LAB ASSIGNMENT-6.3

Hall Ticket no : *2303A51430*

Name : *MADASI SATHWIK*

Task 1: Classes (Student Class)

Scenario:

You are developing a simple student information management module.

Prompt Used:

#Developing a simple Students information management module.

#using of classes we need to complete the students class.

#in the class we need to inculde attributes like name, age, grade and methods to display student information.

#Also, we need to create a method to update the student's grade.

#Also, we need to create a method to check if the student has passed or failed based on the grade.

#should Also check whether the his attdendance is above 75% to write the exam.

#System Should be able to create multiple student objects and manage their information.

CODE:

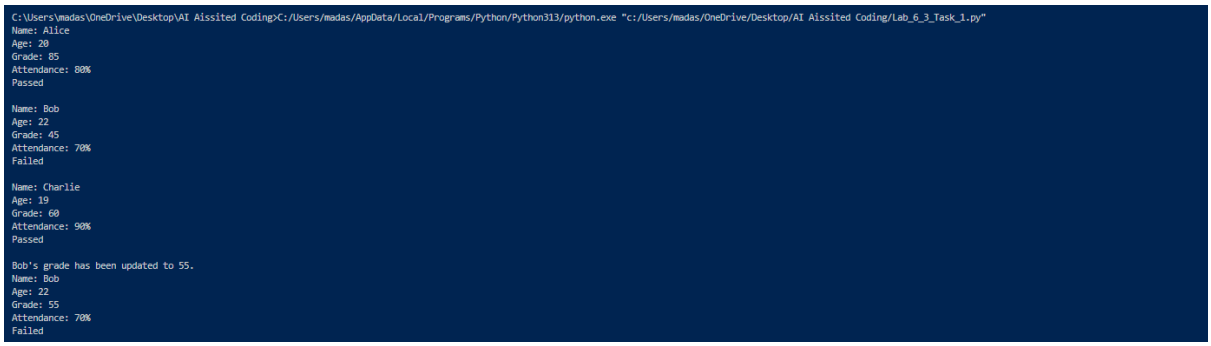
```
class Student:
    def __init__(self, name, age, grade, attendance):
        self.name = name
        self.age = age
        self.grade = grade
        self.attendance = attendance

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")
        print(f"Attendance: {self.attendance}%")

    def update_grade(self, new_grade):
        self.grade = new_grade
        print(f"{self.name}'s grade has been updated to {self.grade}.")

    def has_passed(self):
        if self.grade >= 50 and self.attendance >= 75:
            return True
        else:
            return False
# Creating multiple student objects
student1 = Student("Alice", 20, 85, 80)
student2 = Student("Bob", 22, 45, 70)
student3 = Student("Charlie", 19, 60, 90)
# Displaying student information
student1.display_info()
print("Passed" if student1.has_passed() else "Failed")
print()
student2.display_info()
print("Passed" if student2.has_passed() else "Failed")
print()
student3.display_info()
print("Passed" if student3.has_passed() else "Failed")
print()
# Updating a student's grade
student2.update_grade(55)
student2.display_info()
print("Passed" if student2.has_passed() else "Failed")
```

Output:



Explanation:

• The Student class acts like a template to store student information such as name, age, grade, and attendance. It helps create many student records using one structure.

• The constructor automatically saves the values given when a student object is created. It stores the student's details inside the object.

• The display information method is used to show the stored student details when it is called. It prints name, age, grade, and attendance.

• The update grade method is used to change the student's marks later if needed. It replaces the old grade with the new grade.

• The pass/fail checking method verifies whether the student passed or not. It checks both marks and attendance conditions.

• The student is considered passed only if the grade is 50 or more and attendance is at least 75%. Otherwise, the student is marked as failed.

• The attendance rule is used to decide whether the student is eligible to write the exam. Less than 75% attendance means not eligible.

• Multiple student objects can be created using the same class. Each object stores separate student data.

• The system displays each student's details and prints the result based on the checking method.

Task 2 : Loops (Multiples of a Number).

Scenario:

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

Prompt Used:

#I am writing a utility function to display mutliples of a given number within a specified range using Loops.

#The function should generate and print the 1st 20 multiples of the given number using a loop structure.

#Also need to generate the same functionality using another controlled looping structure.(like while loop instead of for loop or for loop insted of while loop).

#The inputs should be taken from the user for the number whose multiples are to be generated.

#example format to write 12 x 1 = 12.

CODE:

```python
#I am writing a utility function to display mutliples of a given number within a specified range using Loops.
#The function should generate and print the 1st 20 multiples of the given number using a loop structure.
#Also need to generate the same functionality using another controlled looping structure.(like while loop instead of for loop or for loop insted of while loop).
#The inputs should be taken from the user for the number whose multiples are to be generated.
#example  format to write 12 x 1 = 12.
def display_multiples_for_loop(number):
    print(f"Multiples of {number} using for loop:")
    for i in range(1, 21):
        print(f"{number} x {i} = {number * i}")
def display_multiples_while_loop(number):
    print(f"Multiples of {number} using while loop:")
    i = 1
    while i <= 20:
        print(f"{number} x {i} = {number * i}")
        i += 1
# Taking input from the user
num = int(input("Enter a number to display its multiples: "))
# Displaying multiples using for loop
display_multiples_for_loop(num)
print()  # Adding a blank line for better readability
# Displaying multiples using while loop
display_multiples_while_loop(num)
```

Output:

```
C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>C:/Users/madas/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/Lab_6_3_Task_2.py"
Enter a number to display its multiples: 12
Multiples of 12 using for loop:
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
12 x 11 = 132
12 x 12 = 144
12 x 13 = 156
12 x 14 = 168
12 x 15 = 180
12 x 16 = 192
12 x 17 = 204
12 x 18 = 216
12 x 19 = 228
12 x 20 = 240

Multiples of 12 using while loop:
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
12 x 7 = 84
12 x 8 = 96
12 x 9 = 108
12 x 10 = 120
12 x 11 = 132
12 x 12 = 144
12 x 13 = 156
12 x 14 = 168
12 x 15 = 180
12 x 16 = 192
12 x 17 = 204
12 x 18 = 216
12 x 19 = 228
12 x 20 = 240
```

Explanation:

• The program is designed to display the first 20 multiples of a given number. It prints them in multiplication table format.

• The input number is taken from the user. This allows the user to choose which number's multiples to generate.

• One function is created to display multiples using a for loop. It repeats the calculation from 1 to 20 automatically.

• In the for loop method, the loop counter increases step by step and prints each multiple in the format like "number × value = result".

• Another function is created to display multiples using a while loop. This shows the same logic using a different loop control method.

• In the while loop method, counting starts from 1 and continues until it reaches 20 using a condition check.

• After each iteration in the while loop, the counter is increased manually so the loop moves forward.

• Both functions produce the same output, but they use different looping structures. This helps understand the difference between for loop and while loop.

• A blank line is printed between outputs to make the display clean and easy to read.

• Overall, the program demonstrates how to generate multiplication tables using two types of loops and user input.

Task 3: Few-Shot Prompting – Name Formatting

Scenario:

Few-shot prompting improves accuracy by providing multiple examples.

Prompt Used:

#Basic Classification system based on the age using conditional Statements.

#So we need to generate nested if-elif-else condtional statements to classify the age groups.

#The classification groups are as follows:

#Child: 0-12 years

#Teenager: 13-19 years

#Adult: 20-64 years

#Senior: 65 years and above

#Also Generate the same classification using alternative conditional structures like simplified conditions or dictonary-based classification.

#Taking input from the user for age.

#Displaying the classification based on the age provided by the user.

CODE:

```python
#Basic Classification system based on the age using conditional Statements.
#So we need to generate nested if-elif-else condtional statements to classify the age groups.
#The classification groups are as follows:
#Child: 0-12 years
#Teenager: 13-19 years
#Adult: 20-64 years
#Senior: 65 years and above
#Also Generate the same classification using alternative  conditional  structures like simplified conditions or dictonary-based classification.
#Taking input from the user for age.
#Displaying the classification based on the age provided by the user.
def classify_age_nested(age):
    if age >= 0:
        if age <= 12:
            return "Child"
        elif age <= 19:
            return "Teenager"
        elif age <= 64:
            return "Adult"
        else:
            return "Senior"
    else:
        return "Invalid age"
def classify_age_simplified(age):
    if age < 0:
        return "Invalid age"
    elif age <= 12:
        return "Child"
    elif age <= 19:
        return "Teenager"
    elif age <= 64:
        return "Adult"
    else:
        return "Senior"
def classify_age_dictionary(age):
    if age < 0:
        return "Invalid age"
    age_groups = {
        range(0, 13): "Child",
        range(13, 20): "Teenager",
        range(20, 65): "Adult",
        range(65, 150): "Senior"
    }
    for age_range, classification in age_groups.items():
        if age in age_range:
            return classification
    return "Invalid age"
# Taking input from the user
user_age = int(input("Enter your age: "))
# Displaying classification using nested if-elif-else
print("Classification (Nested):", classify_age_nested(user_age))
# Displaying classification using simplified conditions
print("Classification (Simplified):", classify_age_simplified(user_age))
# Displaying classification using dictionary-based classification
print("Classification (Dictionary):", classify_age_dictionary(user_age))
```

Output:

```
Active code page: 65001

C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>C:/Users/madas/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/Lab_6_3_Task_3.py"
Enter your age: 18
Classification (Nested): Teenager
Classification (Simplified): Teenager
Classification (Dictionary): Teenager

C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>
```

Explanation:

• The program classifies a person into an age group based on the age entered by the user. It displays the result using different conditional approaches.
• The input age is taken from the user and converted into an integer so numerical comparisons can be performed.
• One function is created using nested if-elif-else statements. It first checks whether the age is valid (non-negative) and then classifies it step by step.

• In the nested method, conditions are checked inside another condition, which creates a layered validation structure.
• Another function is created using simplified if-elif-else statements. It checks conditions in a straight sequence without nesting, making the logic shorter and more readable.
• In the simplified method, each age limit is checked in increasing order, and once a condition is satisfied, the classification is returned.
• A third function is created using a dictionary with range values as keys and age group labels as values. This provides an alternative structured mapping approach.
• In the dictionary method, the program loops through each range and checks if the entered age belongs to that range, then returns the matching classification.
• Negative ages are handled separately in all methods and return "Invalid age" to ensure proper input validation.
• Overall, the program demonstrates how the same classification task can be solved using nested conditions, simplified conditions, and dictionary-based logic.

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

Scenario:

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

Prompt Used:

#Developing a simple sum calculation module.

#Using loops we need to compute the sum of first n natural numbers.

#The system should take input n from the user.

#We need to create a function sum_to_n() using a for loop.

#The function should iterate from 1 to n and add all numbers.

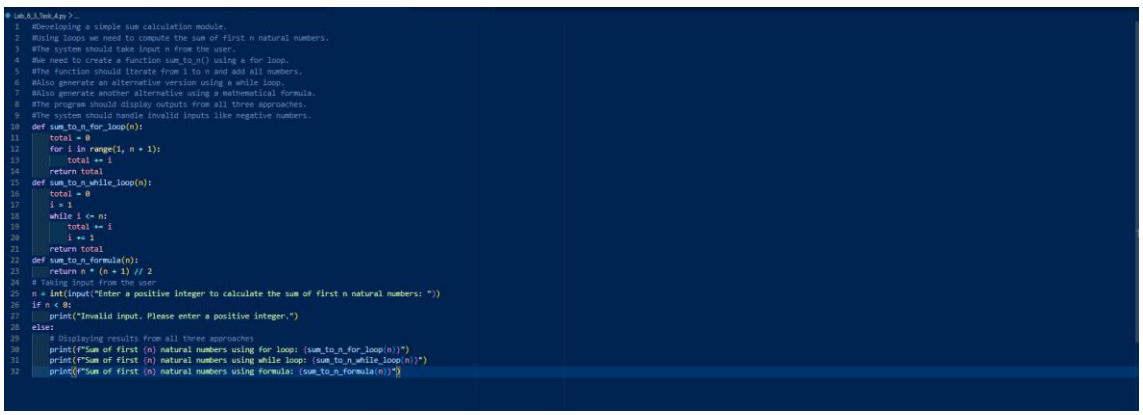
#Also generate an alternative version using a while loop.

#Also generate another alternative using a mathematical formula.

#The program should display outputs from all three approaches.

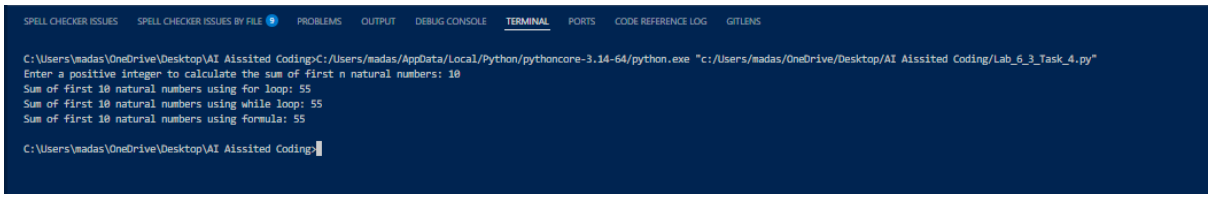
#The system should handle invalid inputs like negative numbers.

CODE:



```
#Developing a simple sum calculation module.
#Using loops we need to compute the sum of first n natural numbers.
#The system should take input n from the user.
#We need to create a function sum_to_n() using a for loop.
#The function should iterate from 1 to n and add all numbers.
#Also generate an alternative version using a while loop.
#Also generate another alternative using a mathematical formula.
#The program should display outputs from all three approaches.
#The system should handle invalid inputs like negative numbers.
def sum_to_n_for_loop(n):
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
def sum_to_n_while_loop(n):
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total
def sum_to_n_formula(n):
    return n * (n + 1) // 2
# Taking input from the user
n = int(input("Enter a positive integer to calculate the sum of first n natural numbers: "))
if n < 0:
    print("Invalid input. Please enter a positive integer.")
else:
    # Displaying results from all three approaches
    print(f"Sum of first {n} natural numbers using for loop: {sum_to_n_for_loop(n)}")
    print(f"Sum of first {n} natural numbers using while loop: {sum_to_n_while_loop(n)}")
    print(f"Sum of first {n} natural numbers using formula: {sum_to_n_formula(n)}")
```

Output:



Explanation:

• The program is designed to calculate the sum of the first n natural numbers based on the value entered by the user.
• The input n is taken from the user and validated to ensure it is not negative before performing calculations.
• One function computes the sum using a for loop, where the loop automatically runs from 1 to n and keeps adding each number to a total variable.
• In the for loop method, iteration control is automatic, which makes the code shorter and easier to read.
• Another function computes the sum using a while loop, where counting starts at 1 and continues until the condition reaches n.
• In the while loop method, the counter variable is increased manually after each iteration, giving more control over the loop flow.
• A third method uses a mathematical formula n×(n+1)/2 to compute the sum instantly without looping, which is the most efficient approach.
• All three approaches produce the same result for valid inputs, but they differ in performance and loop control style.

Task 5: Classes (Bank Account Class)

Scenario:

You are designing a basic banking application.

Prompt Used:

#Developing a basic banking application module.

#Using classes we need to build a BankAccount class.

#In the class we need to include attributes like account_holder and balance.

#We need to create methods deposit(), withdraw(), and check_balance().

#Deposit method should add money to the account balance.

#Withdraw method should subtract money only if sufficient balance is available.

#Check_balance method should display the current balance.

#System should prevent invalid operations like negative deposit or over-withdrawal.

#System should be able to create multiple bank account objects.

#Program should demonstrate deposit and withdrawal with updated balance.

CODE:

```python
#Developing a basic banking application module.
#Using classes we need to build a BankAccount class.
#In the class we need to include attributes like account_holder and balance.
#We need to create methods deposit(), withdraw(), and check_balance().
#Deposit method should add money to the account balance.
#Withdraw method should subtract money only if sufficient balance is available.
#Check_balance method should display the current balance.
#System should prevent invalid operations like negative deposit or over-withdrawal.
#System should be able to create multiple bank account objects.
#Program should demonstrate deposit and withdrawal with updated balance.
class BankAccount:
    def __init__(self, account_holder, balance=0):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount <= 0:
            print("Invalid deposit amount. Please enter a positive value.")
            return
        self.balance += amount
        print(f"{amount} deposited successfully. Current balance: {self.balance}")

    def withdraw(self, amount):
        if amount <= 0:
            print("Invalid withdrawal amount. Please enter a positive value.")
            return
        if amount > self.balance:
            print("Insufficient balance. Withdrawal failed.")
            return
        self.balance -= amount
        print(f"{amount} withdrawn successfully. Current balance: {self.balance}")

    def check_balance(self):
        print(f"Current balance for {self.account_holder}: {self.balance}")
# Creating multiple bank account objects
account1 = BankAccount("Alice", 1000)
account2 = BankAccount("Bob", 500)
# Demonstrating deposit and withdrawal with updated balance
account1.check_balance()
account1.deposit(200)
account1.withdraw(150)
account1.check_balance()
print()
account2.check_balance()
account2.deposit(300)
account2.withdraw(100)
account2.check_balance()
```

Output:

```
C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>C:/Users/madas/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/Lab_6_3_Task_5.py"
Current balance for Alice: 1000
200 deposited successfully. Current balance: 1200
150 withdrawn successfully. Current balance: 1050
Current balance for Alice: 1050

Current balance for Bob: 500
300 deposited successfully. Current balance: 800
100 withdrawn successfully. Current balance: 700
Current balance for Bob: 700

C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>
```

Explanation:

• The program is designed to simulate a basic banking system using a class-based structure.
• A BankAccount class is created with attributes such as account_holder name and balance to store account data.
• The constructor method initializes the account holder details and starting balance when an object is created.
• A deposit method is included to add money to the balance after checking that the deposit amount is valid.
• A withdraw method is included to subtract money only if the account has sufficient

balance and the amount is valid.
• A check_balance method is provided to display the current account balance at any time.
• The class prevents invalid operations like negative deposits and withdrawals greater than the available balance.
• Multiple account objects can be created and managed independently, demonstrating practical use of classes and methods.