AI ASSISTED CODING

LAB ASSIGNMENT – 5.1 & 6.1

NAME:- CH VISHWANATH RAO

BATCH:02

HT NO:- 2303A51095

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.

SCREENSHOT OF CODE:-

```python
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(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_allowance(self):
        if self.experience > 10:
            allowance = 0.10 * self.basicsalary
        elif self.experience >= 5:
            allowance = 0.07 * self.basicsalary
        else:
            allowance = 0.05 * self.basicsalary
        return allowance

emp1 = Employee(101, "PRABHAS", "Manager", 50000, 15)
emp1.display()
allowance = emp1.calculate_allowance()
print("Allowance:", 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(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_allowance(self):
        if self.experience > 10:
            allowance = 0.10 * self.basicsalary
```

```
    elif self.experience >= 5:
        allowance = 0.07 * self.basicsalary
    else:
        allowance = 0.05 * self.basicsalary
    return allowance

emp1 = Employee(101, "PRABHAS", "Manager", 50000, 15)
emp1.display()
allowance = emp1.calculate_allowance()
print("Allowance:", allowance)
```

OUTPUT:-

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.

SCREENSHOT OF THE CODE:’

```
class ElectricityBill:
    def __init__(self, customerid, name, units_consumed):
        self.customerid = customerid
        self.name = name
        self.units_consumed = units_consumed

    def calculate_bill(self):
        if self.units_consumed <= 100:
            bill_amount = self.units_consumed * 5
        elif self.units_consumed <= 300:
            bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
        else:
            bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10
        return bill_amount
    def display(self):
        print("Customer ID:", self.customerid)
        print("Customer Name:", self.name)
        print("Units Consumed:", self.units_consumed)
bill1 = ElectricityBill(201, "VISHWA", 350)
bill1.display()
bill_amount = bill1.calculate_bill()
print("Electricity Bill Amount:", bill_amount)
```

CODE:

```
class ElectricityBill:
  def __init__(self, customerid, name, units_consumed):
    self.customerid = customerid
    self.name = name
    self.units_consumed = units_consumed


  def calculate_bill(self):
    if self.units_consumed <= 100:
      bill_amount = self.units_consumed * 5
    elif self.units_consumed <= 300:
      bill_amount = (100 * 5) + (self.units_consumed - 100) * 7
    else:
      bill_amount = (100 * 5) + (200 * 7) + (self.units_consumed - 300) * 10
    return bill_amount
  def display(self):
    print("Customer ID:", self.customerid)
    print("Customer Name:", self.name)
    print("Units Consumed:", self.units_consumed)
```

```
bill1 = ElectricityBill(201, "VISHWA", 350)
bill1.display()
bill_amount = bill1.calculate_bill()
print("Electricity Bill Amount:", bill_amount)
```

OUTPUT:

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.

SCREENSHOT OF THE CODE:

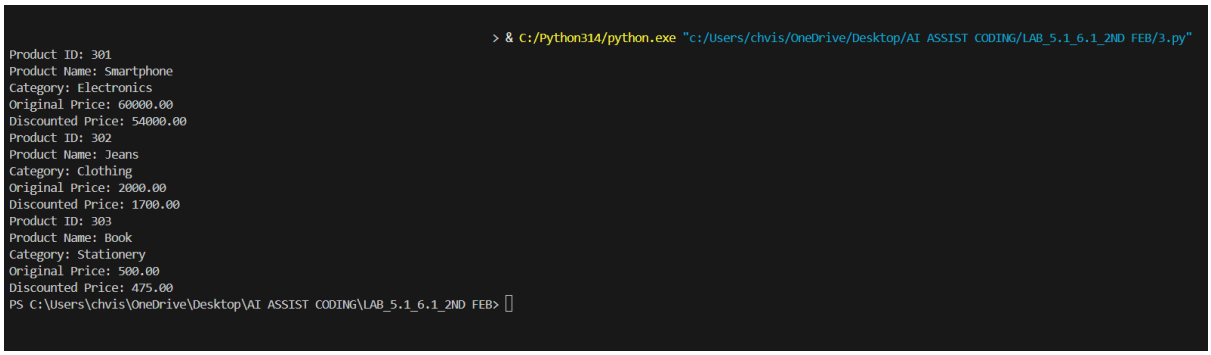
```
class Product:
    def __init__(self, product_id: int, product_name: str, price, category: str):
        self.product_id = product_id
        self.product_name = product_name
        try:
            self.price = float(price)
        except (TypeError, ValueError):
            self.price = 0.0
        self.category = category
    def calculate_discounted_price(self):
        cat = (self.category or "").lower()
        if cat == "electronics":
            discount = 0.10 * self.price
        elif cat == "clothing":
            discount = 0.15 * self.price
        else:
            discount = 0.05 * self.price
        return self.price - discount
    def display_product_details(self):
        discounted_price = self.calculate_discounted_price()
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Original Price: {self.price:.2f}")
        print(f"Discounted Price: {discounted_price:.2f}")
# Example usage:
product1 = Product(301, "Smartphone", 69000, "Electronics")
product1.display_product_details()
product2 = Product(202, "Jeans", 21000, "Clothing")
product2.display_product_details()
product3 = Product(103, "Book", 5000, "Stationery")
product3.display_product_details()
```

CODE:

```
class Product:
    def __init__(self, product_id: int, product_name: str, price, category: str):
        self.product_id = product_id
        self.product_name = product_name
        try:
            self.price = float(price)
        except (TypeError, ValueError):
            self.price = 0.0
        self.category = category
    def calculate_discounted_price(self):
        cat = (self.category or "").lower()
        if cat == "electronics":
            discount = 0.10 * self.price
        elif cat == "clothing":
            discount = 0.15 * self.price
        else:
            discount = 0.05 * self.price
```

```
        return self.price - discount
    def display_product_details(self):
        discounted_price = self.calculate_discounted_price()
        print(f"Product ID: {self.product_id}")
        print(f"Product Name: {self.product_name}")
        print(f"Category: {self.category}")
        print(f"Original Price: {self.price:.2f}")
        print(f"Discounted Price: {discounted_price:.2f}")
# Example usage:
product1 = Product(301, "Smartphone", 69000, "Electronics")
product1.display_product_details()
product2 = Product(202, "Jeans", 21000, "Clothing")
product2.display_product_details()
product3 = Product(103, "Book", 5000, "Stationery")
product3.display_product_details()
```

OUTPUT:



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.

SCREENSHOT OF THE CODE:

```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 calculate_fine(self):
        if self.days_late<=5:
            fine = self.days_late * 5
        elif self.days_late>5 and self.days_late<=10:
            fine = (5 * 5) + (self.days_late - 5) * 10
        else:
            fine = (5 * 5) + (5 * 10) + (self.days_late - 10) * 20
        return fine
    def display_book_details(self):
        fine = self.calculate_fine()
        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}")
        print(f"Total Fine: {fine}")
# Example usage:
book1 = LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald", "Alice", 12)
book1.display_book_details()
book2 = LibraryBook(402, "1984", "ganesh gupta", "Bob", 4)
book2.display_book_details()
```

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 calculate_fine(self):
        if self.days_late<=5:
            fine = self.days_late * 5
        elif self.days_late>5 and self.days_late<=10:
            fine = (5 * 5) + (self.days_late - 5) * 10
        else:
            fine = (5 * 5) + (5 * 10) + (self.days_late - 10) * 20
        return fine
    def display_book_details(self):
        fine = self.calculate_fine()
        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}")
        print(f"Total Fine: {fine}")
# Example usage:
book1 = LibraryBook(401, "The Great Gatsby", "F. Scott Fitzgerald", "Alice", 12)
book1.display_book_details()
book2 = LibraryBook(402, "1984", "ganesh gupta", "Bob", 4)
book2.display_book_details()
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\LAB_5.1_6.1_2ND FEB> & C:/Python314/python.exe "c:/Users/ch
Book ID: 401
Title: The Great Gatsby
Author: F. Scott Fitzgerald
Borrower: Alice
Days Late: 12
Total Fine: 115
Book ID: 402
Title: 1984
Author: ganesh gupta
Borrower: Bob
Days Late: 4
Total Fine: 20
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\LAB_5.1_6.1_2ND FEB>
```

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.

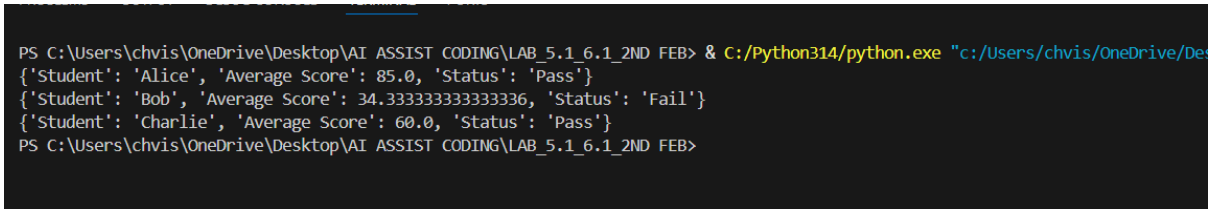
SCREENSHOT OF THE CODE:

```
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

students = {
    "Alice": [85, 78, 92],
    "Bob": [35, 40, 28],
    "Charlie": [50, 60, 70]
}
report = student_report(students)
for entry in report:
    print(entry)
```

CODE:

```
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


students = {
    "Alice": [85, 78, 92],
    "Bob": [35, 40, 28],
    "Charlie": [50, 60, 70]
}
report = student_report(students)
for entry in report:
    print(entry)
```

output:



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

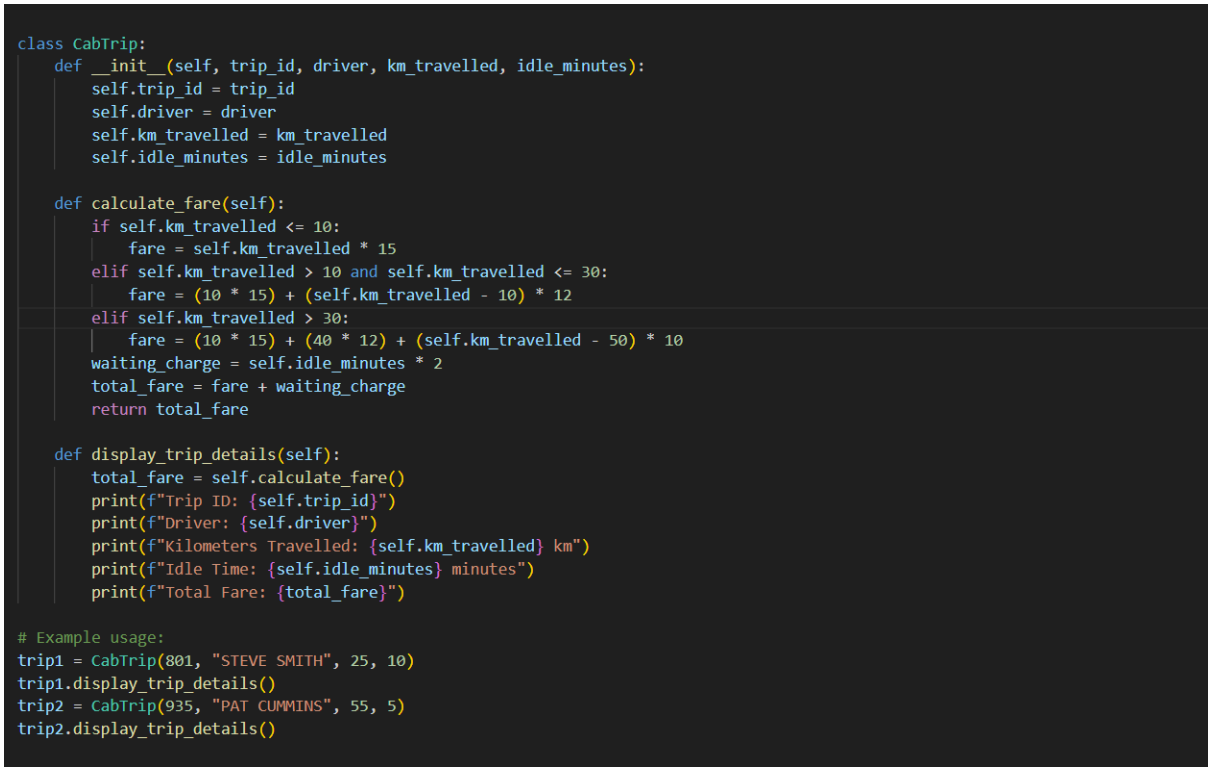
- ₹12 per km for the next 20 km

- ₹10 per km above 30 km

- Waiting charge: ₹2 per minute

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

SCREENSHOT OF THE CODE:



```python
class CabTrip:
    def __init__(self, trip_id, driver, km_travelled, idle_minutes):
        self.trip_id = trip_id
        self.driver = driver
        self.km_travelled = km_travelled
        self.idle_minutes = idle_minutes

    def calculate_fare(self):
        if self.km_travelled <= 10:
            fare = self.km_travelled * 15
        elif self.km_travelled > 10 and self.km_travelled <= 30:
            fare = (10 * 15) + (self.km_travelled - 10) * 12
        elif self.km_travelled > 30:
            fare = (10 * 15) + (40 * 12) + (self.km_travelled - 50) * 10
        waiting_charge = self.idle_minutes * 2
        total_fare = fare + waiting_charge
        return total_fare

    def display_trip_details(self):
        total_fare = self.calculate_fare()
        print(f"Trip ID: {self.trip_id}")
        print(f"Driver: {self.driver}")
        print(f"Kilometers Travelled: {self.km_travelled} km")
        print(f"Idle Time: {self.idle_minutes} minutes")
        print(f"Total Fare: {total_fare}")

# Example usage:
trip1 = CabTrip(801, "STEVE SMITH", 25, 10)
trip1.display_trip_details()
trip2 = CabTrip(935, "PAT CUMMINS", 55, 5)
trip2.display_trip_details()
```

CODE:

```
class CabTrip:
    def __init__(self, trip_id, driver, km_travelled, idle_minutes):
        self.trip_id = trip_id
        self.driver = driver
        self.km_travelled = km_travelled
        self.idle_minutes = idle_minutes

    def calculate_fare(self):
        if self.km_travelled <= 10:
            fare = self.km_travelled * 15
        elif self.km_travelled > 10 and self.km_travelled <= 30:
            fare = (10 * 15) + (self.km_travelled - 10) * 12
        elif self.km_travelled > 30:
```

```
            fare = (10 * 15) + (40 * 12) + (self.km_travelled - 50) * 10
        waiting_charge = self.idle_minutes * 2
        total_fare = fare + waiting_charge
        return total_fare

    def display_trip_details(self):
        total_fare = self.calculate_fare()
        print(f"Trip ID: {self.trip_id}")
        print(f"Driver: {self.driver}")
        print(f"Kilometers Travelled: {self.km_travelled} km")
        print(f"Idle Time: {self.idle_minutes} minutes")
        print(f"Total Fare: {total_fare}")

# Example usage:
trip1 = CabTrip(801, "STEVE SMITH", 25, 10)
trip1.display_trip_details()
trip2 = CabTrip(935, "PAT CUMMINS", 55, 5)
trip2.display_trip_details()
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\LAB_5.1_6.1_2ND FEB> & C:/Python314/python.exe "c:/Users/chv
Trip ID: 801
Driver: STEVE SMITH
Kilometers Travelled: 25 km
Idle Time: 10 minutes
Total Fare: 350
Trip ID: 935
Driver: PAT CUMMINS
Kilometers Travelled: 55 km
Idle Time: 5 minutes
Total Fare: 690
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING\LAB_5.1_6.1_2ND FEB>
```

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

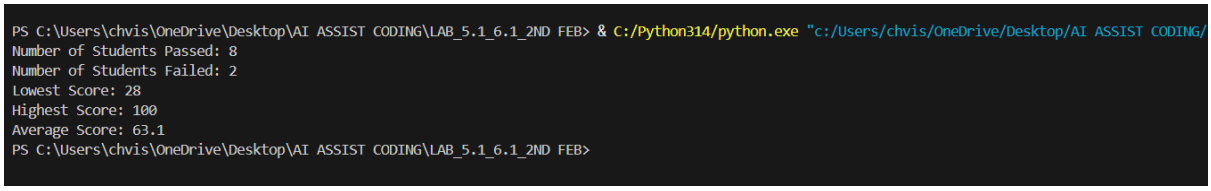
SCREENSHOT OF THE CODE:

```python
def statistics_subjects(scores):
    if not scores:
        return None, None, None
    min_score = min(scores)
    max_score = max(scores)
    avg_score = sum(scores) / len(scores)
    passed_count = len([scr for scr in scores if scr >= 40])
    failed_count = len([scr for scr in scores if scr < 40])
    print(f"Number of Students Passed: {passed_count}")
    print(f"Number of Students Failed: {failed_count}")
    return min_score, max_score, avg_score

# Example usage:
scores_list = [85, 42, 39, 76, 90, 55, 28, 67, 49, 100]
lowest, highest, average = statistics_subjects(scores_list)
print(f"Lowest Score: {lowest}")
print(f"Highest Score: {highest}")
print(f"Average Score: {average}")
```

Code:

```
def statistics_subjects(scores):
    if not scores:
        return None, None, None
    min_score = min(scores)
    max_score = max(scores)
    avg_score = sum(scores) / len(scores)
    passed_count = len([scr for scr in scores if scr >= 40])
    failed_count = len([scr for scr in scores if scr < 40])
    print(f"Number of Students Passed: {passed_count}")
    print(f"Number of Students Failed: {failed_count}")
    return min_score, max_score, avg_score

# Example usage:
scores_list = [85, 42, 39, 76, 90, 55, 28, 67, 49, 100]
lowest, highest, average = statistics_subjects(scores_list)
print(f"Lowest Score: {lowest}")
print(f"Highest Score: {highest}")
print(f"Average Score: {average}")
```

OUTPUT:



TASK 8:

Use AI to generate two solutions for checking prime numbers:

• Naive approach(basic)

• Optimised approach

Prompt:

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

Expected Output:

• Code for both methods.

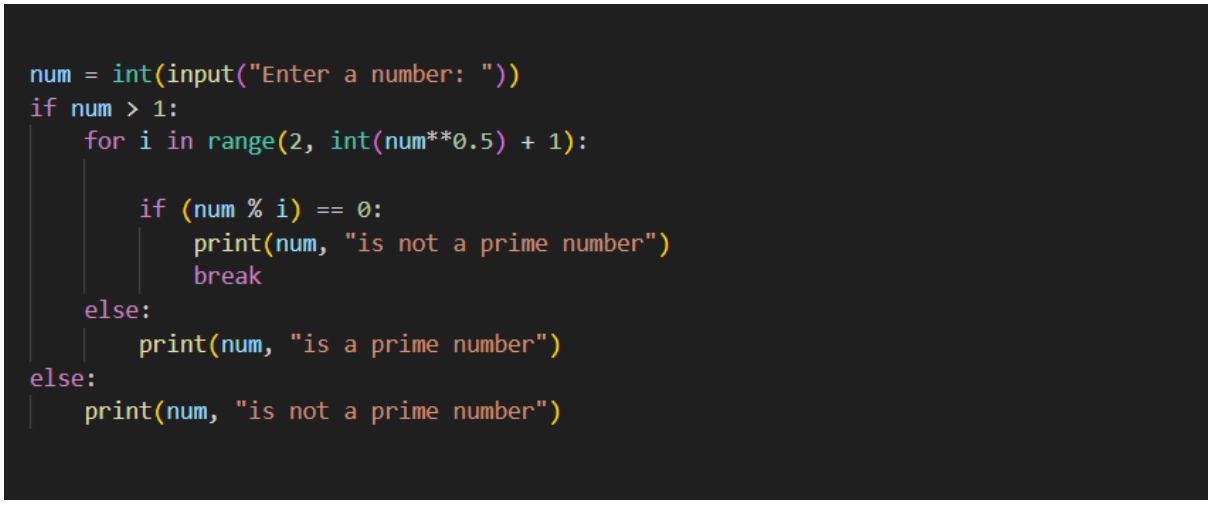
• Transparent explanation of time complexity.

• Comparison highlighting efficiency improvements.

PROMPT:

```
#generate a python program to print prime or not using basic method in an basiclevel of code without any function in native approach
```

SCREENSHOT OF THE CODE:



```
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, int(num**0.5) + 1):

        if (num % i) == 0:
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

CODE:

```
num = int(input("Enter a number: "))
if num > 1:
   for i in range(2, int(num**0.5) + 1):


      if (num % i) == 0:
         print(num, "is not a prime number")
         break
   else:
      print(num, "is a prime number")
else:
   print(num, "is not a prime number")
```

OPTIMIZED CODE:

prompt:

```
#write a python code to print weather the gievn number is prime or not in function method and in an optimized way
```

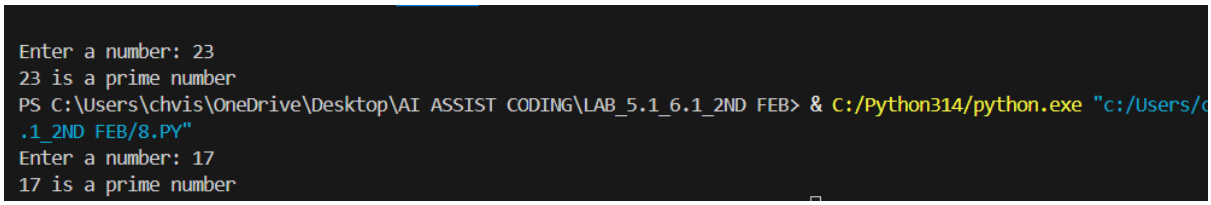
SCREENSHOT OF THE CODE:

```
def is_prime(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
number = int(input("Enter a number: "))
if is_prime(number):
    print(number, "is a prime number")
else:
    print(number, "is not a prime number")
```

CODE:

```
def is_prime(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
number = int(input("Enter a number: "))
if is_prime(number):
    print(number, "is a prime number")
else:
    print(number, "is not a prime number")
```

OUTPUT:



The first method checks for a prime number using inline logic without a function, directly printing the result during execution. The second method uses a reusable is_prime() function that returns a Boolean value. Although both methods check divisors only up to the square root, the function-based approach is cleaner, more modular, and easier to reuse, while the first method is less maintainable.

TASK 9:

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

Instructions:

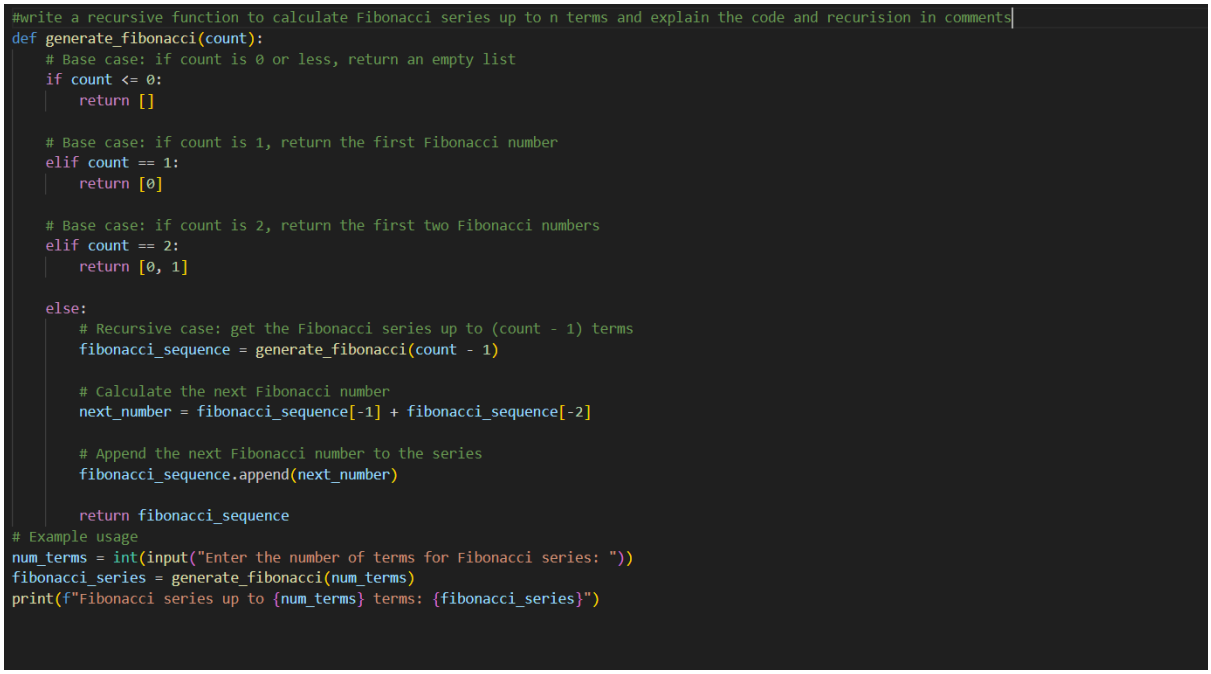
1. Ask AI to add clear comments explaining recursion.

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.

SCREENSHOT OF THE CODE:



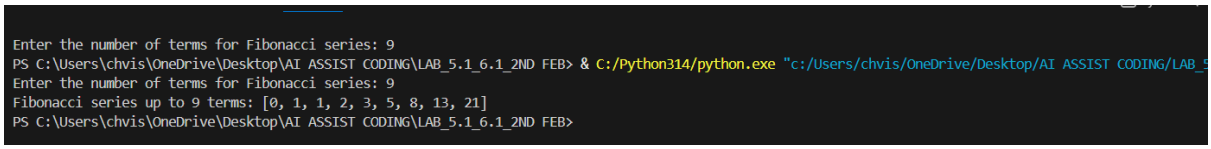
```
#write a recursive function to calculate Fibonacci series up to n terms and explain the code and recurision in comments
def generate_fibonacci(count):
    # Base case: if count is 0 or less, return an empty list
    if count <= 0:
        return []

    # Base case: if count is 1, return the first Fibonacci number
    elif count == 1:
        return [0]

    # Base case: if count is 2, return the first two Fibonacci numbers
    elif count == 2:
        return [0, 1]

    else:
        # Recursive case: get the Fibonacci series up to (count - 1) terms
        fibonacci_sequence = generate_fibonacci(count - 1)

        # Calculate the next Fibonacci number
        next_number = fibonacci_sequence[-1] + fibonacci_sequence[-2]

        # Append the next Fibonacci number to the series
        fibonacci_sequence.append(next_number)

        return fibonacci_sequence
# Example usage
num_terms = int(input("Enter the number of terms for Fibonacci series: "))
fibonacci_series = generate_fibonacci(num_terms)
print(f"Fibonacci series up to {num_terms} terms: {fibonacci_series}")
```

CODE:

```
def generate_fibonacci(count):
    # Base case: if count is 0 or less, return an empty list
    if count <= 0:
        return []

    # Base case: if count is 1, return the first Fibonacci number
    elif count == 1:
        return [0]

    # Base case: if count is 2, return the first two Fibonacci numbers
    elif count == 2:
        return [0, 1]

    else:
        # Recursive case: get the Fibonacci series up to (count - 1) terms
```

```
        fibonacci_sequence = generate_fibonacci(count - 1)

        # Calculate the next Fibonacci number
        next_number = fibonacci_sequence[-1] + fibonacci_sequence[-2]

        # Append the next Fibonacci number to the series
        fibonacci_sequence.append(next_number)

        return fibonacci_sequence
# Example usage
num_terms = int(input("Enter the number of terms for Fibonacci series: "))
fibonacci_series = generate_fibonacci(num_terms)
print(f"Fibonacci series up to {num_terms} terms: {fibonacci_series}")
```

OUTPUT:



TASK 10:

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.

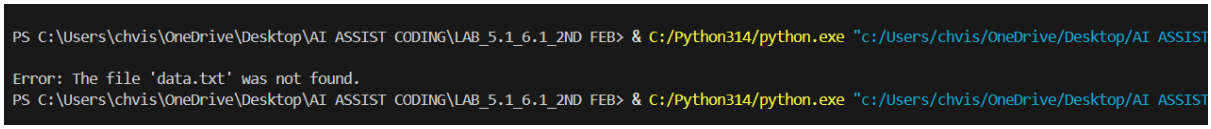
• Validation that explanations align with runtime behavior.

SCREENSHOT OF THE CODE:

```python
#write a python program that reads a text file and processes the data
# Generate code with proper error handling and clear explanations for each
# exception

try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()

            # Process the data (for example, count the number of lines)
            lines = data.splitlines()
            print(f"The file contains {len(lines)} lines.")

        except Exception as e:
            # Handle exceptions during data processing
            print(f"An error occurred while processing the data: {e}")
```

```python
except FileNotFoundError:
    # Handle the case where the file does not exist
    print("Error: The file 'data.txt' was not found.")

except IOError:
    # Handle other I/O errors
    print("Error: An I/O error occurred while trying to read the file.")

except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
```

Output:



CODE:

```
try:
    # Attempt to open the file in read mode
    with open('data.txt', 'r') as file:
        try:
            # Read the contents of the file
            data = file.read()

            # Process the data (for example, count the number of lines)
            lines = data.splitlines()
            print(f"The file contains {len(lines)} lines.")

        except Exception as e:
            # Handle exceptions during data processing
            print(f"An error occurred while processing the data: {e}")

except FileNotFoundError:
    # Handle the case where the file does not exist
    print("Error: The file 'data.txt' was not found.")

except IOError:
    # Handle other I/O errors
    print("Error: An I/O error occurred while trying to read the file.")

except Exception as e:
    # Handle any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
```