AI Assisted Coding

Assignment Number:5.1 and 6

Name: Deepthi

Ht.No: 2303A52390

Batch: 32

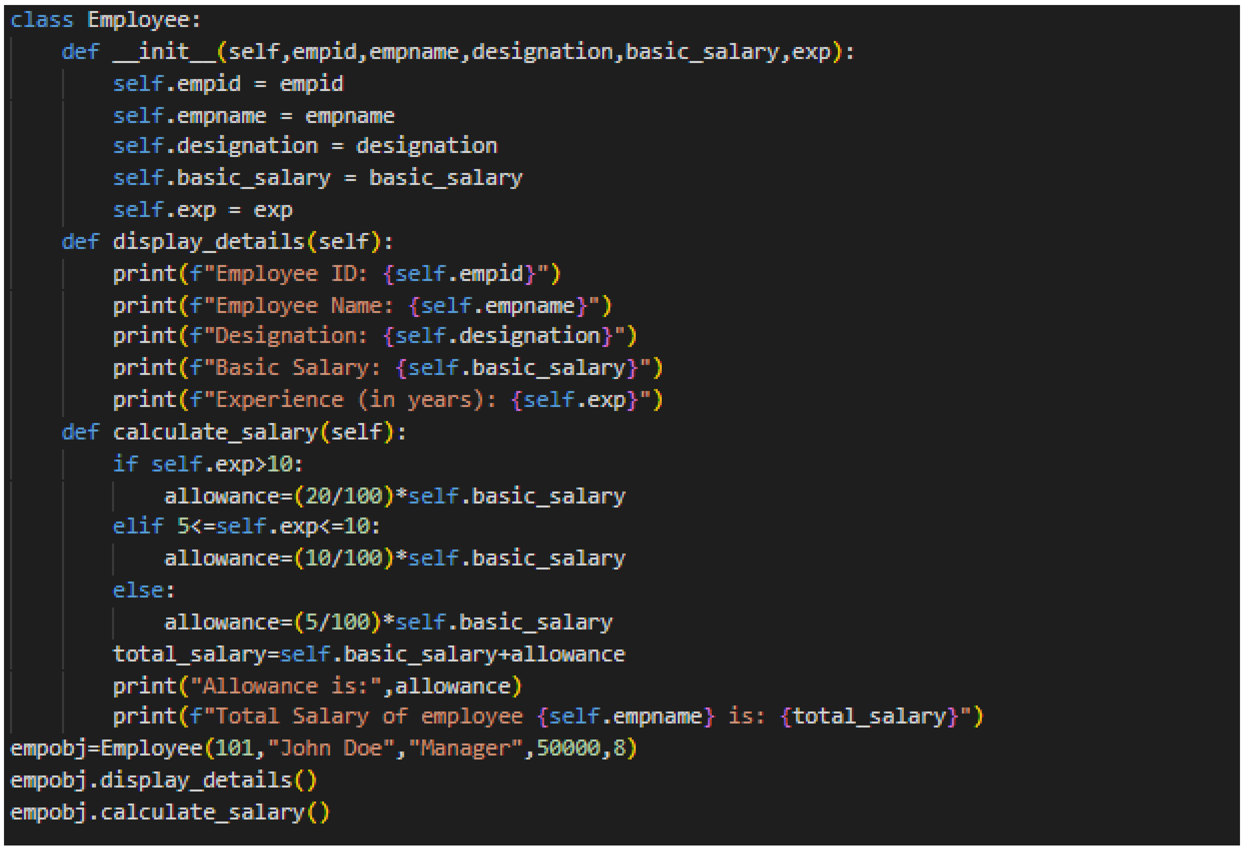
Task 1:

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`

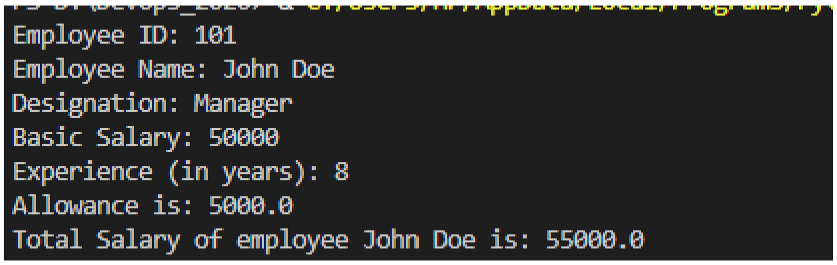
Finally, create at least one instance of the `Employee` class, call the `display_details()` method, and print the calculated allowance.

Code:



```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 (in years): {self.exp}")
    def calculate_salary(self):
        if self.exp>10:
            allowance=(20/100)*self.basic_salary
        elif 5<=self.exp<=10:
            allowance=(10/100)*self.basic_salary
        else:
            allowance=(5/100)*self.basic_salary
        total_salary=self.basic_salary+allowance
        print("Allowance is:",allowance)
        print(f"Total Salary of employee {self.empname} is: {total_salary}")
empobj=Employee(101,"John Doe","Manager",50000,8)
empobj.display_details()
empobj.calculate_salary()
```

Output:



```
Employee ID: 101
Employee Name: John Doe
Designation: Manager
Basic Salary: 50000
Experience (in years): 8
Allowance is: 5000.0
Total Salary of employee John Doe is: 55000.0
```

Explanation:

You should create an Employee class to store employee details like ID, name, designation, salary, and experience.Add a method to display all employee information in a readable format.Implement another method that calculates allowance using conditional statements based on years of experience. Finally, create an object of the class and call both methods to show employee details and allowance.

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:

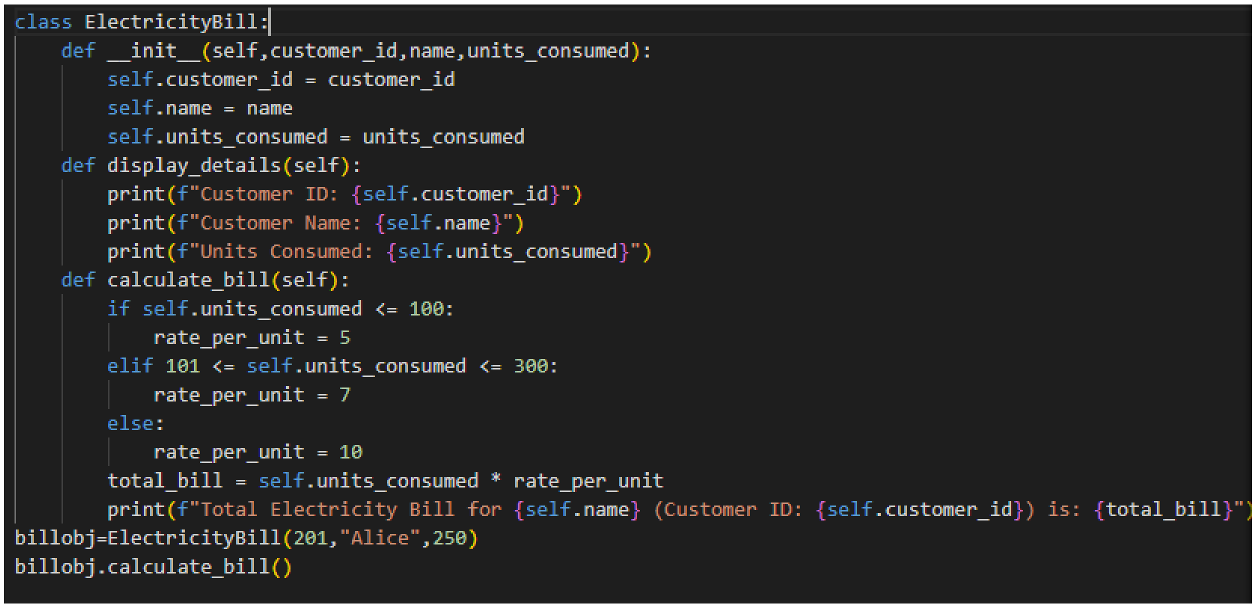
- Units ≤ 100 → ₹5 per unit

- 101 to 300 units → ₹7 per unit

- More than 300 units → ₹10 per unit

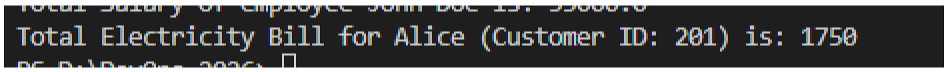
Create a bill object, display details, and print the total bill amount.

Code:



```python
class 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_per_unit = 5
        elif 101 <= self.units_consumed <= 300:
            rate_per_unit = 7
        else:
            rate_per_unit = 10
        total_bill = self.units_consumed * rate_per_unit
        print(f"Total Electricity Bill for {self.name} (Customer ID: {self.customer_id}) is: {total_bill}"
billobj=ElectricityBill(201,"Alice",250)
billobj.calculate_bill()
```

Output:



```
Total Electricity Bill for Alice (Customer ID: 201) is: 1750
```

Explanation:

Create an ElectricityBill class to store customer ID, name, and units consumed. Use a display_details() method to neatly print the customer information. Implement a calculate_bill() method with conditional logic to apply different rates based on unit ranges.Finally, create an object of the class and display the total bill amount.

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

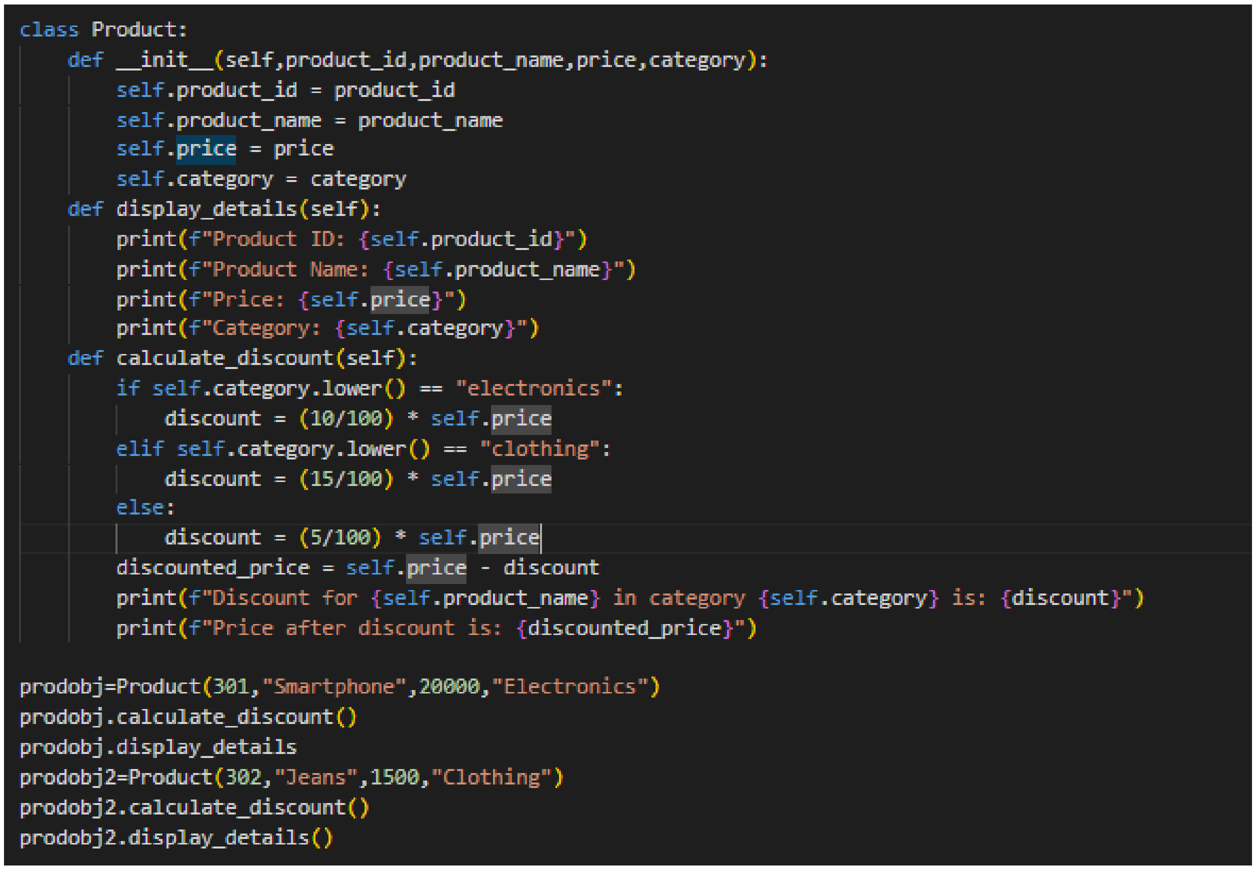
- Clothing → 15% discount

- Grocery → 5% discount

Create at least one product object, display details, and print the final

price after discount.

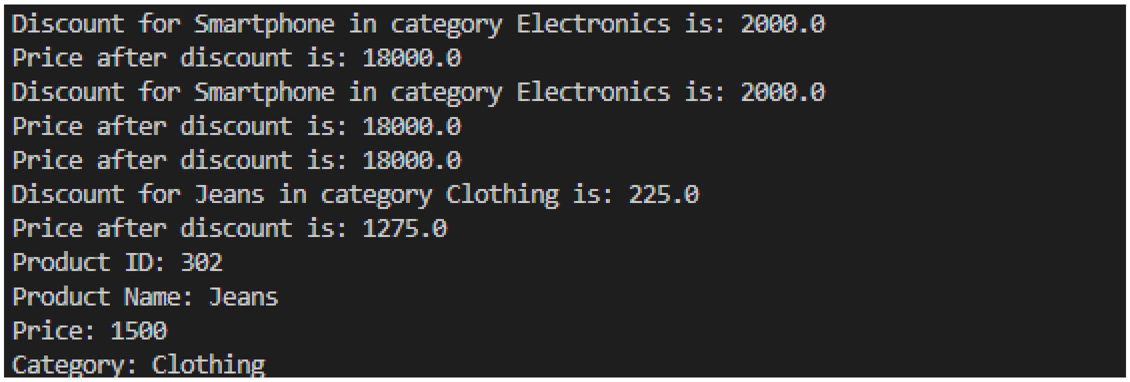
Code:



```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 = (10/100) * self.price
        elif self.category.lower() == "clothing":
            discount = (15/100) * self.price
        else:
            discount = (5/100) * self.price
        discounted_price = self.price - discount
        print(f"Discount for {self.product_name} in category {self.category} is: {discount}")
        print(f"Price after discount is: {discounted_price}")

prodobj=Product(301,"Smartphone",20000,"Electronics")
prodobj.calculate_discount()
prodobj.display_details
prodobj2=Product(302,"Jeans",1500,"Clothing")
prodobj2.calculate_discount()
prodobj2.display_details()
```

Output:



```
Discount for Smartphone in category Electronics is: 2000.0
Price after discount is: 18000.0
Discount for Smartphone in category Electronics is: 2000.0
Price after discount is: 18000.0
Price after discount is: 18000.0
Discount for Jeans in category Clothing is: 225.0
Price after discount is: 1275.0
Product ID: 302
Product Name: Jeans
Price: 1500
Category: Clothing
```

Explanation:

Create a Product class to store product ID, name, price, and category. Use a display_details() method to show all product information clearly. Implement a calculate_discount() method using conditional statements to apply discounts based on the product category. Finally, create a product object, display its details, and print the final price after applying the discount.

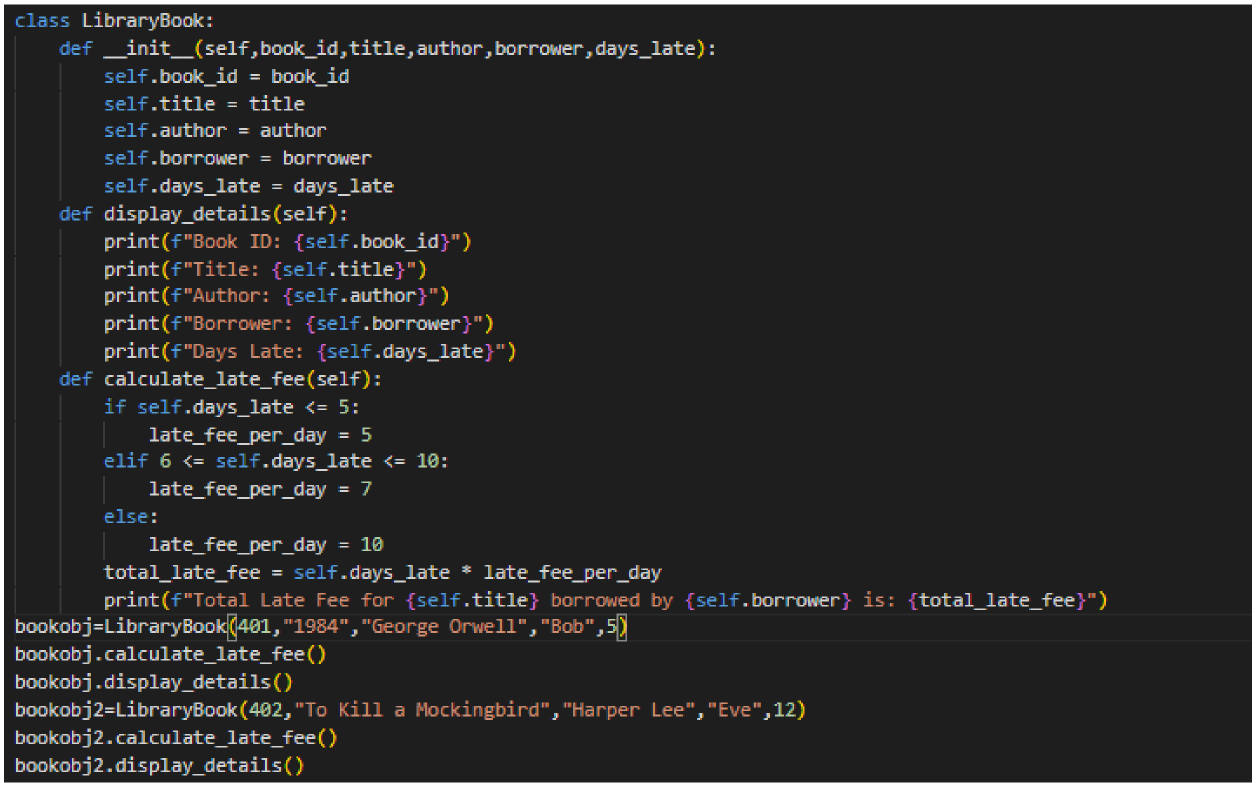
Task 4:

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

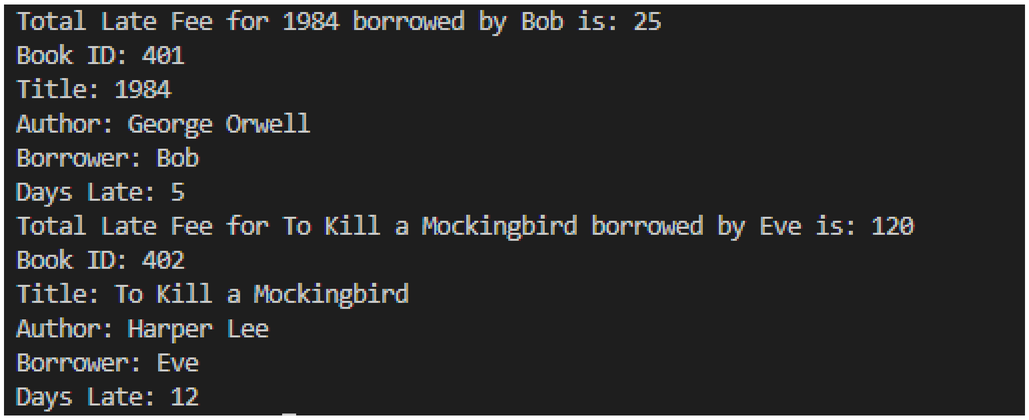
Create a book object, display details, and print the late fee.

Code:



```
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_per_day = 5
        elif 6 <= self.days_late <= 10:
            late_fee_per_day = 7
        else:
            late_fee_per_day = 10
        total_late_fee = self.days_late * late_fee_per_day
        print(f"Total Late Fee for {self.title} borrowed by {self.borrower} is: {total_late_fee}")
bookobj=LibraryBook(401,"1984","George Orwell","Bob",5)
bookobj.calculate_late_fee()
bookobj.display_details()
bookobj2=LibraryBook(402,"To Kill a Mockingbird","Harper Lee","Eve",12)
bookobj2.calculate_late_fee()
bookobj2.display_details()
```

Output:



```
Total Late Fee for 1984 borrowed by Bob is: 25
Book ID: 401
Title: 1984
Author: George Orwell
Borrower: Bob
Days Late: 5
Total Late Fee for To Kill a Mockingbird borrowed by Eve is: 120
Book ID: 402
Title: To Kill a Mockingbird
Author: Harper Lee
Borrower: Eve
Days Late: 12
```

Explanation:

Create a LibraryBook class to store book details such as ID, title, author, borrower, and days late.Use a display_details() method to neatly print all book information.Implement a calculate_late_fee() method using conditional statements to apply different charges based on the number of late days. Finally, create a book object, display its details, and print the total late fee.

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:

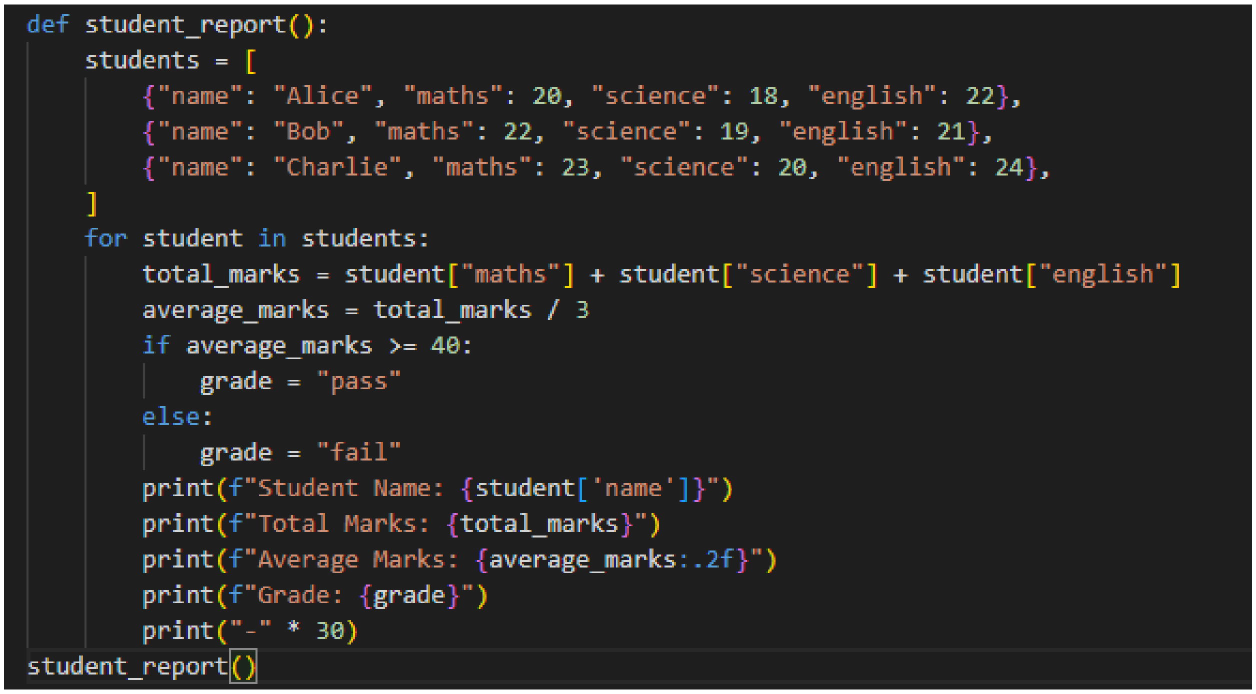
- Calculate the average score for each student

- Determine pass/fail status (pass ≥ 40)

- Return a summary report as a list of dictionaries

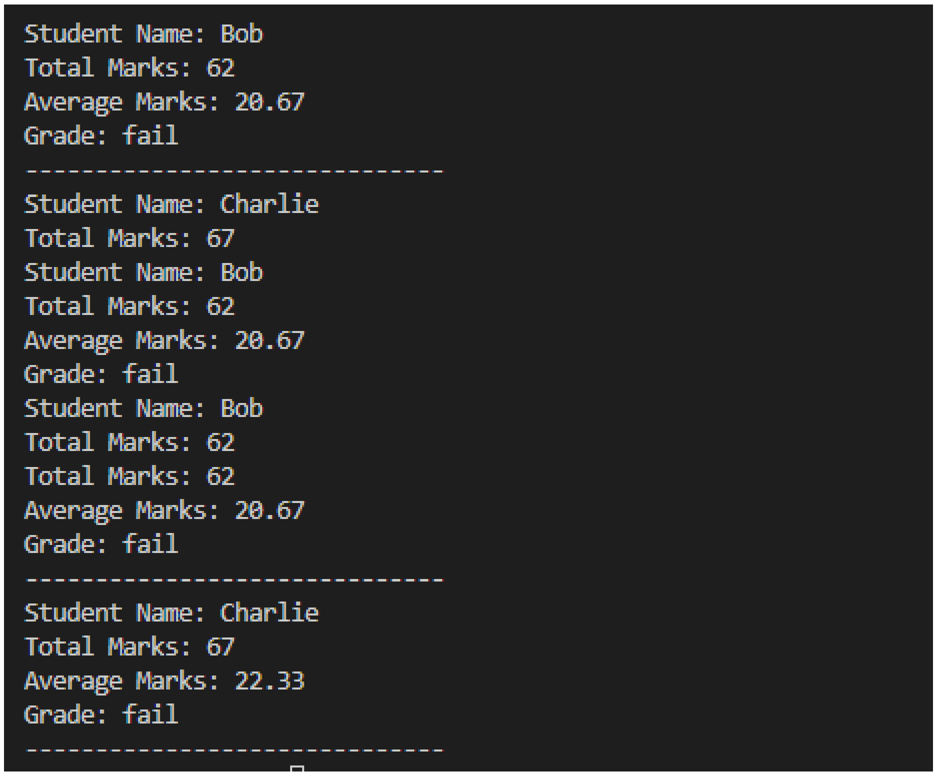
Use Copilot suggestions as you build the function and format the Output

Code:



```
def student_report():
    students = [
        {"name": "Alice", "maths": 20, "science": 18, "english": 22},
        {"name": "Bob", "maths": 22, "science": 19, "english": 21},
        {"name": "Charlie", "maths": 23, "science": 20, "english": 24},
    ]
    for student in students:
        total_marks = student["maths"] + student["science"] + student["english"]
        average_marks = total_marks / 3
        if average_marks >= 40:
            grade = "pass"
        else:
            grade = "fail"
        print(f"Student Name: {student['name']}")
        print(f"Total Marks: {total_marks}")
        print(f"Average Marks: {average_marks:.2f}")
        print(f"Grade: {grade}")
        print("-" * 30)
student_report()
```

Output:



```
Student Name: Bob
Total Marks: 62
Average Marks: 20.67
Grade: fail
------------------------------
Student Name: Charlie
Total Marks: 67
Student Name: Bob
Total Marks: 62
Average Marks: 20.67
Grade: fail
Student Name: Bob
Total Marks: 62
Total Marks: 62
Average Marks: 20.67
Grade: fail
------------------------------
Student Name: Charlie
Total Marks: 67
Average Marks: 22.33
Grade: fail
------------------------------
```

Explanation:

Create a student_report() function that takes a dictionary of student names and their marks as input.Loop through each student to calculate their average score.Use conditional statements to determine pass or fail based on the average (≥ 40).Store each student's summary in a list of dictionaries and return it as the final report.

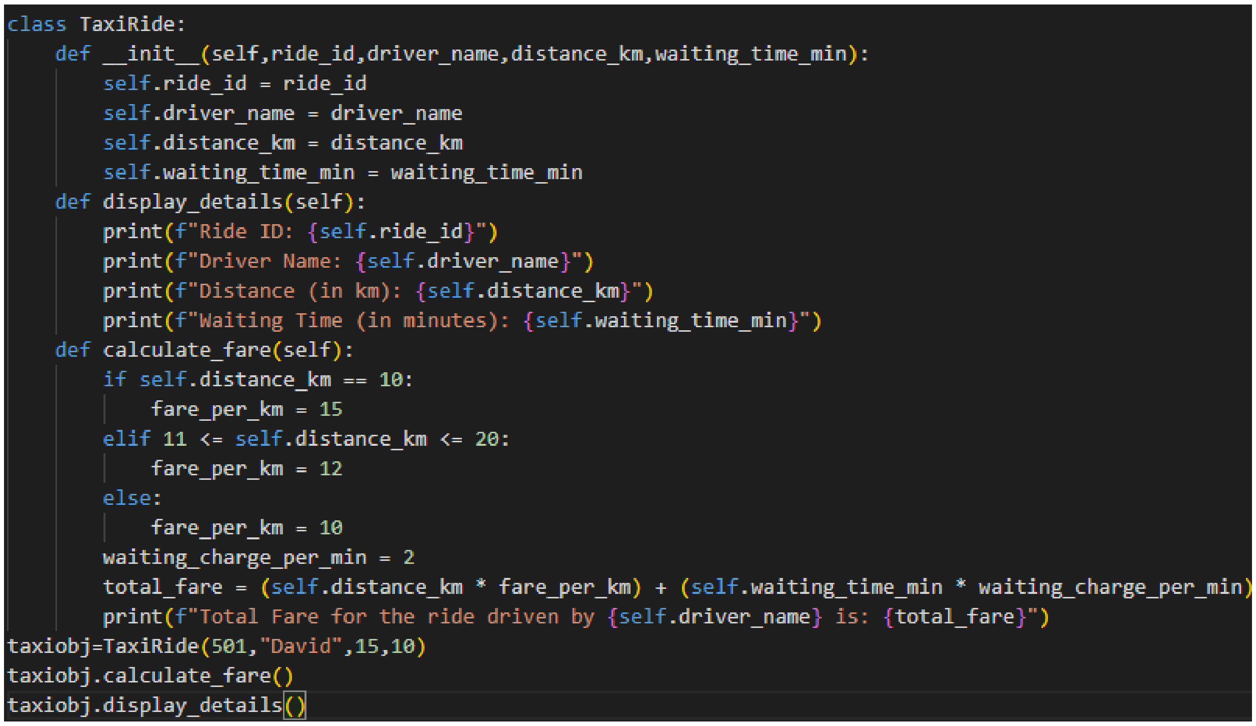
Task 6:

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

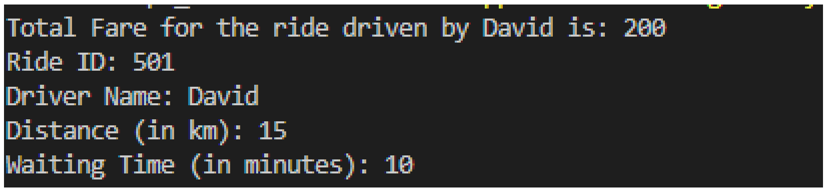
Create a ride object, display details, and print the total fare.

Code:



```python
class TaxiRide:
    def __init__(self,ride_id,driver_name,distance_km,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 (in km): {self.distance_km}")
        print(f"Waiting Time (in minutes): {self.waiting_time_min}")
    def calculate_fare(self):
        if self.distance_km == 10:
            fare_per_km = 15
        elif 11 <= self.distance_km <= 20:
            fare_per_km = 12
        else:
            fare_per_km = 10
        waiting_charge_per_min = 2
        total_fare = (self.distance_km * fare_per_km) + (self.waiting_time_min * waiting_charge_per_min)
        print(f"Total Fare for the ride driven by {self.driver_name} is: {total_fare}")
taxiobj=TaxiRide(501,"David",15,10)
taxiobj.calculate_fare()
taxiobj.display_details()
```

Output:



```
Total Fare for the ride driven by David is: 200
Ride ID: 501
Driver Name: David
Distance (in km): 15
Waiting Time (in minutes): 10
```

Explanation:

Create a TaxiRide class to store ride details such as ride ID, driver name, distance traveled, and waiting time.Use a display_details() method to show all ride information clearly.Implement a calculate_fare() method using conditional logic to apply different per-kilometer rates and add waiting charges.
Finally, create a ride object, display its details, and print the total fare amount.

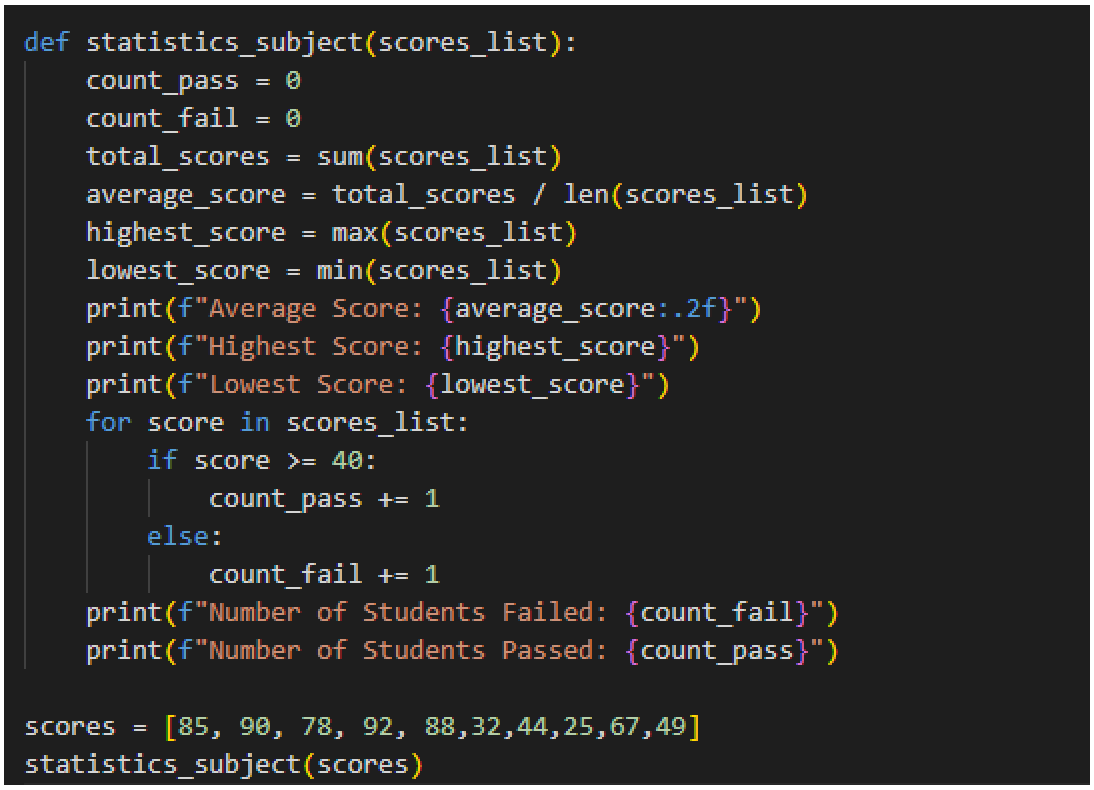
Task 7:

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

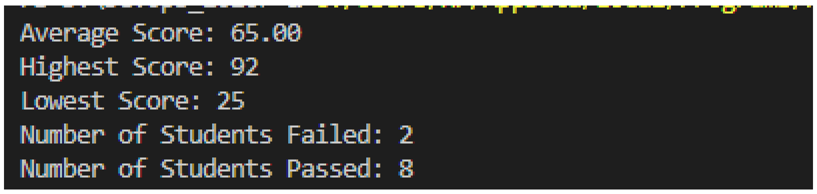
Code:



```python
def statistics_subject(scores_list):
    count_pass = 0
    count_fail = 0
    total_scores = sum(scores_list)
    average_score = total_scores / len(scores_list)
    highest_score = max(scores_list)
    lowest_score = min(scores_list)
    print(f"Average Score: {average_score:.2f}")
    print(f"Highest Score: {highest_score}")
    print(f"Lowest Score: {lowest_score}")
    for score in scores_list:
        if score >= 40:
            count_pass += 1
        else:
            count_fail += 1
    print(f"Number of Students Failed: {count_fail}")
    print(f"Number of Students Passed: {count_pass}")

scores = [85, 90, 78, 92, 88,32,44,25,67,49]
statistics_subject(scores)
```

Output:



```
Average Score: 65.00
Highest Score: 92
Lowest Score: 25
Number of Students Failed: 2
Number of Students Passed: 8
```

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.”

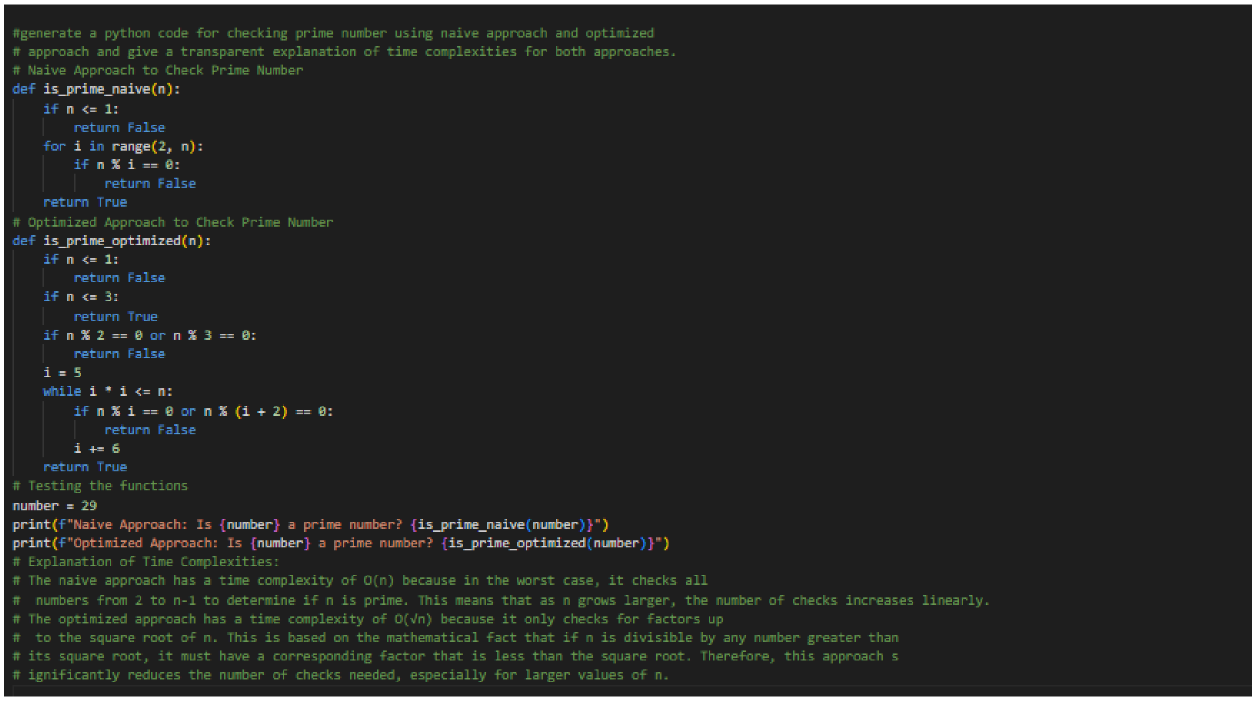
Expected Output:

• Code for both methods.

• Transparent explanation of time complexity.

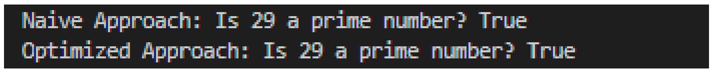
• Comparison highlighting efficiency improvements.

Code:



```python
#generate a python code for checking prime number using naive approach and optimized
# approach and give a transparent explanation of time complexities for both approaches.
# Naive Approach to Check Prime Number
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 to Check Prime Number
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
# Testing the functions
number = 29
print(f"Naive Approach: Is {number} a prime number? {is_prime_naive(number)}")
print(f"Optimized Approach: Is {number} a prime number? {is_prime_optimized(number)}")
# Explanation of Time Complexities:
# The naive approach has a time complexity of O(n) because in the worst case, it checks all
#  numbers from 2 to n-1 to determine if n is prime. This means that as n grows larger, the number of checks increases linearly.
# The optimized approach has a time complexity of O(√n) because it only checks for factors up
#  to the square root of n. This is based on the mathematical fact that if n is divisible by any number greater than
# its square root, it must have a corresponding factor that is less than the square root. Therefore, this approach s
# ignificantly reduces the number of checks needed, especially for larger values of n.
```

Output:



```
Naive Approach: Is 29 a prime number? True
Optimized Approach: Is 29 a prime number? True
```

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.

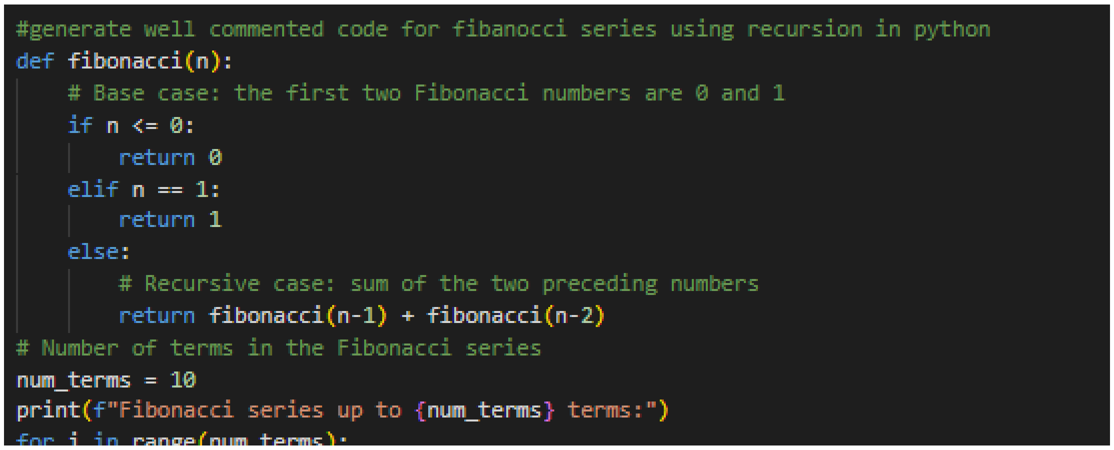
Expected Output:

• Well-commented recursive code.

• Clear explanation of how recursion works.

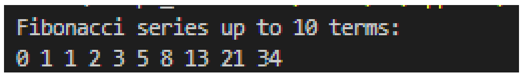
• Verification that explanation matches actual execution

Code:



```
#generate well commented code for fibanocci series using recursion in python
def fibonacci(n):
    # Base case: the first two Fibonacci numbers are 0 and 1
    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)
# Number of terms in the Fibonacci series
num_terms = 10
print(f"Fibonacci series up to {num_terms} terms:")
for i in range(num_terms):
```

Output:



```
Fibonacci series up to 10 terms:
0 1 1 2 3 5 8 13 21 34
```

Explanation:

The recursive Fibonacci function works by calling itself to find earlier Fibonacci values until it reaches base cases like 0 or 1, where the f unction stops. These base cases prevent infinite recursion and give direct results. Each recursive call solves a smaller version of the same problem, making the logic easy to follow. Clear comments explain how the function moves between calls. The explanation matches actual execution because the function follows the same step-by-step recursive process.

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.”

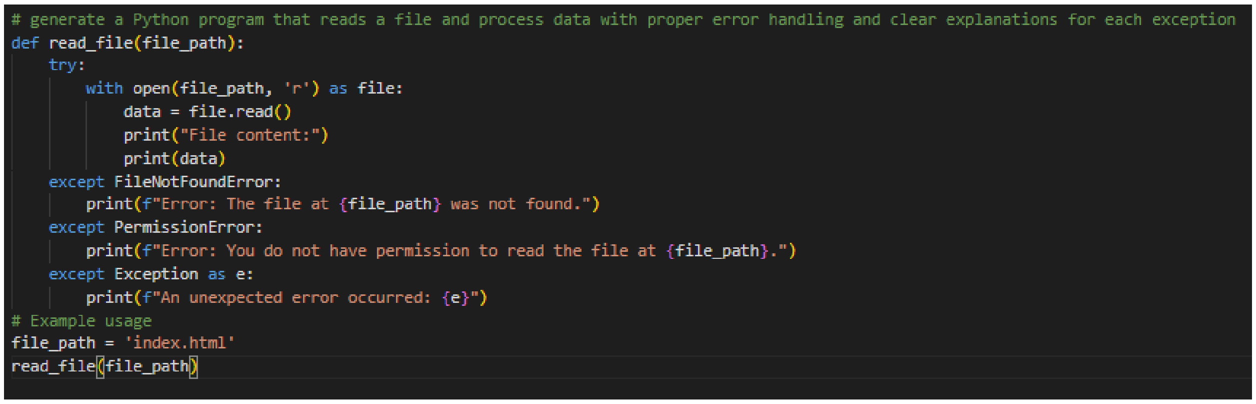
Expected Output:

• Code with meaningful exception handling.

• Clear comments explaining each error scenario.

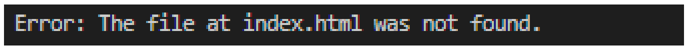
• Validation that explanations align with runtime behavior.

Code:



```python
# generate a Python program that reads a file and process data with proper error handling and clear explanations for each exception
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            data = file.read()
            print("File content:")
            print(data)
    except FileNotFoundError:
        print(f"Error: The file at {file_path} was not found.")
    except PermissionError:
        print(f"Error: You do not have permission to read the file at {file_path}.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
# Example usage
file_path = 'index.html'
read_file(file_path)
```

Output:



```
Error: The file at index.html was not found.
```