ASSIGNMENT-6.3

NAME:N.VAISHNAVI REDDY

ROLL NUMBER:2303A51254

BATCH:18

Task Descrip9on #1: Classes (Student Class)

PROMPT:

Create a Python class named Student.

Add aMributes: name, roll_number, branch.

Use __init__ to iniRalize aMributes.

Add display_details() to print student info.

Create one Student object.

Call the display_details() method.

CODE:

#Create a Python class named Student.

class Student: def __init__(self,

name, age, grade):

self.name = name

self.age = age self.grade

= grade

def get_info(self): return f"Name: {self.name}, Age:

{self.age}, Grade: {self.grade}"

#Add aMributes: name, roll_number, branch.

self.roll_number = roll_number

self.branch = branch

#Use __init__ to iniRalize aMributes.

self.roll_number = roll_number self.branch

= branch

#Add display_details() to print student info.

def display_details(self): print(f"Name:

{self.name}") print(f"Age: {self.age}")

print(f"Grade: {self.grade}") print(f"Roll

Number: {self.roll_number}")

print(f"Branch: {self.branch}")

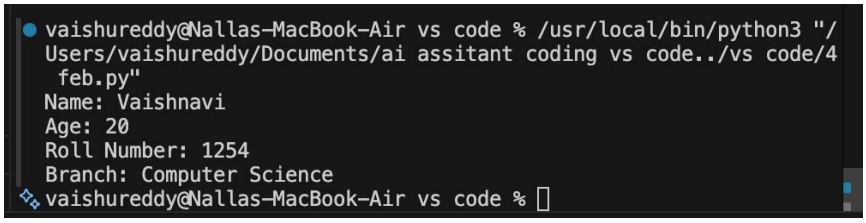
#Create one Student object. student1 = Student("SrinidhiReddy", 20, "A",

"1566", "Computer Science")

#Call the display_details() method.

student1.display_details()

OUTPUT:



-

EXPLANATION:

This program uses Object-Oriented Programming to model student details. A Student class is created with aMributes iniRalized using a constructor. A method is used to display student informaRon. This improves code organizaRon and reusability**.**

on #2: Loops (Mul9ples of a Number)

#Write a Python funcRon that prints the first 10 mulRples of a given number using a for loop.

#The number should be taken as user input.

Task Descrip9

PROMPT:

#Write the same program to print the first 10 mulRples of a given number,use a while loop

CODE:

#Write a Python funcRon that prints the first 10 mulRples of a given number using a for loop.

#The number should be taken as user input. def

print_mulRples():

try:

number = int(input("Enter a number: "))

print(f"The first 10 mulRples of {number} are:")

for i in range(1, 11): print(number * i)

except ValueError:

print("Please enter a valid integer.") print_mulRples()

#Write the same program to print the first 10 mulRples of a given number,use a while loop

def print_mulRples_while():

try:

number = int(input("Enter a number: "))

print(f"The first 10 mulRples of {number} are:")

count = 1 while count <= 10:

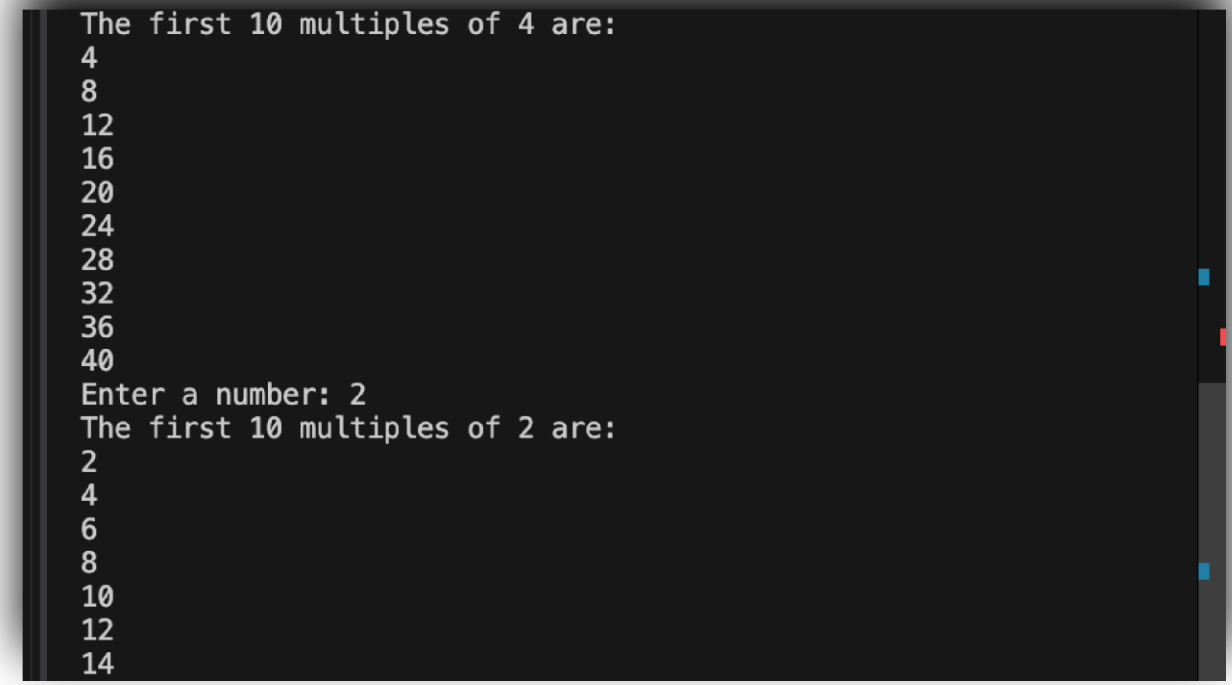
print(number * count)

count += 1 except ValueError:

print("Please enter a valid integer.")

print_mulRples_while()

OUTPUT:



EXPLANATION:

- Uses for loop to generate first 10 mulRples.
- Uses while loop as an alternaRve approach.
    - Accepts user input for flexibility.
- Includes error handling for invalid input**.**

on #3: Condi9onal Statements (Age Classifica9on)

Generate nested if-elif-else condiRonal statements to classify age groups(e.g., child, teenager, adult, senior).

Generate the same classificaRon using alternaRve condiRonal structures (e.g.,simplified condiRons or dicRonary-based logic).

CODE:

Generate nested if-elif-else condiRonal statements to classify age groups(e.g., child, teenager, adult, senior).

age = int(input("Enter your age: ")) if

age < 0:

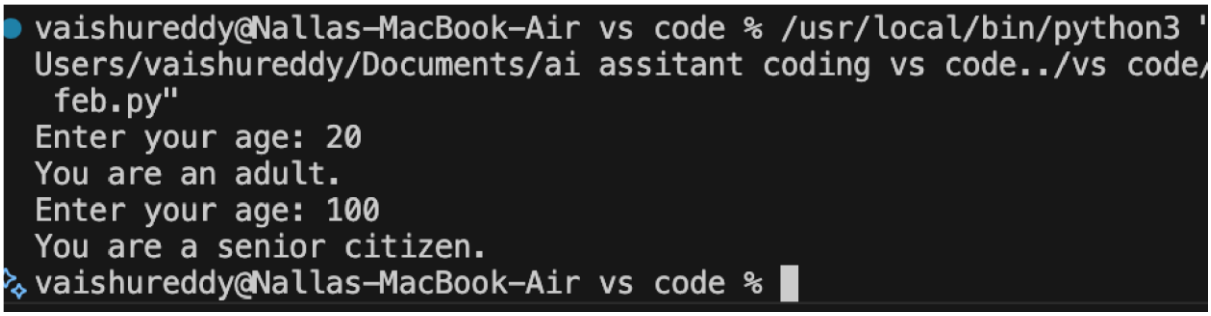
Task Descrip9

PROMPT:

```
   print("Invalid age.") elif
age < 13:
   print("You are a child.") elif
age < 20:
   print("You are a teenager.") elif
age < 65:
   print("You are an adult.") else:
print("You are a senior ciRzen.")


# Generate the same classificaRon using alternaRve condiRonal structures (e.g.,simplified
condiRons or dicRonary-based logic). def classify_age(age):
   """Classify age groups using dicRonary-based logic."""
age_groups = {        range(0, 13): "You are a child.",
range(13, 20): "You are a teenager.",        range(20, 65):
"You are an adult.",
      range(65, 150): "You are a senior ciRzen."
   }
   for age_range, message in age_groups.items():
if age in age_range:            return message
return "Invalid age." user_age = int(input("Enter
your age: ")) print(classify_age(user_age))
```

OUTPUT:



EXPLANATION:

- Applies condiRonal statements for decision making.
- Classifies age into child, teenager, adult, and senior.
- Demonstrates both nested and dicRonary-based logic.
- Improves readability and logical structure.

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

Generate a sum_to_n() funcRon using a for loop.

Generate an alternaRve implementaRon using a while loop or a mathemaRcal formula.

CODE:

Generate a sum_to_n() funcRon using a for loop. def

sum_to_n(n):

"""Calculate the sum of integers from 1 to n."""

total = 0 for i in range(1, n + 1):

total += i

return total

Example usage:

number = int(input("Enter a number to calculate the sum of integers from 1 to that number: "))

print(f"The sum of integers from 1 to {number} is {sum_to_n(number)}")

Generate an alternaRve implementaRon using a while loop or a mathemaRcal formula. def

sum_to_n_while(n):

"""Calculate the sum of integers from 1 to n using a while loop."""

Task Descrip9

PROMPT:

total = 0 i

= 1 while i

<= n:

total += i

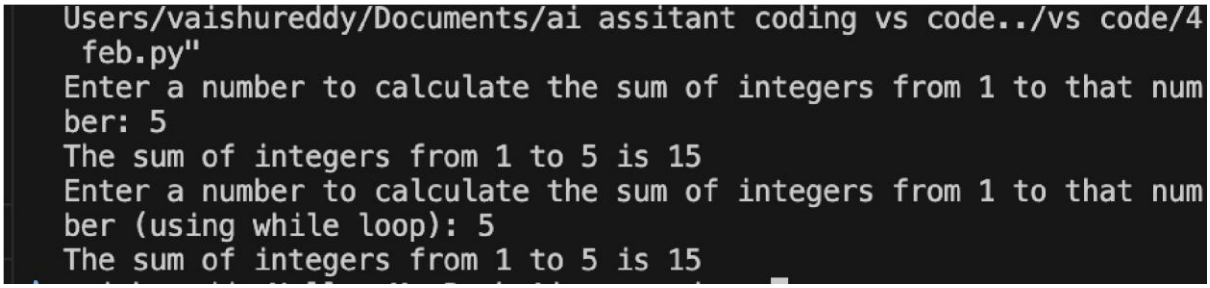
i += 1

return total

Example usage:

number = int(input("Enter a number to calculate the sum of integers from 1 to that number (using while loop): "))

print(f"The sum of integers from 1 to {number} is {sum_to_n_while(number)}") **OUTPUT:**



EXPLANATION:

- Calculates sum using a for loop.
- Provides alternaRve implementaRon using a while loop.
- Uses funcRons for reusability.
- Demonstrates loop control and arithmeRc operaRons.

Task Descrip9on #5: Classes (Bank Account Class)

PROMPT:

#Write a Python BankAccount class with deposit(), withdraw(), and check_balance() methods

Create an instance of BankAccount

Demonstrate the funcRonality of each method

CODE:

#Write a Python BankAccount class with deposit(), withdraw(), and check_balance()

methods class BankAccount: def __init__(self, iniRal_balance=0):

self.balance = iniRal_balance

  def deposit(self, amount):

if amount > 0:

    self.balance += amount

```
            print(f"Deposited: ${amount}")
        else:
            print("Deposit amount must be posiRve.")


    def withdraw(self, amount):        if
0 < amount <= self.balance:
self.balance -= amount
print(f"Withdrew: ${amount}")
        else:
            print("Insufficient funds or invalid withdrawal amount.")


    def check_balance(self):
        print(f"Current balance: ${self.balance}") # Create an instance of
BankAccount account = BankAccount(100)  # IniRal balance of $100
account.check_balance() account.deposit(50)
account.check_balance() account.withdraw(30)
account.check_balance() account.withdraw(150)  # AMempt to
withdraw more than the balance account.check_balance()
# Demonstrate the funcRonality of each method
account.deposit(-20)  # AMempt to deposit a negaRve
amount account.check_balance() account.withdraw(50)
account.check_balance() account.deposit(200)

account.check_balance()
account.withdraw(100)
account.check_balance() OUTPUT:
```

```
vaishureddy@Nallas-MacBook-Air vs code % /usr/local/bin/python3 "
Users/vaishureddy/Documents/ai assitant coding vs code../vs code/
 feb.py"
Withdrew: $30
Current balance: $120
Insufficient funds or invalid withdrawal amount.
Current balance: $120
Deposit amount must be positive.
Current balance: $120
Withdrew: $50
Current balance: $70
Deposited: $200
Current balance: $270
Withdrew: $100
Current balance: $170
vaishureddy@Nallas-MacBook-Air vs code %
```

EXPLANATION:

- Models real-world banking system using a class.
- Implements deposit, withdraw, and balance check methods.
- Ensures validaRon for transacRons.
- Demonstrates encapsulaRon and data security.