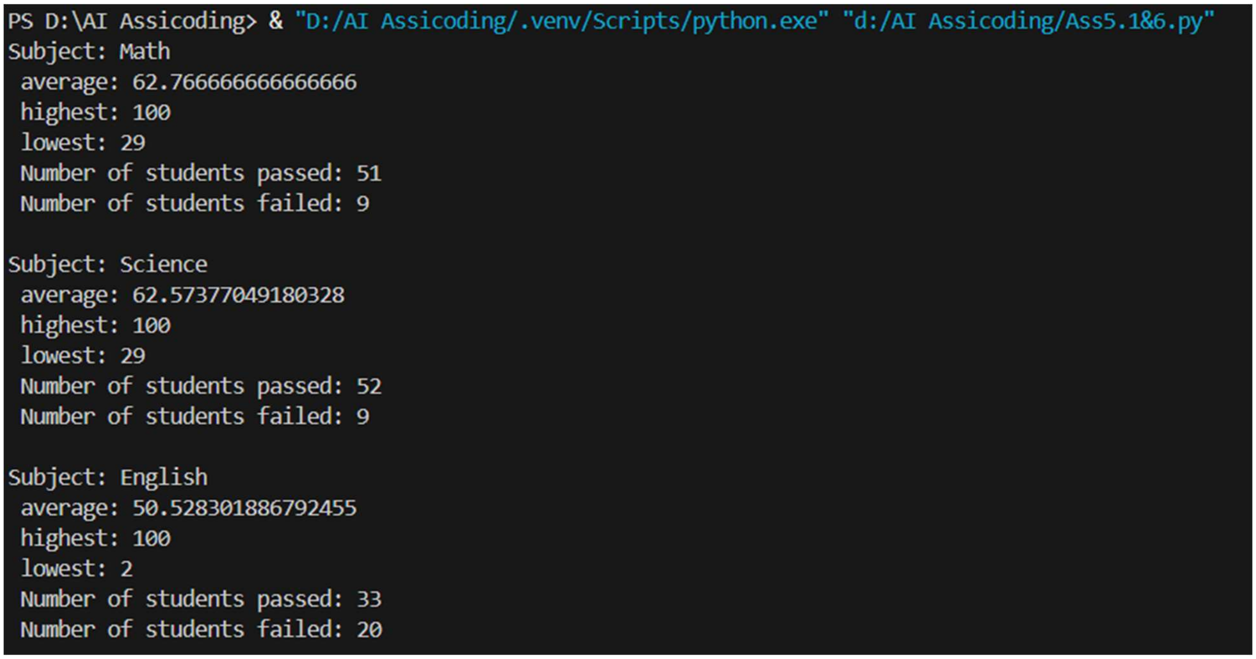
Allow Copilot to assist with aggregations and logic

Code:



```python
def statistics_subject(scores_list):
    result = {}
    for subject, scores in scores_list.items():
        if not scores:
            result[subject] = {
                'average':None,
                'highest':None,
                'lowest':None
            }
            continue
        average_score = sum(scores)/len(scores)
        highest_score = max(scores)
        lowest_score = min(scores)
        pass_students=0
        failed_students=0
        for score in scores:
            if score >= 40:
                pass_students+=1
            else:
                failed_students+=1
        result[subject] = {
            'average':average_score,
            'highest':highest_score,
            'lowest':lowest_score,
            'Number of students passed':pass_students,
            'Number of students failed':failed_students
        }
    return result
#Example usage:
score_data={
    'Math':[85, 78, 92, 45, 33, 67, 89, 90, 100, 56, 73, 49, 38, 77, 84, 91,
60, 55, 44, 39, 72, 81, 69, 88, 95, 40, 66, 74, 82, 87, 93, 50, 61, 79, 83,
94, 71, 64, 57, 46, 75, 68, 54, 42, 41, 62, 63, 59, 58, 52, 51, 53, 47, 43,
36, 37, 34, 35, 30, 29],
'Science': [88, 90, 76, 54, 39, 67, 85, 92, 100, 45, 73, 81, 60, 49, 38,
77, 84, 91, 70, 55, 44, 33, 72, 79, 68, 87, 95, 40, 66, 74, 82, 89, 50, 61,
78, 83, 94, 71, 64, 57, 46, 75, 69, 54, 42, 41, 62, 63, 59, 58, 52, 51, 53,
47, 43, 36, 37, 34, 35, 30, 29],
'English': [70, 80, 65, 50, 40, 30, 90, 85, 75, 95, 60, 55, 45, 35, 25,
100, 78, 82, 88, 92, 68, 58, 48, 38, 28, 22, 66, 72, 74, 84, 86, 94, 52, 54,
56, 64, 62, 44, 42, 34, 32, 26, 24, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2]
}
performance_stats = statistics_subject(score_data)
for subject, stats in performance_stats.items():
    print(f"Subject: {subject}")
    for stat_name, value in stats.items():
        print(f" {stat_name}: {value}")
    print()
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Subject: Math
 average: 62.766666666666666
 highest: 100
 lowest: 29
 Number of students passed: 51
 Number of students failed: 9

Subject: Science
 average: 62.57377049180328
 highest: 100
 lowest: 29
 Number of students passed: 52
 Number of students failed: 9

Subject: English
 average: 50.528301886792455
 highest: 100
 lowest: 2
 Number of students passed: 33
 Number of students failed: 20
```

Task 8: (Transparency in Algorithm Optimization)

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

- Naive approach(basic)
- Optimized approach

Prompt:

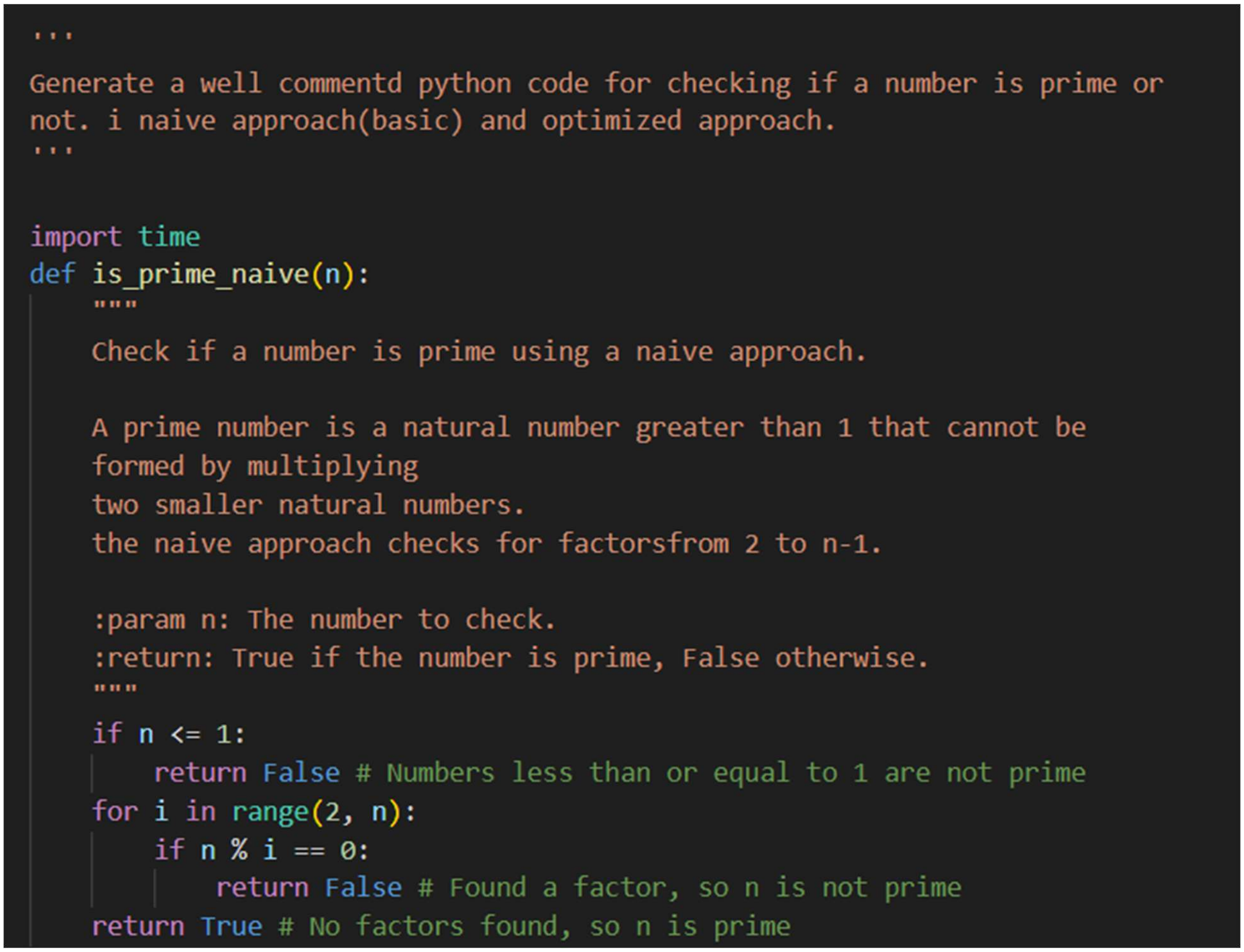
"Generate Python code for two prime-checking methods and explain how the optimized version improves performance."

Expected Output:

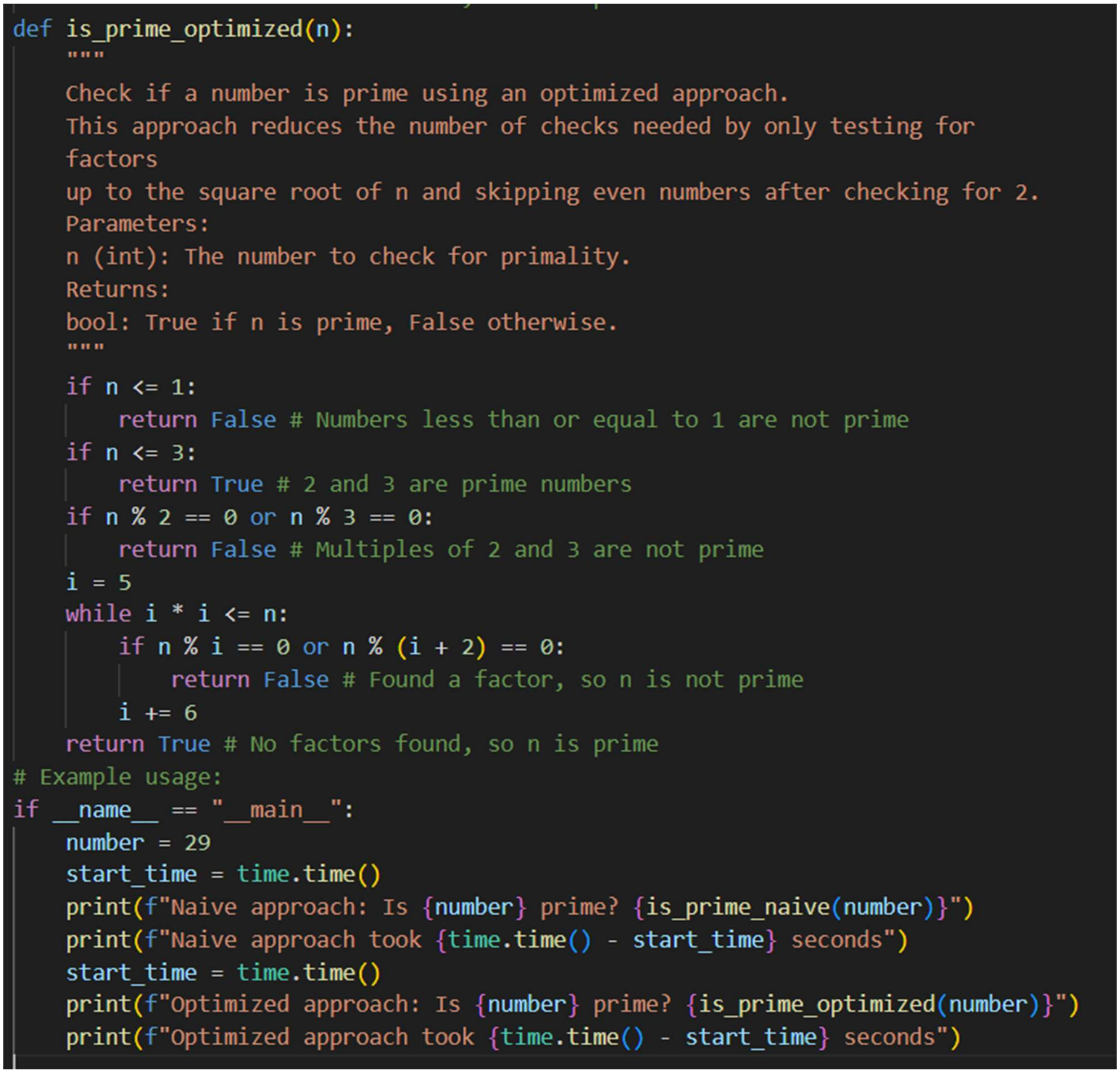
- Code for both methods.
- Transparent explanation of time complexity.

- Comparison highlighting efficiency improvements.

Code:

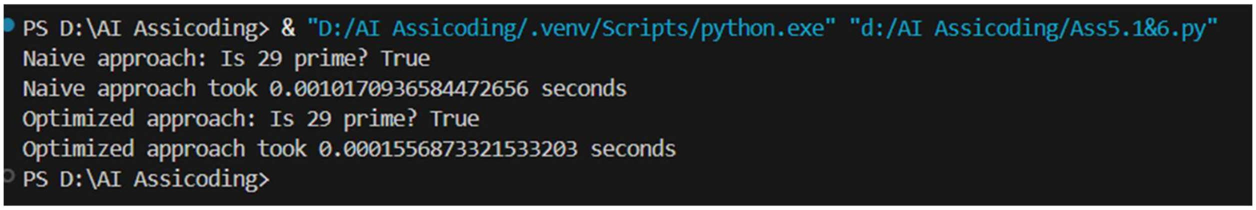


```python
'''
Generate a well commentd python code for checking if a number is prime or
not. i naive approach(basic) and optimized approach.
'''


import time
def is_prime_naive(n):
    """
    Check if a number is prime using a naive approach.

    A prime number is a natural number greater than 1 that cannot be
    formed by multiplying
    two smaller natural numbers.
    the naive approach checks for factorsfrom 2 to n-1.

    :param n: The number to check.
    :return: True if the number is prime, False otherwise.
    """
    if n <= 1:
        return False # Numbers less than or equal to 1 are not prime
    for i in range(2, n):
        if n % i == 0:
            return False # Found a factor, so n is not prime
    return True # No factors found, so n is prime
```



```python
def is_prime_optimized(n):
    """
    Check if a number is prime using an optimized approach.
    This approach reduces the number of checks needed by only testing for
    factors
    up to the square root of n and skipping even numbers after checking for 2.
    Parameters:
    n (int): The number to check for primality.
    Returns:
    bool: True if n is prime, False otherwise.
    """
    if n <= 1:
        return False # Numbers less than or equal to 1 are not prime
    if n <= 3:
        return True # 2 and 3 are prime numbers
    if n % 2 == 0 or n % 3 == 0:
        return False # Multiples of 2 and 3 are not prime
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False # Found a factor, so n is not prime
        i += 6
    return True # No factors found, so n is prime
# Example usage:
if __name__ == "__main__":
    number = 29
    start_time = time.time()
    print(f"Naive approach: Is {number} prime? {is_prime_naive(number)}")
    print(f"Naive approach took {time.time() - start_time} seconds")
    start_time = time.time()
    print(f"Optimized approach: Is {number} prime? {is_prime_optimized(number)}")
    print(f"Optimized approach took {time.time() - start_time} seconds")
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Naive approach: Is 29 prime? True
Naive approach took 0.0010170936584472656 seconds
Optimized approach: Is 29 prime? True
Optimized approach took 0.00015568733215332031 seconds
PS D:\AI Assicoding>
```

Task Description #9 (Transparency in Algorithm Optimization)

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.

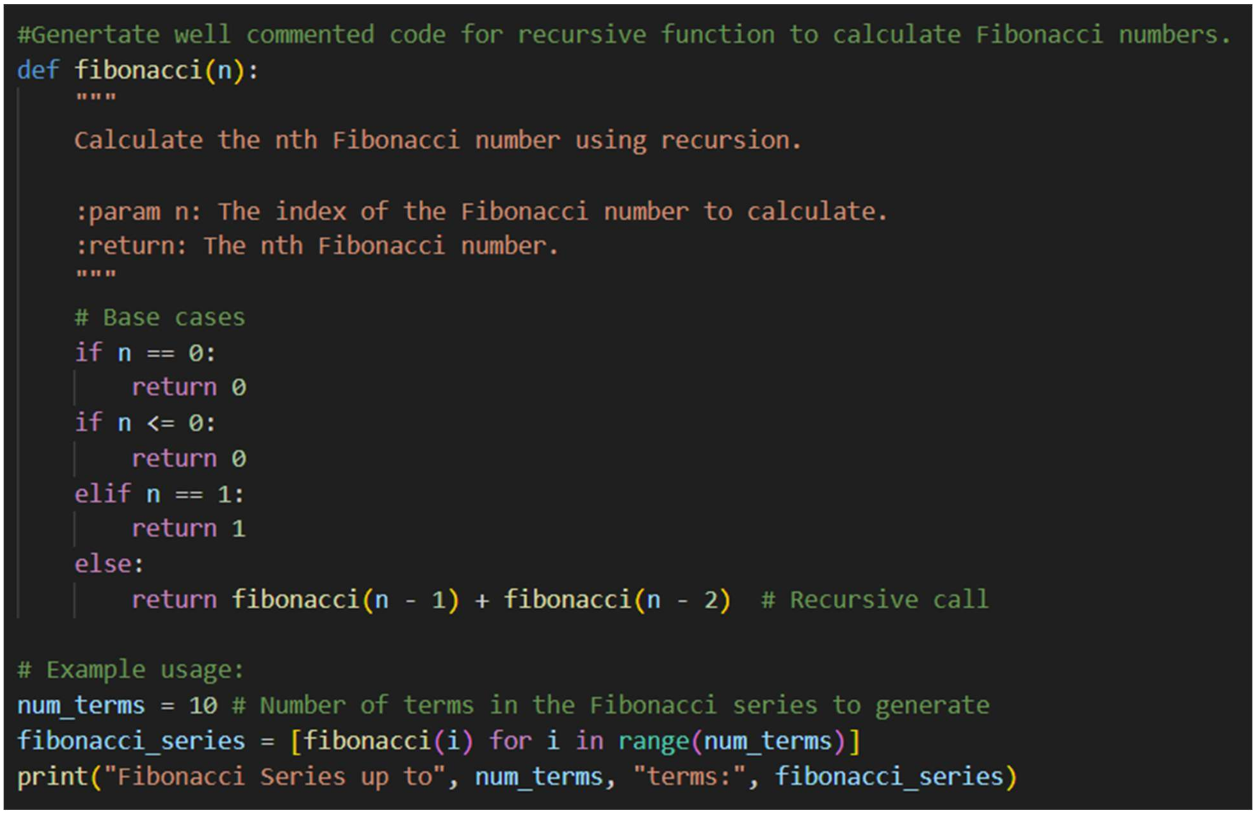
Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

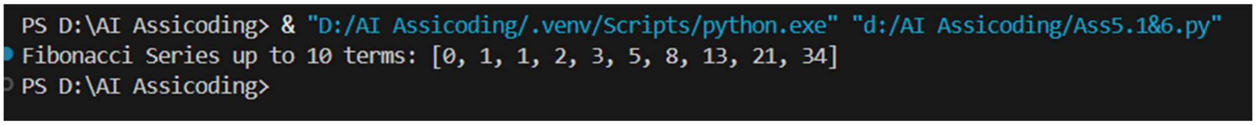
• Verification that explanation matches actual execution.

Code:



```python
#Genertate well commented code for recursive function to calculate Fibonacci numbers.
def fibonacci(n):
    """
    Calculate the nth Fibonacci number using recursion.

    :param n: The index of the Fibonacci number to calculate.
    :return: The nth Fibonacci number.
    """
    # Base cases
    if n == 0:
        return 0
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)  # Recursive call

# Example usage:
num_terms = 10 # Number of terms in the Fibonacci series to generate
fibonacci_series = [fibonacci(i) for i in range(num_terms)]
print("Fibonacci Series up to", num_terms, "terms:", fibonacci_series)
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
Fibonacci Series up to 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS D:\AI Assicoding>
```

Task Description #10 (Transparency in Error Handling)

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

Prompt:

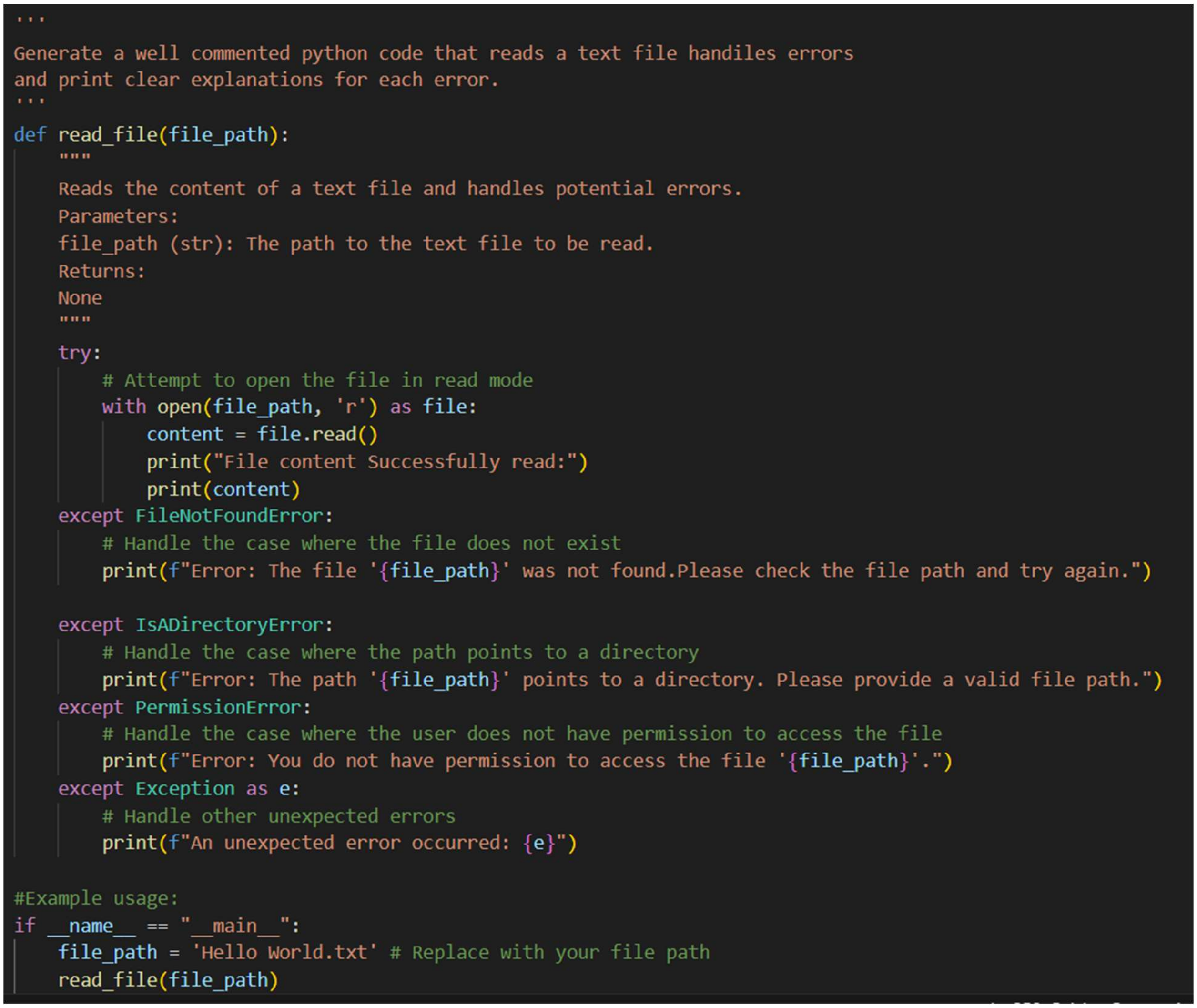
“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

- Code with meaningful exception handling.
- Clear comments explaining each error scenario.

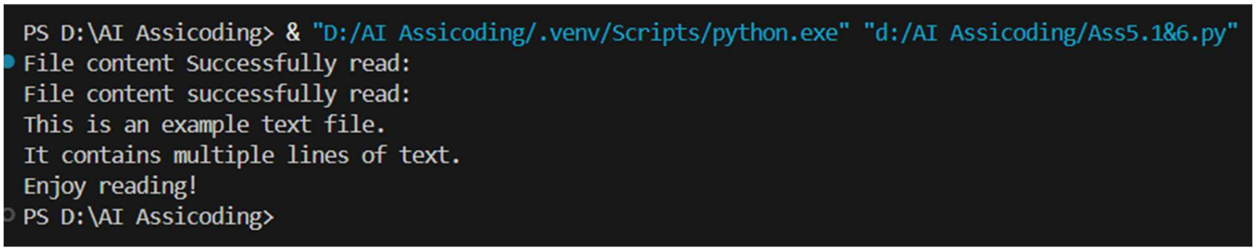
- Validation that explanations align with runtime behavior.

Code:



```
'''
Generate a well commented python code that reads a text file handiles errors
and print clear explanations for each error.
'''
def read_file(file_path):
    """
    Reads the content of a text file and handles potential errors.
    Parameters:
    file_path (str): The path to the text file to be read.
    Returns:
    None
    """
    try:
        # Attempt to open the file in read mode
        with open(file_path, 'r') as file:
            content = file.read()
            print("File content Successfully read:")
            print(content)
    except FileNotFoundError:
        # Handle the case where the file does not exist
        print(f"Error: The file '{file_path}' was not found.Please check the file path and try again.")

    except IsADirectoryError:
        # Handle the case where the path points to a directory
        print(f"Error: The path '{file_path}' points to a directory. Please provide a valid file path.")
    except PermissionError:
        # Handle the case where the user does not have permission to access the file
        print(f"Error: You do not have permission to access the file '{file_path}'.")
    except Exception as e:
        # Handle other unexpected errors
        print(f"An unexpected error occurred: {e}")

#Example usage:
if __name__ == "__main__":
    file_path = 'Hello World.txt' # Replace with your file path
    read_file(file_path)
```

Output:



```
PS D:\AI Assicoding> & "D:/AI Assicoding/.venv/Scripts/python.exe" "d:/AI Assicoding/Ass5.1&6.py"
File content Successfully read:
File content successfully read:
This is an example text file.
It contains multiple lines of text.
Enjoy reading!
PS D:\AI Assicoding>
```