ASSIGNMENT-6.3

M.PARAMESH

HN0: 2303A53021

BATCH – 46

Task Description #1: Classes (Student Class)

Scenario

You are developing a simple student information management module.

Task

• Use an AI tool (GitHub Copilot / Cursor AI / Gemini) to complete a Student class.

• The class should include attributes such as name, roll number, and branch.

• Add a method display_details() to print student information.

• Execute the code and verify the output.

• Analyze the code generated by the AI tool for correctness and clarity.

Expected Output #1

• A Python class with a constructor (__init__) and a display_details() method.

• Sample object creation and output displayed on the console.

• Brief analysis of AI-generated code.

Prompt used:

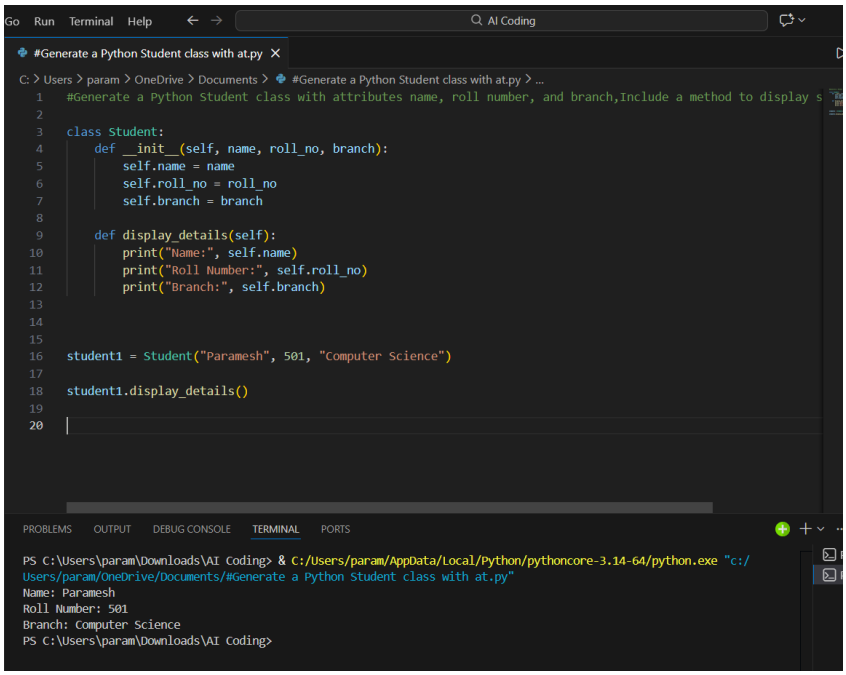
Generate a Python Student class with attributes name, roll number, and branch.
Include a method to display student details.
Create a sample object and print the output

Code :

```
class Student:
    def __init__(self, name, roll_no, branch):
```

```
        self.name = name
        self.roll_no = roll_no
        self.branch = branch
    def display_details(self):
        print("Name:", self.name)
        print("Roll Number:", self.roll_no)
        print("Branch:", self.branch)
student1 = Student("Paramesh", 101, "Computer Science")
student1.display_details()
```

output :



Explaination :

1) The AI correctly generated a Python class using a constructor (__init__) to initialize student details.

2) The display_details() method cleanly prints all attributes in a readable format.

Task Description #2: Loops (Multiples of a Number)

Scenario

You are writing a utility function to display multiples of a given number.

Task

• Prompt the AI tool to generate a function that prints the first 10 multiples of a given number

using a loop.

• Analyze the generated loop logic.

• Ask the AI to generate the same functionality using another controlled looping structure (e.g.,

while instead of for).

Expected Output #2

• Correct loop-based Python implementation.

• Output showing the first 10 multiples of a number.

• Comparison and analysis of different looping approaches.

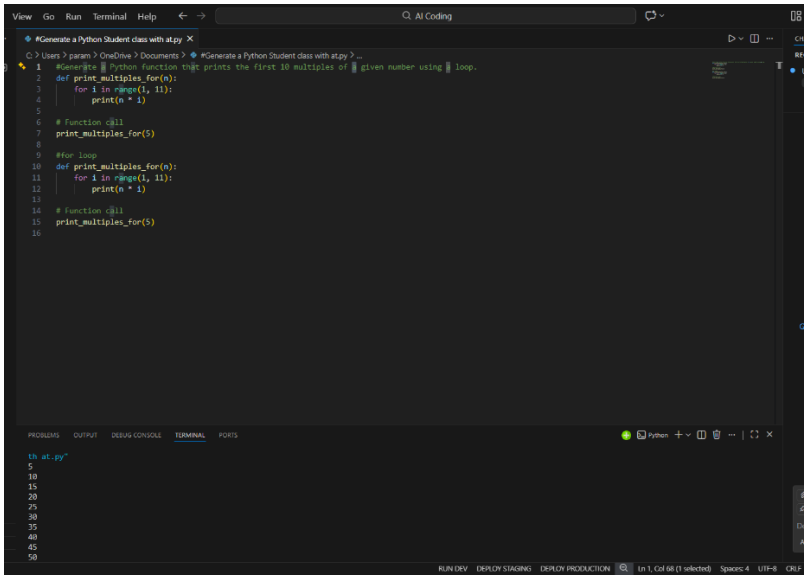
Prompt used:

Generate a Python function that prints the first 10 multiples of a given number using a loop.
Then generate the same functionality using a while loop instead of a for loop.

Code :

```
def print_multiples_for(n):
    for i in range(1, 11):
        print(n * i)
# Function call
print_multiples_for(5)
```

output :



Explaination :

1)The for loop version is more concise and easier to read when the number of iterations is fixed.

2)The while loop provides more control over the loop variable but requires manual incrementing.

3)Both implementations are correct and produce identical output, demonstrating different looping approaches.

Task Description #3: Conditional Statements (Age Classification)

Scenario

You are building a basic classification system based on age.

Task

• Ask the AI tool to generate nested if-elif-else conditional statements to classify age groups(e.g., child, teenager, adult, senior).

• Analyze the generated conditions and logic.

• Ask the AI to generate the same classification using alternative conditional structures (e.g.,simplified conditions or dictionary-based logic).

Expected Output #3

• A Python function that classifies age into appropriate groups.

• Clear and correct conditional logic.

• Explanation of how the conditions work.

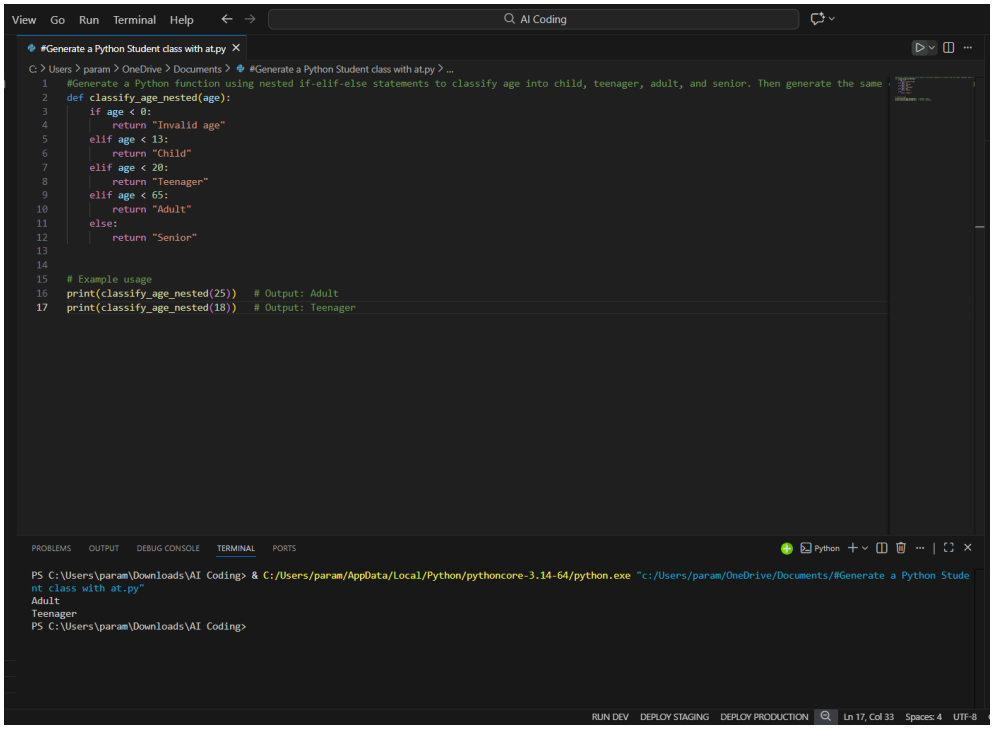
Prompt used:

Generate a Python function using nested if-elif-else statements to classify age into child, teenager, adult, and senior.
Then generate the same classification using a simplified or alternative conditional structure

Code :

```
def classify_age_nested(age):
    if age < 0:
        return "Invalid age"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"
# Example usage
print(classify_age_nested(25))   # Output: Adult
print(classify_age_nested(18))   # Output: Teenager
```

output :



Expalaination :

1) The if-elif-else structure checks age ranges step by step and assigns the correct category.

2)The simplified approach uses a list of limits, making the logic more compact and easier to modify.

Task Description #4: For and While Loops (Sum of First n Numbers)

Scenario

You need to calculate the sum of the first n natural numbers.

Task

• Use AI assistance to generate a sum_to_n() function using a for loop.

• Analyze the generated code.

• Ask the AI to suggest an alternative implementation using a while loop or a mathematical

formula.

Expected Output #4

• Python function to compute the sum of first n numbers.

• Correct output for sample inputs.

• Explanation and comparison of different approaches

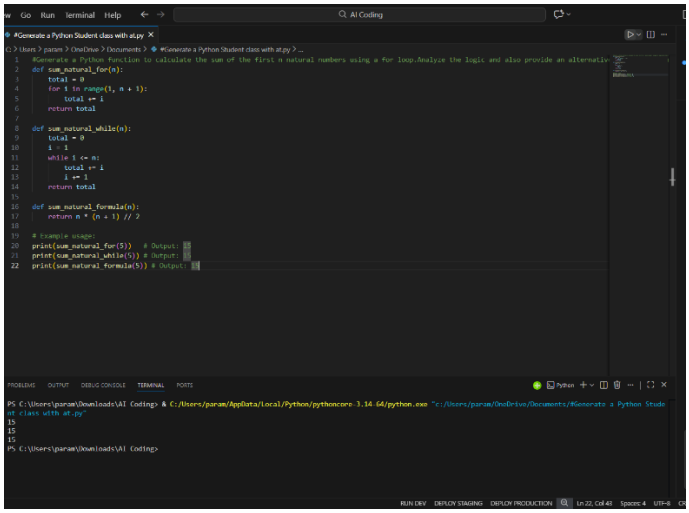
Prompt used :

Generate a Python function to calculate the sum of the first n natural numbers using a for loop.
Analyze the logic and also provide an alternative implementation using a while loop or a mathematical formula.

Code :

```
def sum_natural_for(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
def sum_natural_while(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
def sum_natural_formula(n):
    return n * (n + 1) // 2
# Example usage:
print(sum_natural_for(5))   # Output: 15
print(sum_natural_while(5)) # Output: 15
print(sum_natural_formula(5)) # Output: 15
```

output :



Explaination :

1) The for loop approach adds numbers sequentially and is easy to understand for beginners.

2)The while loop gives manual control over iteration but requires careful incrementing.

3)The mathematical formula is the most efficient method with constant time complexity

Task Description #5: Classes (Bank Account Class)

Scenario

You are designing a basic banking application.

Task

• Use AI tools to generate a Bank Account class with methods such as deposit(), withdraw(),and check_balance().

• Analyze the AI-generated class structure and logic.

• Add meaningful comments and explain the working of the code.

Expected Output #5

• Complete Python Bank Account class.

• Demonstration of deposit and withdrawal operations with updated balance.

• Well-commented code with a clear explanation.

Prompt used:

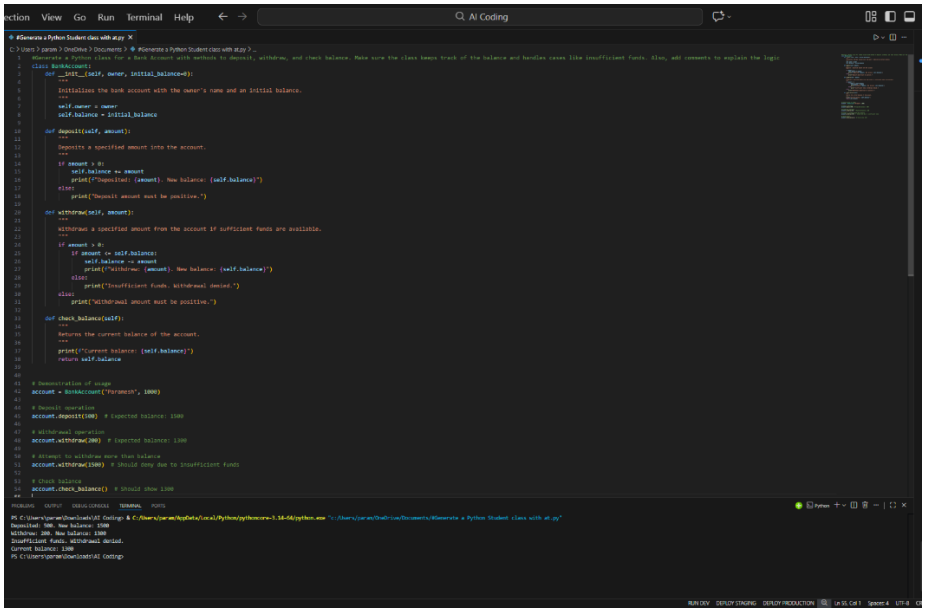
Generate a Python class for a Bank Account with methods to deposit, withdraw, and check balance. Make sure the class keeps track of the balance and handles cases like insufficient funds. Also, add comments to explain the logic

Code :

```
class BankAccount:
  def __init__(self, owner, initial_balance=0):
    """
    Initializes the bank account with the owner's name and an initial balance.
    """
    self.owner = owner
    self.balance = initial_balance

  def deposit(self, amount):
    """
    Deposits a specified amount into the account.
    """
    if amount > 0:
      self.balance += amount
      print(f"Deposited: {amount}. New balance: {self.balance}")
    else:
      print("Deposit amount must be positive.")

  def withdraw(self, amount):
    """
    Withdraws a specified amount from the account if sufficient funds are available.
    """
    if amount > 0:
      if amount <= self.balance:
        self.balance -= amount
```

```
            print(f"Withdrew: {amount}. New balance: {self.balance}")
        else:
            print("Insufficient funds. Withdrawal denied.")
    else:
        print("Withdrawal amount must be positive.")

  def check_balance(self):
    """
    Returns the current balance of the account.
    """
    print(f"Current balance: {self.balance}")
    return self.balance
# Demonstration of usage
account = BankAccount("Paramesh", 1000)
# Deposit operation
account.deposit(500)  # Expected balance: 1500
# Withdrawal operation
account.withdraw(200)  # Expected balance: 1300
# Attempt to withdraw more than balance
account.withdraw(1500)  # Should deny due to insufficient funds
# Check balance
account.check_balance()  # Should show 1300
```

output:



Explaination:

1)The BankAccount class initializes with an owner name and an initial balance, ensuring each account is uniquely identified.

2)The deposit() method checks for positive amounts before adding them to the balance, ensuring basic validation.

3)The withdraw() method ensures that the withdrawal amount is positive and that the account has sufficient funds before completing the transaction. If not, it prevents the withdrawal.

4)The check_balance() method simply returns and prints the current balance, giving a clear snapshot of account funds.