

ASSIGNMENT -6.3

2303a51343

b-10

TASK-1

Prompt: Create a Python Student class with name, roll_number, and branch.

Take all values from the user using input() Include __init__() and display_details() methods.

Create an object using user input and display the details

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}")

name = input("Enter student's name: ")
roll_number = input("Enter student's roll number: ")
branch = input("Enter student's branch: ")
student1 = Student(name, roll_number, branch)
student1.display_details()
```

Output:

The screenshot shows the Microsoft Visual Studio Code interface. In the Explorer sidebar, several files are listed: assignment3.3.py, assignment3.4.py, assignment4.3.py, assignment5.py, assignment6.3.py, python.py, and retailshop.py. The assignment6.3.py file is currently selected. The main editor area displays the following Python code:

```
#Create a Python Student class with name, roll_number, and branch.
#Take all values from the user using input().
#include __init__() and display_details() methods.
#create an object using user input and display the details.

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("Name: " + self.name)
        print("Roll Number: " + str(self.roll_number))
        print("Branch: " + self.branch)

name = input("Enter student's name: ")
roll_number = input("Enter student's roll number: ")
branch = input("Enter student's branch: ")

student1 = Student(name, roll_number, branch)
student1.display_details()
```

The terminal window at the bottom shows the execution of the code and its output:

```
PS C:\Users\Rashmita\OneDrive\Desktop\AI ASSISTED> & "C:\Users\Rashmita\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "C:\Users\Rashmita\vscode\extensions\ms-python.debugger-2025.19.2026012701\win32-x64\lib\debug\launcher" "50801" -- "C:\Users\Rashmita\OneDrive\Desktop\AI ASSISTED\assignment6.3.py"
Enter student's name: Rashmita
Enter student's roll number: 1343
Enter student's branch: cse
Name: Rashmita
Roll Number: 1343
Branch: cse
PS C:\Users\Rashmita\OneDrive\Desktop\AI ASSISTED>
```

The status bar at the bottom right indicates the date as 04-02-2026.

Analysis:

- AI-generated code correctly defines a Student class with a constructor to initialize name, roll number, and branch.
- The display_details() method clearly prints the student information, making the output easy to understand.
- Overall, the code is simple, readable, and follows proper object-oriented programming

TASK-2

Prompt: Write a Python program that asks the user for a number and prints the first 10 multiples using a for loop

Code:

```
def print_multiples(number):

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

    for i in range(1, 11):

        print(number * i, end=" ")

    print() # New line at the end

# Example usage

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

print_multiples(user_number)
```

Output:

The screenshot shows the Visual Studio Code interface with the 'AI ASSISTED' extension active. The Explorer sidebar shows several Python files. The main editor window displays a Python script named 'assignment6.3.py' which prints the first 10 multiples of a user-specified number. The terminal below shows the execution of the script and its output. A sidebar on the right provides AI assistance and recent session history.

```
20 student1 = Student(name, roll_number, branch)
21 student1.display_details()"""
22 #
23 #Write a Python program that asks the user for a number and prints the first 10 multiples using
24 #def print_multiples(number):
25     print("The first 10 multiples of " + str(number) + " are:")
26     for i in range(1, 11):
27         print(str(number) * i, end=" ")
28     print() # New line at the end
29 # Example usage
30 user_number = int(input("Enter a number: "))
31 print_multiples(user_number)
32
33
34
35 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

619:38 '... 'c:/Users/Rashmita/OneDrive/Desktop/AI ASSISTED/assignment6.3.py'
Enter a number: 6
The first 10 multiples of 6 are:
6 12 18 24 30 36 42 48 54 60

PS C:\Users\Rashmita\OneDrive\Desktop\AI ASSISTED

Ln 35, Col 1 Spaces: 4 UTF-8 CR/LF Python Go Live 09:51 ENG IN 04-02-2026

Analysis:

The AI-generated code correctly uses a loop to print the first 10 multiples of a given number.

The loop logic is clear and easy to understand. Both the for loop and while loop versions work properly and produce the expected output.

TASK-3:

Prompt: Write a Python program that takes age as user input and classifies it as child, teenager, adult, or senior using if-elif-else.

Code: def classify_age(age):

```
if age < 13:
    return "Child"
elif 13 <= age < 20:
    return "Teenager"
elif 20 <= age < 60:
    return "Adult"
else:
    return "Senior"

# Example usage
user_age = int(input("Enter your age: "))
```

```
age_group = classify_age(user_age)  
print(f"You are classified as: {age_group}")
```

Output:

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view with several Python files: `assignment3.3.py`, `assignment3.4.py`, `assignment4.3.py`, `assignment5.4.py`, and `assignment6.3.py`.
- Editor:** The main editor area displays Python code for classifying ages. The code defines a function `classify_age` that returns age groups based on input. It includes a usage example where it prints "You are classified as: Teenager".
- Terminal:** The terminal shows the command to run the script and its output. The output indicates the script was run from the `AI ASSISTED` folder, and the user input "15" resulted in the output "You are classified as: Teenager".
- Bottom Bar:** Includes icons for file operations like Save, Undo, Redo, and a Launchpad icon.
- Bottom Status Bar:** Shows the current file is `assignment6.3.py`, with status indicators for line 51, column 1, 4 spaces, CRLF, Python 3.13, and Go Live buttons.

Analysis: The AI-generated code uses simple if-elif-else conditions to check the age entered by the user. Each condition clearly matches an age group such as child, teenager, adult, or senior. The code is easy to read and gives correct results for different age values.

TASK-4

Prompt: Write a Python program that takes n as user input and calculates the sum of the first n natural numbers using a for loop.

Code: def sum_of_natural_numbers(n):

```
total = 0

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

return total

# Example usage

user_n = int(input("Enter a natural number n: "))

sum_n = sum_of_natural_numbers(user_n)

print(f"The sum of the first {user_n} natural numbers is: {sum_n}")
```

Output:

The screenshot shows the Visual Studio Code interface with the 'AI ASSISTED' extension active. The 'assignment6.3.py' file is open in the editor. The code defines a function to classify age groups (Teenager, Adult, Senior) and another to calculate the sum of natural numbers. A tooltip from the AI assistant suggests a formula for calculating the sum of natural numbers: $\text{sum} = \frac{n(n+1)}{2}$. The 'RECENT SESSIONS' panel shows two completed sessions related to the current file.

```
37     elif 13 <= age < 20:
38         return "Teenager"
39     elif 20 <= age < 60:
40         return "Adult"
41     else:
42         return "Senior"
43
44 # Example usage
45 user_age = int(input("Enter your age: "))
46 age_group = classify_age(user_age)
47 print(f"You are classified as {age_group}")
48
49 # Write a Python program that takes n as user input and calculates the sum of the first n natural
50 def sum_of_natural_numbers(n):
51     total = 0
52     for i in range(1, n + 1):
53         total += i
54     return total
55
56 # Example usage
57 user_n = int(input("Enter a natural number n: "))
58 sum_n = sum_of_natural_numbers(user_n)
59 print(f"The sum of the first {user_n} natural numbers is: {sum_n}")
60
61
62
```

Analysis: The AI-generated code correctly calculates the sum of the first n natural numbers using a loop. The logic is simple and easy to understand, whether it uses a for loop, a while loop, or a formula. The program gives the correct result for any positive number entered by the user.

TASK-5

Prompt: Create a Python Bankaccount class that takes account holder name and initial balance from the user. Include deposit(), withdraw(), and check_balance() methods. Allow the user to perform deposit and withdrawal operations using input().

Code: class BankAccount:

```
def __init__(self, holder_name, initial_balance):
    self.holder_name = holder_name
    self.balance = initial_balance

def deposit(self, amount):
    self.balance += amount
    print(f"Deposited: {amount}. New balance: {self.balance}")
```

```
def withdraw(self, amount):
    if amount > self.balance:
        print("Insufficient balance!")
```

```

else:
    self.balance -= amount
    print(f"Withdrew: {amount}. New balance: {self.balance}")

```

```

def check_balance(self):
    print(f"Current balance: {self.balance}")

holder_name = input("Enter account holder name: ")

initial_balance = float(input("Enter initial balance: "))

account = BankAccount(holder_name, initial_balance)

account.check_balance()

deposit_amount = float(input("Enter amount to deposit: "))

account.deposit(deposit_amount)

withdraw_amount = float(input("Enter amount to withdraw: "))

account.withdraw(withdraw_amount)

account.check_balance()

```

Output:

```

File Edit Selection View ... < > Q AI ASSISTED
EXPLORER SEARCH AI ASSISTED
assignment3.3.py assignment3.4.py assignment5.4.py assignment6.3.py
assignment3.3.py
assignment3.4.py
assignment4.3.py
assignment4.4.py
assignment5.4.py
assignment6.3.py
pythonpy
retailshop.py
Sum.java
AI ASSISTED
The sum of the first (user_n) natural numbers is: (sum_n)
#Create a Python BankAccount class that takes account holder name and initial balance from the user.
#Include deposit(), withdraw(), and check_balance() methods#Allow the user to perform deposit and withdrawal operations using inputs
class BankAccount:
    def __init__(self, holder_name, initial_balance):
        self.holder_name = holder_name
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited: {amount}. New balance: {self.balance}")

    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient balance!")
        else:
            self.balance -= amount
            print(f"Withdrew: {amount}. New balance: {self.balance}")

    def check_balance(self):
        print(f"Current balance: {self.balance}")

holder_name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))
account = BankAccount(holder_name, initial_balance)
account.check_balance()
deposit_amount = float(input("Enter amount to deposit: "))
account.deposit(deposit_amount)
withdraw_amount = float(input("Enter amount to withdraw: "))
account.withdraw(withdraw_amount)
account.check_balance()

Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITHUB

PS C:\Users\rashmitha\OneDrive\Desktop\VAI ASSISTED> cd "C:\Users\rashmitha\OneDrive\Desktop\VAI ASSISTED"; & "C:\Users\rashmitha\AppData\Local\Microsoft\Windows\Start Menu\Programs\Python\Python3.10\python.exe" "C:\Users\rashmitha\OneDrive\Desktop\VAI ASSISTED\assignment6.3.py"
Enter amount to withdraw: 200
Withdrew: 200.0. New balance: 4900.0
Current balance: 4900.0
PS C:\Users\rashmitha\OneDrive\Desktop\VAI ASSISTED> [REDACTED]

assignment6.3.py
Describe what to build next
Agent -> Auto >

Analysis: Your Bankaccount code is functional using classes and methods for deposit, withdrawal, and balance checks.

It handles basic operations and prevents overdrafts but lacks input validation and only allows one transaction of each type.

