Lab Assignment 5.1 & 6.1

Name : N. Nanda Mukesh

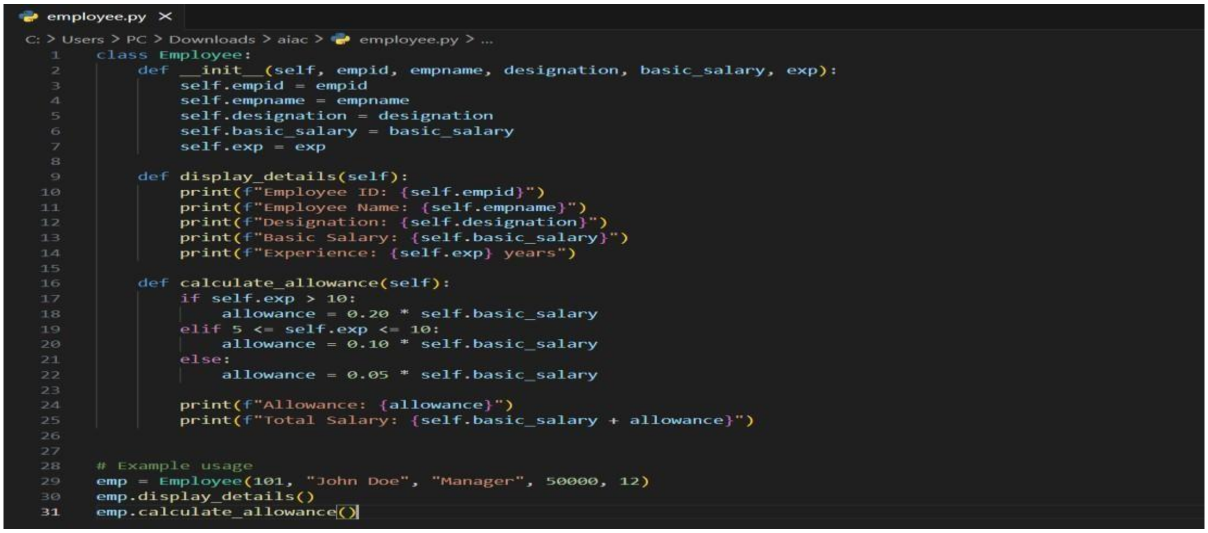
Ht.no : 2303A51410

Batch-24

Task 1:

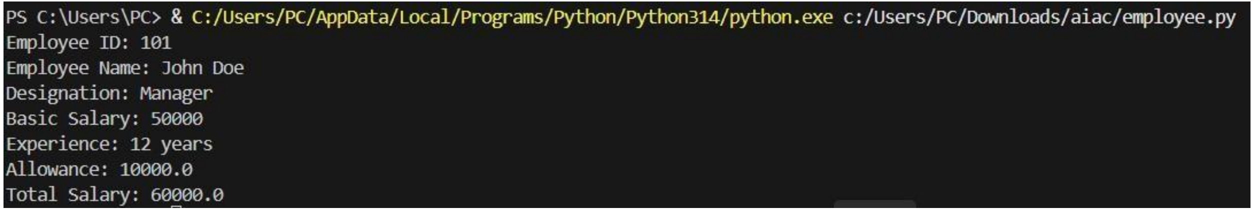
Employee Data: Create Python code that defines a class named `Employee` with the following attributes: `empid`, `empname`, `designation`, `basic_salary`, and `exp`. Implement a method `display_details()` to print all employee details. Implement another method `calculate_allowance()` to determine additional 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: {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}")


# Example usage
emp = Employee(101, "John Doe", "Manager", 50000, 12)
emp.display_details()
emp.calculate_allowance()
```

Output :

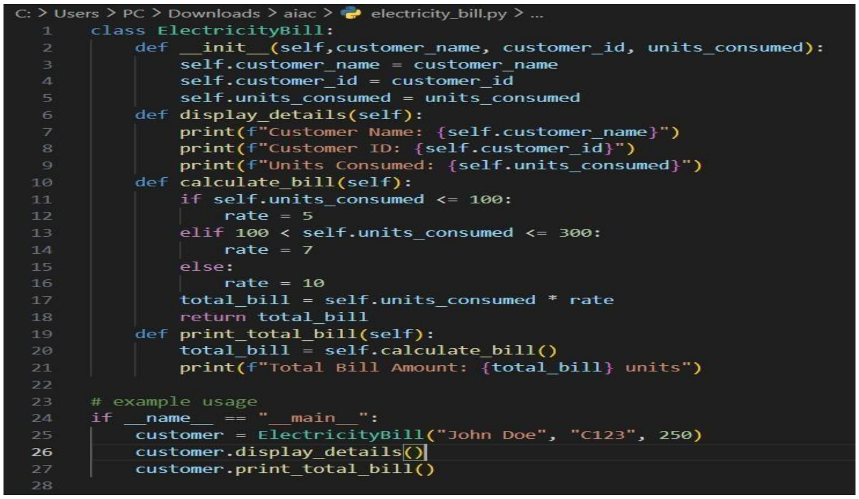


```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/employee.py
Employee ID: 101
Employee Name: John Doe
Designation: Manager
Basic Salary: 50000
Experience: 12 years
Allowance: 10000.0
Total Salary: 60000.0
```

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.

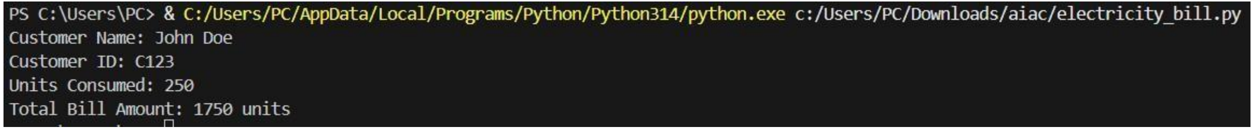
Code :



C: > Users > PC > Downloads > aiac > electricity_bill.py > ...

```python
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:
            rate = 5
        elif 100 < self.units_consumed <= 300:
            rate = 7
        else:
            rate = 10
        total_bill = self.units_consumed * rate
        return total_bill
    def print_total_bill(self):
        total_bill = self.calculate_bill()
        print(f"Total Bill Amount: {total_bill} units")

# example usage
if __name__ == "__main__":
    customer = ElectricityBill("John Doe", "C123", 250)
    customer.display_details()
    customer.print_total_bill()
```

Output:



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/electricity_bill.py
Customer Name: John Doe
Customer ID: C123
Units Consumed: 250
Total Bill Amount: 1750 units
```

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:

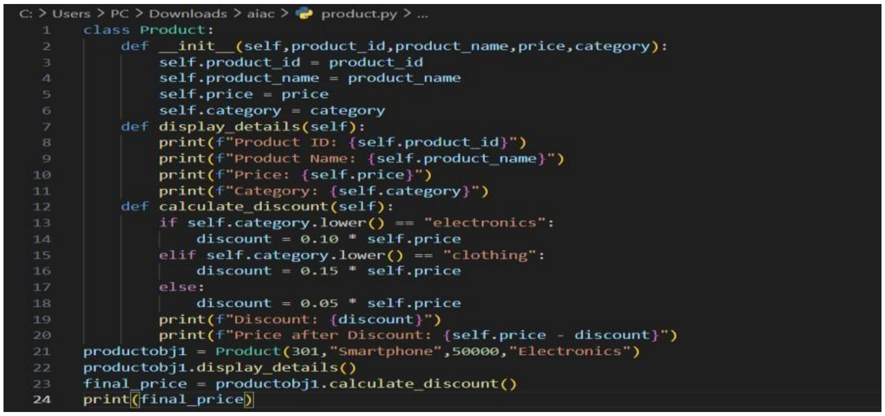
- Electronics → 10% discount

- Clothing → 15% discount

- Grocery → 5% discount

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

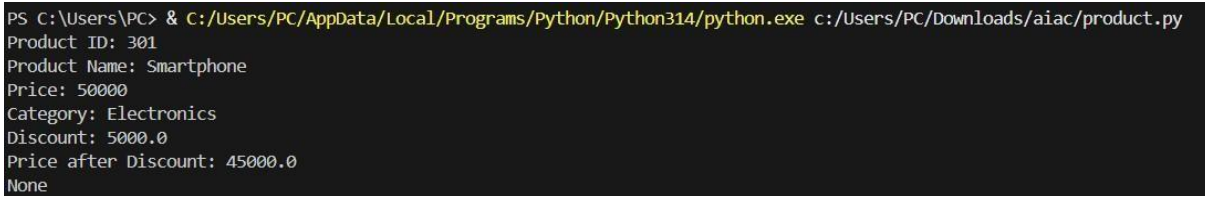
Code :



C: > Users > PC > Downloads > aiac > product.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.lower() == "electronics":
            discount = 0.10 * self.price
        elif self.category.lower() == "clothing":
            discount = 0.15 * self.price
        else:
            discount = 0.05 * self.price
        print(f"Discount: {discount}")
        print(f"Price after Discount: {self.price - discount}")
productobj1 = Product(301,"Smartphone",50000,"Electronics")
productobj1.display_details()
final_price = productobj1.calculate_discount()
print(final_price)
```

Output :



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/product.py
Product ID: 301
Product Name: Smartphone
Price: 50000
Category: Electronics
Discount: 5000.0
Price after Discount: 45000.0
None
```

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:

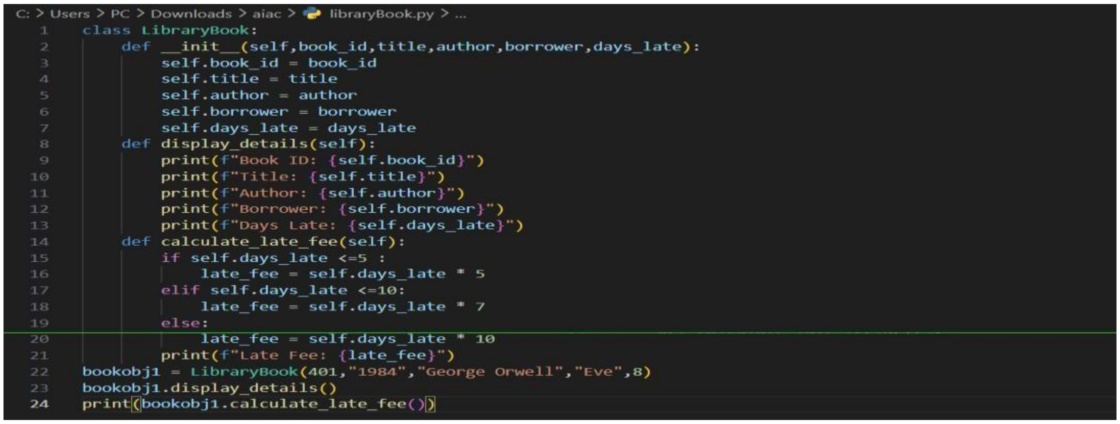
- Days late ≤ 5 → ₹5 per day

- 6 to 10 days late → ₹7 per day

- More than 10 days late → ₹10 per day

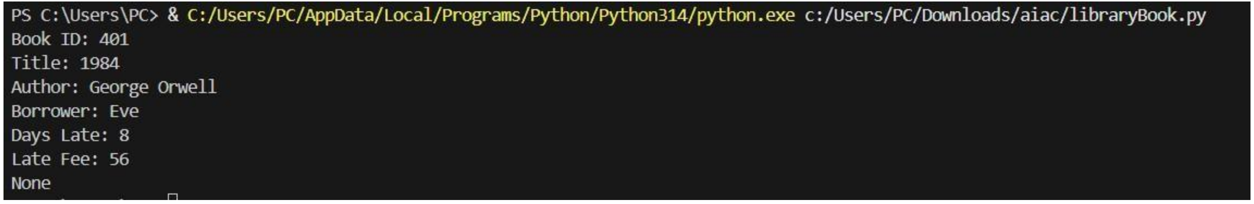
Create a book object, display details, and print the late fee.

Code :



```
C: > Users > PC > Downloads > aiac > libraryBook.py > ...
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 self.days_late <=10:
            late_fee = self.days_late * 7
        else:
            late_fee = self.days_late * 10
        print(f"Late Fee: {late_fee}")
bookobj1 = LibraryBook(401,"1984","George Orwell","Eve",8)
bookobj1.display_details()
print(bookobj1.calculate_late_fee())
```

Output:



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/libraryBook.py
Book ID: 401
Title: 1984
Author: George Orwell
Borrower: Eve
Days Late: 8
Late Fee: 56
None
```

Task 5:

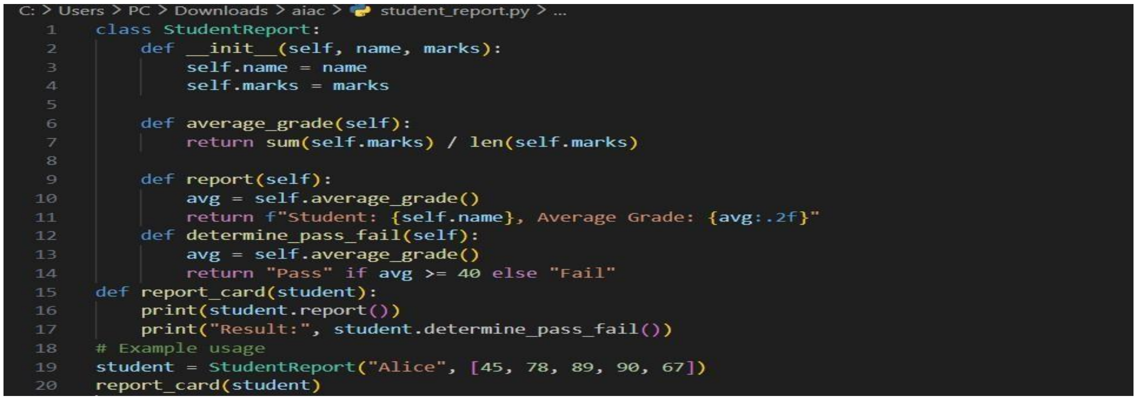
Student Performance Report - Define a function `student_report(student_data)` that accepts a dictionary containing student names and their marks. The function should: - Calculate the average score for each student

- Determine pass/fail status (pass ≥ 40)

- Return a summary report as a list of dictionaries

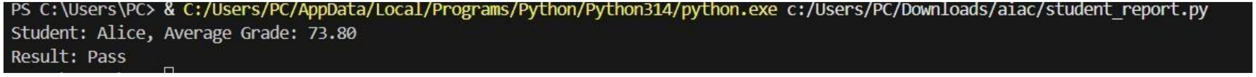
Use Copilot suggestions as you build the function and format the output.

Code :



```
C: > Users > PC > Downloads > aiac > student_report.py > ...
class StudentReport:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def average_grade(self):
        return sum(self.marks) / len(self.marks)

    def report(self):
        avg = self.average_grade()
        return f"Student: {self.name}, Average Grade: {avg:.2f}"
    def determine_pass_fail(self):
        avg = self.average_grade()
        return "Pass" if avg >= 40 else "Fail"
def report_card(student):
    print(student.report())
    print("Result:", student.determine_pass_fail())
# Example usage
student = StudentReport("Alice", [45, 78, 89, 90, 67])
report_card(student)
```

Output :



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/student_report.py
Student: Alice, Average Grade: 73.80
Result: Pass
```

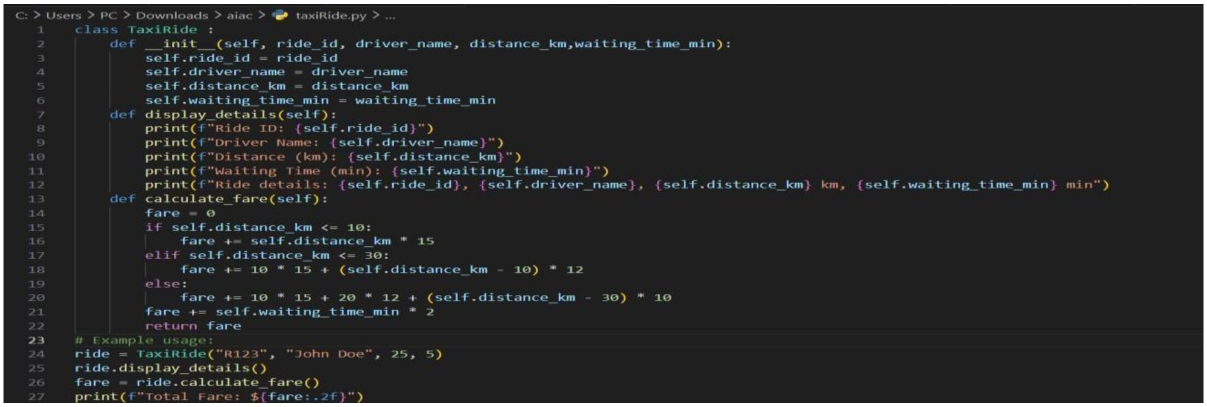
Task 6:

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

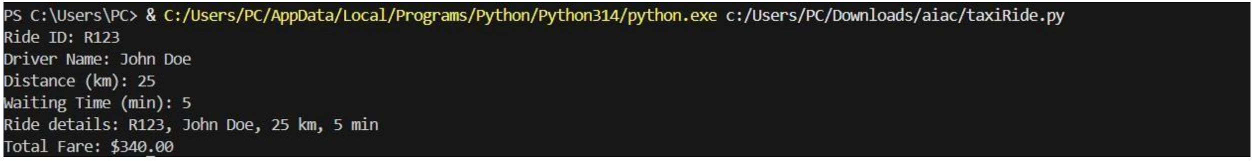
Create a ride object, display details, and print the total fare.

Code:



```
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 (km): {self.distance_km}")
        print(f"Waiting Time (min): {self.waiting_time_min}")
        print(f"Ride details: {self.ride_id}, {self.driver_name}, {self.distance_km} km, {self.waiting_time_min} min")
    def calculate_fare(self):
        fare = 0
        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_min * 2
        return fare
# Example usage:
ride = TaxiRide("R123", "John Doe", 25, 5)
ride.display_details()
fare = ride.calculate_fare()
print(f"Total Fare: ${fare:.2f}")
```

Output :



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/taxiRide.py
Ride ID: R123
Driver Name: John Doe
Distance (km): 25
Waiting Time (min): 5
Ride details: R123, John Doe, 25 km, 5 min
Total Fare: $340.00
```

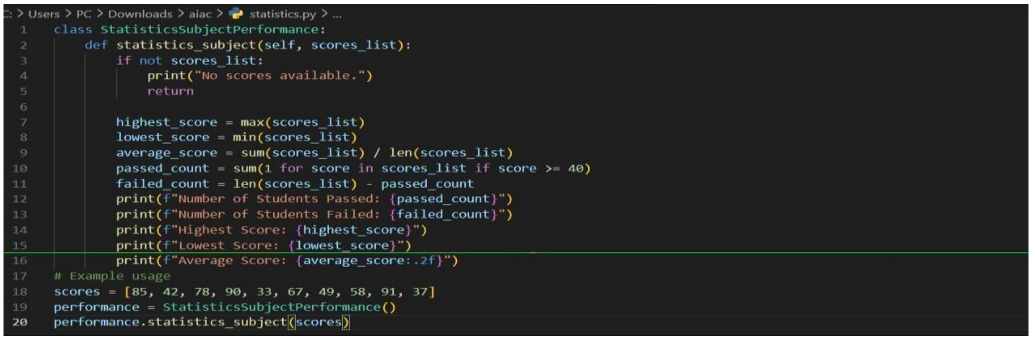
Task 7:

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)

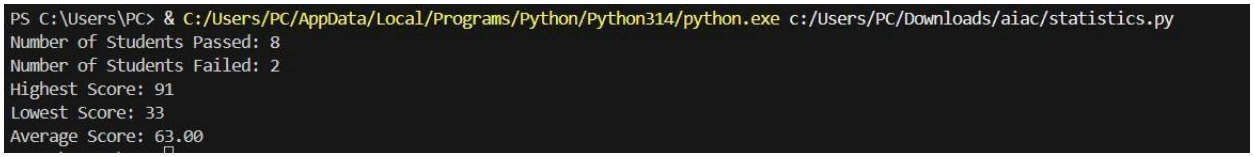
Allow Copilot to assist with aggregations and logic

Code :



```
class StatisticsSubjectPerformance:
    def statistics_subject(self, scores_list):
        if not scores_list:
            print("No scores available.")
            return

        highest_score = max(scores_list)
        lowest_score = min(scores_list)
        average_score = sum(scores_list) / len(scores_list)
        passed_count = sum(1 for score in scores_list if score >= 40)
        failed_count = len(scores_list) - passed_count
        print(f"Number of Students Passed: {passed_count}")
        print(f"Number of Students Failed: {failed_count}")
        print(f"Highest Score: {highest_score}")
        print(f"Lowest Score: {lowest_score}")
        print(f"Average Score: {average_score:.2f}")
# Example usage
scores = [85, 42, 78, 90, 33, 67, 49, 58, 91, 37]
performance = StatisticsSubjectPerformance()
performance.statistics_subject(scores)
```

Output:



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/statistics.py
Number of Students Passed: 8
Number of Students Failed: 2
Highest Score: 91
Lowest Score: 33
Average Score: 63.00
```

Task Description #8 (Transparency in Algorithm Optimization) Task:

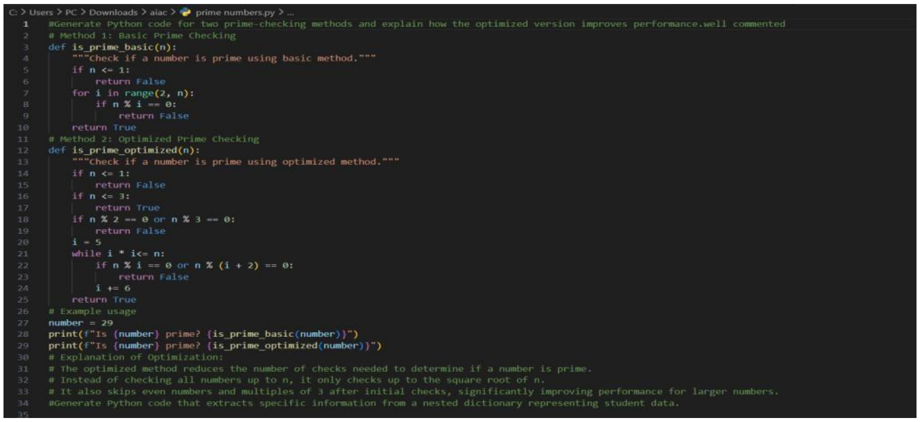
Use AI to generate two solutions for checking prime numbers:

- 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.

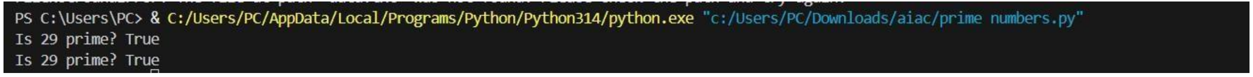
Code :



C: > Users > PC > Downloads > aiac > prime numbers.py > ...

```python
#Generate Python code for two prime-checking methods and explain how the optimized version improves performance.well commented
# Method 1: Basic Prime Checking
def is_prime_basic(n):
    """Check if a number is prime using basic method."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
# Method 2: Optimized Prime Checking
def is_prime_optimized(n):
    """Check if a number is prime using optimized method."""
    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
# Example usage
number = 29
print(f"Is {number} prime? {is_prime_basic(number)}")
print(f"Is {number} prime? {is_prime_optimized(number)}")
# Explanation of Optimization:
# The optimized method reduces the number of checks needed to determine if a number is prime.
# Instead of checking all numbers up to n, it only checks up to the square root of n.
# It also skips even numbers and multiples of 3 after initial checks, significantly improving performance for larger numbers.
#Generate Python code that extracts specific information from a nested dictionary representing student data.
```

Output:



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/PC/Downloads/aiac/prime numbers.py"
Is 29 prime? True
Is 29 prime? True
```

Task Description #9 (Transparency in Recursive Algorithms)

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

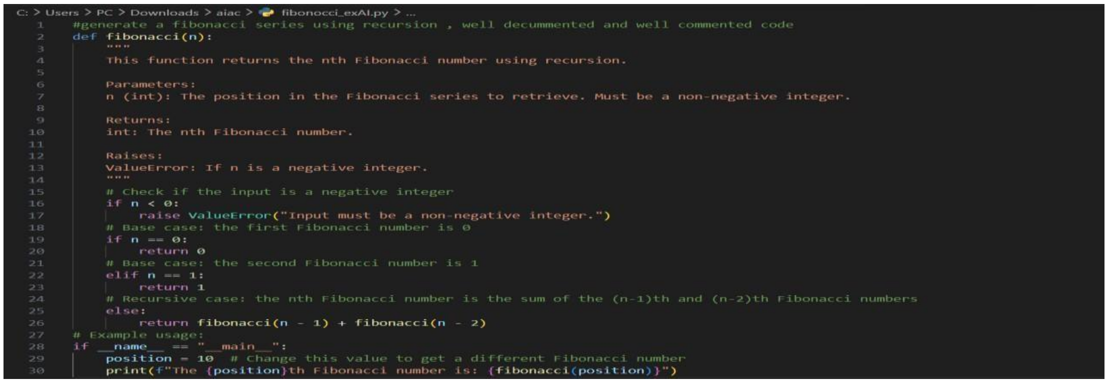
Instructions:

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.

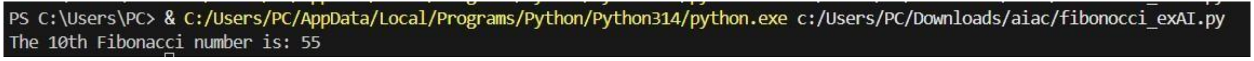
Code:



C: > Users > PC > Downloads > aiac > fibonocci_exAI.py > ...

```python
#generate a fibonacci series using recursion , well decummented and well commented code
def fibonacci(n):
    """
    This function returns the nth Fibonacci number using recursion.

    Parameters:
    n (int): The position in the Fibonacci series to retrieve. Must be a non-negative integer.

    Returns:
    int: The nth Fibonacci number.

    Raises:
    ValueError: If n is a negative integer.
    """
    # Check if the input is a negative integer
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    # Base case: the first Fibonacci number is 0
    if n == 0:
        return 0
    # Base case: the second Fibonacci number is 1
    elif n == 1:
        return 1
    # Recursive case: the nth Fibonacci number is the sum of the (n-1)th and (n-2)th Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage:
if __name__ == "__main__":
    position = 10  # Change this value to get a different Fibonacci number
    print(f"The {position}th Fibonacci number is: {fibonacci(position)}")
```

Output :



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/fibonocci_exAI.py
The 10th Fibonacci number is: 55
```

Task Description #10 (Transparency in Error Handling)

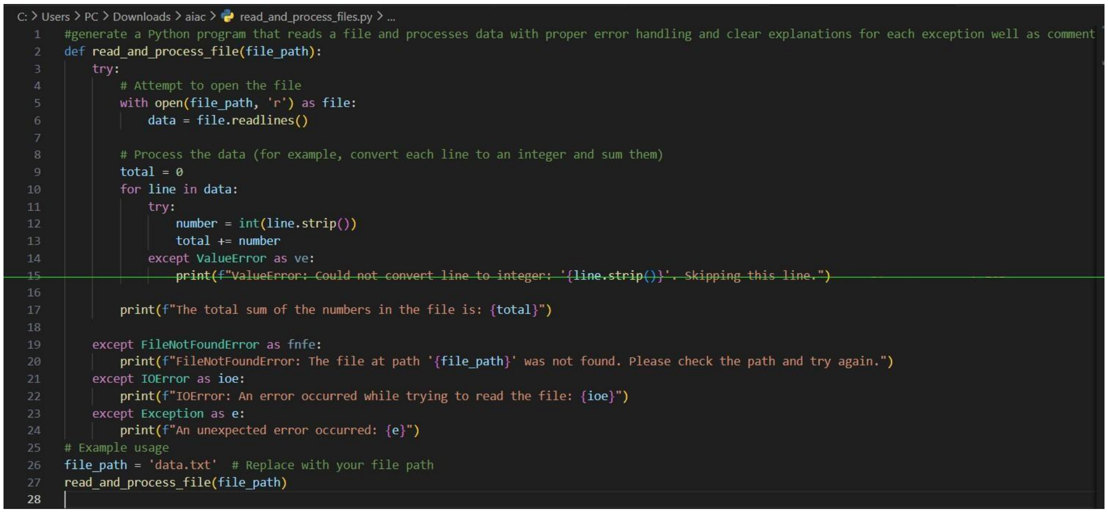
Task: Use AI to generate a Python program that reads a file and processes data.

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.

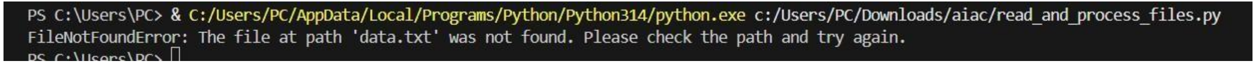
Code:



C: > Users > PC > Downloads > aiac > read_and_process_files.py > ...

```python
#generate a Python program that reads a file and processes data with proper error handling and clear explanations for each exception well as comment
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 example, convert each line to an integer and sum them)
        total = 0
        for line in data:
            try:
                number = int(line.strip())
                total += number
            except ValueError as ve:
                print(f"ValueError: Could not convert line to integer: '{line.strip()}'. Skipping this line.")

        print(f"The total sum of the numbers in the file is: {total}")

    except FileNotFoundError as fnfe:
        print(f"FileNotFoundError: The file at path '{file_path}' was not found. Please check the path and try again.")
    except IOError as ioe:
        print(f"IOError: An error occurred while trying to read the file: {ioe}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
# Example usage
file_path = 'data.txt'  # Replace with your file path
read_and_process_file(file_path)
```

Output:



```
PS C:\Users\PC> & C:/Users/PC/AppData/Local/Programs/Python/Python314/python.exe c:/Users/PC/Downloads/aiac/read_and_process_files.py
FileNotFoundError: The file at path 'data.txt' was not found. Please check the path and try again.
PS C:\Users\PC>
```