

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING																		
Program Name: <b>B. Tech</b>		Assignment Type: Lab	Academic Year:2025-2026																	
Course Coordinator Name		Dr. Rishabh Mittal																		
Instructor(s) Name		<table border="1"> <tr><td>Mr. S Naresh Kumar</td></tr> <tr><td>Ms. B. Swathi</td></tr> <tr><td>Dr. Sasanko Shekhar Gantayat</td></tr> <tr><td>Mr. Md Sallauddin</td></tr> <tr><td>Dr. Mathivanan</td></tr> <tr><td>Mr. Y Srikanth</td></tr> <tr><td>Ms. N Shilpa</td></tr> <tr><td>Dr. Rishabh Mittal (Coordinator)</td></tr> <tr><td>Dr. R. Prashant Kumar</td></tr> <tr><td>Mr. Ankushavali MD</td></tr> <tr><td>Mr. B Viswanath</td></tr> <tr><td>Ms. Sujitha Reddy</td></tr> <tr><td>Ms. A. Anitha</td></tr> <tr><td>Ms. M.Madhuri</td></tr> <tr><td>Ms. Katherashala Swetha</td></tr> <tr><td>Ms. Velpula sumalatha</td></tr> <tr><td>Mr. Bingi Raju</td></tr> </table>		Mr. S Naresh Kumar	Ms. B. Swathi	Dr. Sasanko Shekhar Gantayat	Mr. Md Sallauddin	Dr. Mathivanan	Mr. Y Srikanth	Ms. N Shilpa	Dr. Rishabh Mittal (Coordinator)	Dr. R. Prashant Kumar	Mr. Ankushavali MD	Mr. B Viswanath	Ms. Sujitha Reddy	Ms. A. Anitha	Ms. M.Madhuri	Ms. Katherashala Swetha	Ms. Velpula sumalatha	Mr. Bingi Raju
Mr. S Naresh Kumar																				
Ms. B. Swathi																				
Dr. Sasanko Shekhar Gantayat																				
Mr. Md Sallauddin																				
Dr. Mathivanan																				
Mr. Y Srikanth																				
Ms. N Shilpa																				
Dr. Rishabh Mittal (Coordinator)																				
Dr. R. Prashant Kumar																				
Mr. Ankushavali MD																				
Mr. B Viswanath																				
Ms. Sujitha Reddy																				
Ms. A. Anitha																				
Ms. M.Madhuri																				
Ms. Katherashala Swetha																				
Ms. Velpula sumalatha																				
Mr. Bingi Raju																				
Course Code	23CS002PC304	Course Title	AI Assisted Coding																	
Year/Sem	III/II	Regulation	R23																	
Date and Day of Assignment	Week3 – Wednesday	Time(s)	23CSBTB01 To 23CSBTB52																	
Duration	2 Hours	Applicable to Batches	All batches																	
AssignmentNumber: <b>6.3</b> (Present assignment number)/ <b>24</b> (Total number of assignments)																				
Q.No.	Question	Expected Time to complete																		
1	<b>Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals</b> <b>Lab Objectives</b> <ul style="list-style-type: none"> <li>• To explore AI-powered auto-completion features for core Python constructs such as classes, loops, and conditional statements.</li> <li>• To analyze how AI tools suggest logic for object-oriented programming and control structures.</li> <li>• To evaluate the correctness, readability, and completeness of AI-generated Python code.</li> </ul>	Week3 - Wednesday																		

**Lab Outcomes (LOs)**

After completing this lab, students will be able to:

- Use AI tools to generate and complete Python class definitions and methods.
- Understand and assess AI-suggested loop constructs for iterative tasks.
- Generate and evaluate conditional statements using AI-driven prompts.
- Critically analyze AI-assisted code for correctness, clarity, and efficiency.

**Task Description #1: Classes (Student Class)****Scenario**

You are developing a simple student information management module.

**Prompt:** Write a Python program that defines a class named Student with attributes name, roll number, and branch. Use a constructor to initialize these values. Add a method display\_details() to print student information. In the main program, take name, roll number, and branch as user input, create a Student object, and display the details.

```

1 class Student:
2     print(f"Branch: {self.branch}")
3
4 if __name__ == "__main__":
5     # Read student details from user input
6     name = input("Enter student name: ").strip()
7     try:
8         roll_number = int(input("Enter roll number: ").strip())
9     except ValueError:
10        print("Invalid roll number entered. Defaulting to 0.")
11        roll_number = 0
12    branch = input("Enter branch: ").strip()
13
14    student = Student(name, roll_number, branch)
15    student.display_details()

```

Terminal Output:

```

PS C:\Users\sai\OneDrive\Desktop\AI> python student.py
Roll Number: 101
Branch: Computer Science
PS C:\Users\sai\OneDrive\Desktop\AI> python student.py
Enter student name: sai
Enter roll number: 2303A
Invalid roll number entered. Defaulting to 0.
Enter branch: 12345
Name: sai
Roll Number: 0
Branch: 12345

```

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

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

**Prompt:****1 – For Loop**

Write a Python function that asks the user to enter a number and prints the first 10 multiples of that number using a for loop.

**2-While Loop**

Rewrite the same program that prints the first 10 multiples of a number, but use a while loop instead of a for loop. Keep user input dynamic.

**3- Analysis**

Explain the difference between the for loop and while loop implementations for printing multiples of a number.

```
File Edit Selection View Go Run ... Q AI
EXPLORER
multiples.py
print_first_10_multiples
1 def print_first_10_multiples() -> None:
2     """Ask the user for a number and print its first 10 multiples."""
3     try:
4         n = int(input("Enter a number: ").strip())
5     except ValueError:
6         print("Invalid number entered.")
7         return
8     for i in range(1, 11):
9         print(n * i)
10
11
12
13 if __name__ == "__main__":
14     print_first_10_multiples()
15
PROBLEMS OUTPUT TERMINAL ... powershell + + - - - | | |
PS C:\Users\saiiku\OneDrive\Desktop\AI> python multiples.py
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
Ln 10, Col 21 Spaces: 4 UTF-8 CRLF Python conda (3.14.2) B4 Go Live
20°C Sunny 09:35 04-02-2026
```

```
File Edit Selection View Go Run ... Q AI
EXPLORER
multiples_while.py
print_first_10_multiples_while
1 def print_first_10_multiples_while() -> None:
2     i = 1
3     while i <= 10:
4         print(n * i)
5         i += 1
6
7
8 if __name__ == "__main__":
9     print_first_10_multiples_while()
10
PROBLEMS OUTPUT TERMINAL ... powershell + + - - - | | |
PS C:\Users\saiiku\OneDrive\Desktop\AI> python multiples_while.py
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
Ln 12, Col 15 Spaces: 4 UTF-8 CRLF Python conda (3.14.2) B4 Go Live
20°C Sunny 09:38 04-02-2026
```

```
File Edit Selection View Go Run ... Q AI
EXPLORER
multiples_while.py
print_first_10_multiples_while
1 def print_first_10_multiples_while() -> None:
2     i = 1
3     while i <= 10:
4         print(n * i)
5         i += 1
6
7
8 if __name__ == "__main__":
9     print_first_10_multiples_while()
10
PROBLEMS OUTPUT TERMINAL ... powershell + + - - - | | |
PS C:\Users\saiiku\OneDrive\Desktop\AI> python multiples_while.py
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
Ln 12, Col 15 Spaces: 4 UTF-8 CRLF Python conda (3.14.2) B4 Go Live
20°C Sunny 09:41 04-02-2026
```

**Task Description #3: Conditional Statements (Age Classification)****Scenario**

You are building a basic classification system based on age.

**Prompt****1 – if-elif-else**

Write a Python program that takes age as user input and classifies the person into Child (0–12), Teenager (13–19), Adult (20–59), or Senior (60+) using nested if-elif-else statements.

**2 – Alternative Approach**

Rewrite the age classification program using a different approach such as dictionary-based range mapping or simplified logical conditions. Take age as user input.

**3 – Explanation**

Explain how the conditions in the age classification program work and why the ranges do not overlap.

```
1 def classify_age() -> None:
2     print("Invalid input. Please enter an integer age.")
3     return
4
5     if age < 0:
6         print("Invalid age. Age cannot be negative.")
7     else:
8         if age <= 12:
9             print("Child (0-12)")
10        elif age <= 19:
11            print("Teenager (13-19)")
12        elif age <= 59:
13            print("Adult (20-59)")
14        else:
15            print("Senior (60+)")
16
17 if __name__ == "__main__":
18     classify_age()
```

Terminal output: Enter age: 20, Adult (20-59)

```
1 def classify_age_map() -> None:
2     """Prompt for age and classify using a range->label mapping."""
3     try:
4         age = int(input("Enter age: ").strip())
5     except ValueError:
6         print("Invalid input. Please enter an integer age.")
7         return
8
9     if age < 0:
10        print("Invalid age. Age cannot be negative.")
11        return
12
13    range_map = [
14        ((0, 12), "Child (0-12)"),
15        ((13, 19), "Teenager (13-19)"),
16        ((20, 59), "Adult (20-59)"),
17        ((60, float('inf')), "Senior (60+)")
18    ]
19
20    for (low, high), label in range_map:
21        if low <= age <= high:
22            print(label)
23            return
```

Terminal output: Enter age: 20, Adult (20-59)

```

1 def classify_age_map() -> None
2 """Prompt for age and clas
3 try:
4     age = int(input("Enter
5 except ValueError:
6     print("Invalid input.
7     return
8
9 if age < 0:
10    print("Invalid age. Ag
11    return
12
13 range_map = {
14     (0, 12), "Child (0-12
15     (13, 19), "Teenager (
16     (20, 59), "Adult (20
17     ((50, float('inf'))), "
18 }
19
20 for (low, high), label in
21     if low <= age <= high:
22         print(label)
23         return

```

**PYTHON CLASS CREATION FOR STUDENT ATTRIBUTES**

- Variable: `age` stores the user input (converted to `int`).
- Invalid check: `if age < 0:` catches negative ages first and stops further checks.
- Sequential checks: The code uses `if` → `elif` → `else`. Each condition is evaluated in order and once one is true the rest are skipped.
- Comparisons: Each branch uses simple numeric comparisons like `age < 12`, `age < 19`, `age < 59`. These test the upper bound for that category.

**Why the ranges don't overlap**

- **Implicit lower bounds:** Because of the sequence, an `elif` branch is only reached when all previous checks were false. For example, `elif age < 19:` is only evaluated when `age > 12`, so it effectively means `13 <= age < 19`.
- **Inclusive endpoints handled unambiguously:** Boundaries are chosen so each integer maps to exactly one category (e.g., 12 → Child, 13 → Teenager, 19 → Teenager, 20 → Adult, 59 → Adult, 60 → Senior).
- **elif vs separate ifs:** Using `elif` prevents multiple branches from matching; using independent `if` statements could allow overlaps and multiple outputs.
- **Alternate mapping approach:** The range-map version explicitly tests `low <= age <= high` for non-overlapping ranges like `(0,12)`, `(13,19)`, `(20,59)`, `(60,inf)`, which also guarantees uniqueness.

**SESSIONS**

**IN PROGRESS**

- Prime Number Check in Python Without F...  
Input needed. Local • 3 wks

**TODAY**

- Python class creation for Student attrib...  
Completed in 16c. Local • 42 sec

**LAST WEEK**

- User Data Collection and Protection in Pyt...  
Completed. Local • 5 days
- User Data Collection and Protection in Pyt...  
Completed. Local • 5 days
- Python code for calculating units consumed  
Completed. Local • 6 days
- Leap Year Determination Function in Python  
Completed. Local • 6 days

**OLDER**

- Python program for factorial calculation wi...  
Completed. Local • 3 wks
- write a program fibonacci series  
Completed. Local • 3 wks
- Code for adding two numbers  
Completed. Local • 3 wks

#### 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.

##### Prompt

##### 1 – For Loop

Write a Python function `sum_to_n()` that asks the user to enter a number n and calculates the sum of the first n natural numbers using a for loop.

##### Prompt 2 – Alternatives

Provide two more versions of the `sum_to_n()` program: one using a while loop and another using the mathematical formula  $n(n+1)/2$ . Use user input.

##### Prompt 3 – Comparison

Compare the loop-based and formula-based approaches for calculating the sum of n natural numbers in terms of efficiency and readability.

```

1 def sum_to_n() -> None:
2     """Prompt the user for n and print the sum of first n natural numbers."""
3     try:
4         n = int(input("Enter a number n: ").strip())
5     except ValueError:
6         print("Invalid input. Please enter an integer.")
7         return
8
9     if n <= 0:
10        print("Please enter a positive integer.")
11        return
12
13    total = 0
14    for i in range(1, n + 1):
15        total += i
16
17    print(f"Sum of first (n) natural numbers is: {total}")
18
19 if __name__ == "__main__":
20     sum_to_n()

```

**PYTHON CLASS CREATION FOR STUDENT A...**

- ✓ Organizing tool calls
- ✓ I'll add `sum_to_n.py`, implement `sum_to_n()` with input validation, and update the todo list.
- ✓ Starting: Create `sum_to_n.py` file (1/3)
- ✓ Edited: `sum_to_n.py` +0 -0
- ✓ Completed: Create `sum_to_n.py` file (1/3)
- Done — added `sum_to_n.py`.
- File: `sum_to_n.py`

**Run It:**

```
python sum_to_n.py
```

**Test script (2/3)**

- `sum_to_n.py`

**Describe what to build next:**

Agent: GPT-5 mini



The image displays three sequential screenshots of a Visual Studio Code (VS Code) editor interface, illustrating the development and execution of a Python program to calculate the sum of the first  $n$  natural numbers.

**Top Screenshot:** The editor shows the file explorer on the left with a list of files including `sum_to_n_formula.py`. The main editor window displays the code for `sum_to_n_formula.py`, which uses a mathematical formula to calculate the sum. The code is as follows:

```
def sum_to_n_formula() -> None:
    """Prompt the user for n and compute the sum of first n natural numbers using the formula"""
    try:
        n = int(input("Enter a positive integer n: ").strip())
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return
    if n <= 0:
        print("Please enter a positive integer.")
        return
    total = n * (n + 1) // 2
    print(f"Sum of first (n) natural numbers is: {total}")

if __name__ == "__main__":
    sum_to_n_formula()
```

The right-hand side of the interface shows a chat window titled "PYTHON CLASS CREATION FOR STUDENT A..." and a terminal window displaying the output of the program for  $n=10$ , resulting in a sum of 55.

**Middle Screenshot:** The editor shows the file explorer with `sum_to_n_while.py` selected. The main editor window displays the code for `sum_to_n_while.py`, which uses a loop to calculate the sum. The code is as follows:

```
def sum_to_n_while() -> None:
    """Prompt the user for n and compute the sum of first n natural numbers using a loop"""
    try:
        n = int(input("Enter a positive integer n: ").strip())
    except ValueError:
        print("Invalid input. Please enter an integer.")
        return
    if n <= 0:
        print("Please enter a positive integer.")
        return
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    print(f"Sum of first (n) natural numbers is: {total}")

if __name__ == "__main__":
    sum_to_n_while()
```

The right-hand side of the interface shows the same chat window and a terminal window displaying the output of the program for  $n=10$ , resulting in a sum of 55.

**Bottom Screenshot:** The editor shows the file explorer with `sum_to_n_while.py` selected. The main editor window displays the code for `sum_to_n_while.py`. The right-hand side of the interface shows a chat window titled "PYTHON CLASS CREATION FOR STUDENT ATTRIBUTES" and a terminal window displaying the output of the program for  $n=10$ , resulting in a sum of 55.

**Task Description #5: Classes (Bank Account Class)****Scenario**

You are designing a basic banking application.

**Prompt :** Write a complete Python program that defines a class BankAccount with attributes account holder\_name and balance. Use a constructor to initialize these values. Include methods deposit(amount), withdraw(amount), and check\_balance(), ensuring withdrawal is not allowed if the balance is insufficient.

In the main program, take user input for account holder name and initial balance. Then run a looped menu system where the user can repeatedly choose to deposit money, withdraw money, check balance, or exit.

For deposit and withdrawal operations, ask the user to enter the amount dynamically each time. Validate inputs to prevent negative deposits and overdrawing. Display the updated balance after every transaction. Add comments explaining the logic.

```

1  class BankAccount:
2      def __init__(self, holder_name, initial_balance):
3          self.holder_name = holder_name
4          self.balance = initial_balance
5
6      def deposit(self, amount):
7          """Deposit money into the account. Validates positive amount and updates balance."""
8          if amount > 0:
9              self.balance += amount
10             print(f"Deposited {amount}. New balance: {self.balance}")
11         else:
12             print("Invalid amount. Deposit cancelled.")
13
14      def withdraw(self, amount):
15          """Withdraw money from the account. Validates positive amount and sufficient balance. If insufficient, prints error and returns False. If successful, updates balance and returns True."""
16          if amount > 0 and self.balance >= amount:
17              self.balance -= amount
18              print(f"Withdrew {amount}. New balance: {self.balance}")
19              return True
20          else:
21              print("Invalid amount or insufficient funds. Withdrawal cancelled.")
22              return False
23
24      def check_balance(self):
25          """Display the current account balance."""
26          print(f"Account holder: {self.holder_name}. Balance: {self.balance}")
27
28  # Main program
29  holder_name = input("Enter account holder name: ").strip()
30  initial_balance = float(input("Enter initial balance: ").strip())
31
32  # Create BankAccount object
33  account = BankAccount(holder_name, initial_balance)
34
35  # Looped menu system
36  while True:
37      print("\nChoose an option:")
38      print("1. Deposit")
39      print("2. Withdraw")
40      print("3. Check balance")
41      print("4. Exit")
42      choice = input("Enter choice (1-4): ").strip()
43
44      if choice == '1':
45          # Deposit flow: ask for amount, validate positive
46          amount = float(input("Enter amount to deposit: ").strip())
47          try:
48              account.deposit(amount)
49          except ValueError:
50              print("Invalid amount. Deposit cancelled.\n")
51              continue
52          if amount <= 0:
53              print("Deposit amount must be positive.\n")
54              continue
55          account.deposit(amount)
56          print()
57      elif choice == '2':
58          # Withdraw flow: ask for amount, validate positive and sufficient funds
59          amount = float(input("Enter amount to withdraw: ").strip())
60          try:
61              account.withdraw(amount)
62          except ValueError:
63              print("Invalid amount. Withdrawal cancelled.\n")
64              continue
65
66  # Exit the program
67  print("Exiting the program. Goodbye!")

```

```

PS C:\Users\saiku\OneDrive\Desktop\AI> python bank_account.py
Enter account holder name: sai
Enter initial balance: 1000
Choose an option:
1. Deposit
2. Withdraw
3. Check balance
4. Exit
Enter choice (1-4): 1
Enter amount to deposit: 100
Deposited 100. New balance: 1100
Account holder: sai. Balance: 1100.00

Choose an option:
1. Deposit
2. Withdraw
3. Check balance
4. Exit
Enter choice (1-4): 2
Enter amount to withdraw: 100
Withdrew 100. New balance: 1000.00
Account holder: sai. Balance: 1000.00

Choose an option:
1. Deposit
2. Withdraw
3. Check balance
4. Exit
Enter choice (1-4): 3
Account holder: sai. Balance: 1000.00

Choose an option:
1. Deposit

```

**Note:** Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.