AI Assisted Coding

A.Shashank || Batch-09 || 2303A51639

Task Description #1: Classes (Student Class)

Prompt: Develop a student class and constructor with a display_details () method and class should have attributes name, roll number, and branch.

Code:

```
class Student:
    def __init__(self, name, roll_number, branch):
        self.name = name
        self.roll_number = roll_number
        self.branch = branch

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Branch: {self.branch}")


# Example usage
student1 = Student("John Doe", 101, "Computer Science")
student1.display_details()
```

Output:

```
Name: John Doe
Roll Number: 101
Branch: Computer Science
```

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

Prompt: generate a function that prints the first 10 multiples of a given number using a loop.

generate a function that prints the first 10 multiples of a given number using a while loop.

Code:

```
#generate a function that prints the first 10 multiples of a
given number using a loop.
def print_multiples(number):
    print(f"First 10 multiples of {number}:")
    for i in range(1, 11):
        print(number * i, end=" ")
    print()

# Example usage
user_input = int(input("Enter a number to find its multiples:
"))
print_multiples(user_input)

#generate a function that prints the first 10 multiples of a
given number using a while loop.
def print_multiples_while(number):
    print(f"First 10 multiples of {number}:")
    i = 1
    while i <= 10:
        print(number * i, end=" ")
        i += 1
    print()

# Example usage
user_input2 = int(input("Enter a number to find its multiples
(using while loop): "))
print_multiples_while(user_input2)
```

Output:

```
Enter a number to find its multiples: 9
First 10 multiples of 9:
9 18 27 36 45 54 63 72 81 90
Enter a number to find its multiples (using while loop): 6
First 10 multiples of 6:
6 12 18 24 30 36 42 48 54 60
```

Task Description #3: Conditional Statements (Age Classification)

Prompt: generate nested if-elif-else conditional statements to classify age groups

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

Code:

```
#generate nested if-elif-else conditional statements to
classify age groups
def classify_age(age):
    if age < 0:
        print("Invalid age: Age cannot be negative")
    elif age < 13:
        if age < 5:
            print(f"Age {age}: Toddler")
        else:
            print(f"Age {age}: Child")
    elif age < 20:
        if age < 18:
            print(f"Age {age}: Pre-teen/Early Teen")
        else:
            print(f"Age {age}: Late Teen")
    elif age < 60:
        if age < 40:
            print(f"Age {age}: Young Adult")
        else:
            print(f"Age {age}: Middle-aged Adult")
    else:
        print(f"Age {age}: Senior Citizen")

# Example usage
user_age = int(input("Enter your age for classification: "))
classify_age(user_age)

#generate the same classification using alternative
conditional structures (e.g.,simplified conditions or
dictionary-based logic).
# Alternative age classification using dictionary-based logic
def classify_age_dict(age):
    if age < 0:
        print("Invalid age: Age cannot be negative")
        return
    age_groups = [
        (lambda a: a < 5, "Toddler"),
        (lambda a: a < 13, "Child"),
```

```
        (lambda a: a < 18, "Pre-teen/Early Teen"),
        (lambda a: a < 20, "Late Teen"),
        (lambda a: a < 40, "Young Adult"),
        (lambda a: a < 60, "Middle-aged Adult"),
        (lambda a: True, "Senior Citizen")
    ]
    for cond, label in age_groups:
        if cond(age):
            print(f"Age {age}: {label}")
            break

# Example usage
user_age2 = int(input("Enter your age for classification
(alternative): "))
classify_age_dict(user_age2)
```

Output:

```
Enter your age for classification: 25
Age 25: Young Adult
Enter your age for classification (alternative): 64
Age 64: Senior Citizen
```

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

Prompt: Function to calculate sum of first n numbers using a for loop

Function to calculate sum of first n numbers using a while loop

Code:

```
# Function to calculate sum of first n numbers using a for
loop
def sum_to_n(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

# Example usage
user_n = int(input("Enter n to find sum of first n numbers
(using for loop): "))
print(f"Sum of first {user_n} numbers: {sum_to_n(user_n)}")

#Function to calculate sum of first n numbers using a while
loop
def sum_to_n_while(n):
    total = 0
```

```
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

# Example usage
user_n2 = int(input("Enter n to find sum of first n numbers
(using while loop): "))
print(f"Sum of first {user_n2} numbers:
{sum_to_n_while(user_n2)}")
```

Output:

```
Enter n to find sum of first n numbers (using for loop): 9
Sum of first 9 numbers: 45
Enter n to find sum of first n numbers (using while loop): 6
Sum of first 6 numbers: 21
```

Task Description-5: Classes (Bank Account Class)

Prompt: develop a Bank Account class with methods for deposit(), withdraw(), and check_balance()

Code:

```
#develop a Bank Account class with methods for deposit(),
withdraw(), and check_balance()
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount <= 0:
            print("Deposit amount must be positive")
            return False
        self.balance += amount
        print(f"Deposited: ${amount}. New balance:
${self.balance}")
        return True

    def withdraw(self, amount):
        if amount <= 0:
```

```
            print("Withdrawal amount must be positive")
            return False
        if amount > self.balance:
            print(f"Insufficient funds. Current balance:
${self.balance}")
            return False
        self.balance -= amount
        print(f"Withdrawn: ${amount}. New balance:
${self.balance}")
        return True

    def check_balance(self):
        print(f"Account holder: {self.account_holder}")
        print(f"Current balance: ${self.balance}")
        return self.balance

# Example usage
account = BankAccount("Jane Smith", 1000)
account.check_balance()
account.deposit(500)
account.withdraw(200)
account.check_balance()
account.withdraw(2000)
```

Output:

```
Account holder: Jane Smith
Current balance: $1000
Deposited: $500. New balance: $1500
Withdrawn: $200. New balance: $1300
Account holder: Jane Smith
Current balance: $1300
Insufficient funds. Current balance: $1300
```