

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		Assignment Type: Lab	
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS 2 (Mounika)	
CourseCode	24CS002PC215	CourseTitle	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week3 - Wednesday	Time(s)	
Duration	2 Hours	Applicableto Batches	
AssignmentNumber:6.3(Present assignment number)/24(Total number of assignments)			

Q.No.	Question	Expected Time to complete
1	Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals Lab Objectives:	Week3 - Wednesday

- To explore AI-powered auto-completion features for core Python constructs.
- To analyze how AI suggests logic for class definitions, loops, and conditionals.
- To evaluate the completeness and correctness of code generated by AI assistants.

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Use AI tools to generate and complete class definitions and methods.
- Understand and assess AI-suggested loops for iterative tasks.
- Generate conditional statements through prompt-driven suggestions.
- Critically evaluate AI-assisted code for correctness and clarity.

Task Description#1 (Classes)

- Use AI to complete a Student class with attributes and a method.
- Check output
- Analyze the code generated by AI tool

Expected Output#1

- Class with constructor and display_details() method

Prompt-1: Create a python function complete a Student class with attributes and a method. • Class with constructor and display_details() method.

CODE:

```
class Student:
    def __init__(self, name, student_id, age, course):
        self.name = name
        self.student_id = student_id
        self.age = age
        self.course = course

    def display_details(self):
        """
        Method to display all student [details].
        """
        print("Student Details:")
        print(f"Name: {self.name}")
        print(f"Student ID: {self.student_id}")
        print(f"Age: {self.age}")
        print(f"Course: {self.course}")

if __name__ == "__main__":
    # Take input from user
    print("Enter Student Details:")
    print("-" * 30)

    name = input("Enter student name: ")
    student_id = input("Enter student ID: ")
    age = int(input("Enter student age: "))
    course = input("Enter course/major: ")

    # Create a student object with user input
    student = Student(name, student_id, age, course)

    print("\n" + "="*30 + "\n")
    # Display student [details]
    student.display_details()
```

OUTPUT:

```

PS C:\Users\moham\Desktop\Python> & "C:/Program Files/Python314/python.exe" c:/Users/moham/Desktop/Python/Code.py
Enter Student Details:
-----
Enter student name: Mohammed Nizamuddin
Enter student ID: 2503B05144
Enter student age: 22
Enter course/major: MTech CSE
-----
Student Details:
Name: Mohammed Nizamuddin
Student ID: 2503B05144
Age: 22
Course: MTech CSE

```

Task Description#2 (Loops)

- Prompt AI to complete a function that prints the first 10 multiples of a number using a loop.
- Analyze the generated code
- Ask AI to generate code using other controlled looping

Expected Output#2

- Correct loop-based implementation

Prompt: create a python a function that prints the first 10 multiples of a number using a loop
And generate code using other controlled looping.

CODE:

```

def print_multiples_for(number):
    print(f"\nUsing For Loop:")
    print(f"First 10 multiples of {number}:")
    print("-" * 40)
    for i in range(1, 11):
        multiple = number * i
        print(f"{number} x {i} = {multiple}")
> def print_multiples_while(number):...
if __name__ == "__main__":
    print("=" * 50)
    print("PROGRAM: Print First 10 Multiples of a Number")
    print("=" * 50)
    try:
        number = int(input("\nEnter a number: "))

        print_multiples_for(number)
        print_multiples_while(number)

    except ValueError:
        print("Error: Please enter a valid integer number!")
    except Exception as e:
        print(f"An error occurred: {e}")

```

OUTPUT:

```
Enter a number: 3

Using For Loop:
First 10 multiples of 3:
-----
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

Using While Loop:
First 10 multiples of 3:
-----
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

Task Description#3 (Conditional Statements)

- Ask AI to write nested if-elif-else conditionals to classify age groups.
- Analyze the generated code
- Ask AI to generate code using other conditional statements

Expected Output#3

- Age classification function with appropriate conditions and with explanation

Prompt-1: create a python code nested if-elif-else conditionals to classify age groups.

CODE:

```
def classify_age_nested(age):
    if age < 0:
        return "Invalid age"
    elif age <= 2:
        return "Infant"
    elif age <= 12:
        return "Child"
    elif age <= 17:
        return "Teenager"
    elif age <= 64:
        if age <= 25:
            return "Young Adult"
        elif age <= 39:
            return "Adult"
        else:
            return "Middle-aged Adult"
    elif age <= 79:
        return "Senior"
    else:
        return "Elderly"

def classify_age_simple(age):
    if age < 0:
        return "Invalid age"
    elif 0 <= age <= 2:
        return "Infant"
    elif 3 <= age <= 12:
        return "Child"
    elif 13 <= age <= 17:
        return "Teenager"
    elif 18 <= age <= 25:
        return "Young Adult"
```

```

        elif 26 <= age <= 39:
            return "Adult"
        elif 40 <= age <= 64:
            return "Middle-aged Adult"
        elif 65 <= age <= 79:
            return "Senior"
        else:
            return "Elderly"

def classify_age_ternary(age):
    if age < 0:
        return "Invalid age"
    return ("Infant" if age <= 2 else
            "Child" if age <= 12 else
            "Teenager" if age <= 17 else
            "Young Adult" if age <= 25 else
            "Adult" if age <= 39 else
            "Middle-aged Adult" if age <= 64 else
            "Senior" if age <= 79 else
            "Elderly")

if __name__ == "__main__":
    print("Age Classification Program")
    print("0-2: Infant | 3-12: Child | 13-17: Teenager | 18-25: Young Adult")
    print("26-39: Adult | 40-64: Middle-aged | 65-79: Senior | 80+: Elderly")
    print("-" * 60)

    try:
        age = int(input("\nEnter age: "))

```

```

try:
    age = int(input("\nEnter age: "))

    print(f"\nNested If-Elif-Else: {classify_age_nested(age)}")
    print(f"Simple If-Elif-Else: {classify_age_simple(age)}")
    print(f"Ternary Operators: {classify_age_ternary(age)}")

except ValueError:
    print("Error: Please enter a valid integer age!")

```

```

-----
Age Classification Program
0-2: Infant | 3-12: Child | 13-17: Teenager | 18-25: Young Adult
26-39: Adult | 40-64: Middle-aged | 65-79: Senior | 80+: Elderly
-----

Enter age: 21

Nested If-Elif-Else: Young Adult
Simple If-Elif-Else: Young Adult
Ternary Operators: Young Adult

```

Explanation:

1. NESTED IF-ELIF-ELSE:

- Uses nested conditions within elif blocks
- Allows for hierarchical decision making
- More complex structure but provides fine-grained control
- Example: First checks if age <= 64, then nested if checks sub-categories

2. SIMPLE IF-ELIF-ELSE:
- Sequential conditions with explicit ranges
 - Each condition uses 'and' to specify range boundaries
 - Easy to read and understand
 - Most common approach for age classification

TERNARY OPERATORS:

- Uses chained ternary operators (conditional expressions)
- Compact one-liner solution
- Less readable for complex conditions
- Pythonic but can be hard to maintain

Task Description#4 (For and While loops)

- Generate a sum_to_n() function to calculate sum of first n numbers
- Analyze the generated code
- Get suggestions from AI with other controlled looping

Expected Output#4

- Python code with explanation

Prompt: Generate a sum_to_n() function to calculate sum of first n 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_recursive(n):
    if n <= 0:
        return 0
    if n == 1:
        return 1
    return n + sum_to_n_recursive(n - 1)

def sum_to_n_formula(n):
    return n * (n + 1) // 2

if __name__ == "__main__":
    print("=" * 60)
    print("SUM OF FIRST N NUMBERS PROGRAM")
    print("=" * 60)

    try:
        n = int(input("\nEnter a positive integer (n): "))

        if n < 0:
            print("Error: Please enter a positive integer!")
        else:
            print("\n" + "=" * 60)
    
```

```

if n < 0:
    print("Error: Please enter a positive integer!")
else:
    print("\n" + "=" * 60)
    print("RESULTS USING DIFFERENT LOOPING METHODS:")
    print("=" * 60)

    result1 = sum_to_n_for_loop(n)
    print(f"For Loop: {result1}")

    result2 = sum_to_n_while_loop(n)
    print(f"While Loop: {result2}")

    result3 = sum_to_n_recursive(n)
    print(f"Recursive: {result3}")

    result4 = sum_to_n_formula(n)
    print(f"Mathematical Formula: {result4}")

```

OUTPUT:

```

=====
SUM OF FIRST N NUMBERS PROGRAM
=====

Enter a positive integer (n): 5
=====

RESULTS USING DIFFERENT LOOPING METHODS:
=====

For Loop: 15
While Loop: 15
Recursive: 15
Mathematical Formula: 15

```

EXPLANATION:

1. FOR LOOP:

- Iterates from 1 to n using range(1, n+1)
- Accumulates sum in a variable
- Simple and readable
- Time Complexity: O(n)
- Space Complexity: O(1)

2. WHILE LOOP:

- Uses counter variable initialized to 1
- Continues until counter > n
- Manual counter incrementation
- Time Complexity: O(n)
- Space Complexity: O(1)

3. RECURSIVE:

- Breaks problem into smaller subproblems
- Base case: n <= 0 returns 0, n == 1 returns 1
- Recursive case: n + sum(n-1)
- Time Complexity: O(n)
- Space Complexity: O(n) due to call stack

4. MATHEMATICAL FORMULA:

- Uses formula: $n * (n + 1) / 2$
- Most efficient method
- No loops needed
- Time Complexity: O(1)
- Space Complexity: O(1)

Task Description#5 (Class)

- Use AI to build a BankAccount class with deposit, withdraw, and balance methods.
- Analyze the generated code
- Add comments and explain code

Expected Output#5

- Python code with explanation

Prompt: a BankAccount class with deposit, withdraw, and balance methods.

```
class BankAccount:

    def __init__(self, account_number, account_holder, initial_balance=0.0):
        self.account_number = account_number
        self.account_holder = account_holder

        # Validate initial balance
        if initial_balance < 0:
            raise ValueError("Initial balance cannot be negative")
        self.balance = float(initial_balance)

    def deposit(self, amount):
        if amount <= 0:
            print("Error: Deposit amount must be positive!")
            return False

        self.balance += amount
        print(f"Deposit successful! Amount deposited: ${amount:.2f}")
        return True

    def withdraw(self, amount):
        if amount <= 0:
            print("Error: Withdrawal amount must be positive!")
            return False

        if amount > self.balance:
            print(f"Error: Insufficient balance! Current balance: ${self.balance:.2f}")
            return False
```

```

        self.balance -= amount
        print(f"Withdrawal successful! Amount withdrawn: ${amount:.2f}")
        return True

    def get_balance(self):
        return self.balance

    def display_balance(self):
        print(f"\nAccount Holder: {self.account_holder}")
        print(f"Account Number: {self.account_number}")
        print(f"Current Balance: ${self.balance:.2f}")

    def display_account_info(self):
        print("\n" + "=" * 50)
        print("ACCOUNT INFORMATION")
        print("=" * 50)
        print(f"Account Number: {self.account_number}")
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ${self.balance:.2f}")
        print("=" * 50)

if __name__ == "__main__":
    print("-" * 60)
    print("BANK ACCOUNT MANAGEMENT SYSTEM")
    print("-" * 60)

```

```

try:
    # Create a bank account
    print("\nCreating new bank account...")
    account_number = input("Enter account number: ")
    account_holder = input("Enter account holder name: ")

    initial_balance_input = input("Enter initial balance (press Enter for $0.00): ")
    initial_balance = float(initial_balance_input) if initial_balance_input else 0.0

    # Initialize bank account
    account = BankAccount(account_number, account_holder, initial_balance)
    print("\nAccount created successfully!")
    account.display_account_info()

    # Menu-driven program
    while True:
        print("\n" + "-" * 60)
        print("MENU:")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Display Account Info")
        print("5. Exit")
        print("-" * 60)

        choice = input("Enter your choice (1-5): ")

        if choice == "1":
            amount = float(input("Enter amount to deposit: $"))
            account.deposit(amount)
            account.display_balance()

```

```
choice = input("Enter your choice (1-5): ")

if choice == "1":
    amount = float(input("Enter amount to deposit: $"))
    account.deposit(amount)
    account.display_balance()

elif choice == "2":
    amount = float(input("Enter amount to withdraw: $"))
    account.withdraw(amount)
    account.display_balance()

elif choice == "3":
    balance = account.get_balance()
    print(f"\nCurrent Balance: ${balance:.2f}")

elif choice == "4":
    account.display_account_info()

elif choice == "5":
    print("\nThank you for using Bank Account Management System!")
    print("=" * 60)
    print("CODE ANALYSIS:")
    print("=" * 60)
```

OUTPUT:

```
=====  
BANK ACCOUNT MANAGEMENT SYSTEM  
=====  
  
Creating new bank account...  
Enter account number: 2503B05144  
Enter account holder name: Mohammed Nizamuddin  
Enter initial balance (press Enter for $0.00): 5000
```

```
Account created successfully!
```

```
=====  
ACCOUNT INFORMATION  
=====  
  
Account Number: 2503B05144  
Account Holder: Mohammed Nizamuddin  
Current Balance: $5000.00  
=====
```

```
-----  
MENU:  
1. Deposit  
2. Withdraw  
3. Check Balance  
4. Display Account Info  
5. Exit  
-----
```

```
Enter your choice (1-5): 3
```

```
Current Balance: $5000.00
```

```
-----  
MENU:  
1. Deposit  
2. Withdraw  
3. Check Balance  
4. Display Account Info  
5. Exit
```

```
Enter your choice (1-5): 1
Enter amount to deposit: $700
Deposit successful! Amount deposited: $700.00
```

```
Account Holder: Mohammed Nizamuddin
Account Number: 2503B05144
Current Balance: $5700.00
```

```
-----  
MENU:
```

1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit

```
-----  
Enter your choice (1-5): 4
```

```
=====  
ACCOUNT INFORMATION  
=====
```

```
Account Number: 2503B05144
Account Holder: Mohammed Nizamuddin
Current Balance: $5700.00
```

```
-----  
MENU:
```

1. Deposit
2. Withdraw
3. Check Balance
4. Display Account Info
5. Exit

```
-----  
Enter your choice (1-5): 5
```

```
Thank you for using Bank Account Management System!
```

Explanation:

CLASS STRUCTURE:

1. `__init__` (Constructor):
 - Initializes account with `account_number`, `account_holder`, and `balance`
 - Validates `initial_balance` to prevent negative values
 - Uses encapsulation to store account data as instance variables
 - Default parameter allows creating account with zero balance
2. `deposit(amount)`:
 - Validates deposit amount (must be positive)
 - Updates balance by adding deposit amount

	<ul style="list-style-type: none"> - Provides user feedback on success - Returns boolean for programmatic success checking <p>3. withdraw(amount):</p> <ul style="list-style-type: none"> - Validates withdrawal amount (must be positive) - Checks for sufficient balance before withdrawal - Prevents overdraft (withdrawal > balance) - Updates balance by subtracting withdrawal amount - Provides user feedback on success/failure <p>4. get_balance():</p> <ul style="list-style-type: none"> - Getter method to retrieve current balance - Provides controlled access to balance attribute - Returns float value of balance <p>5. display_balance():</p> <ul style="list-style-type: none"> - Formats and displays balance information - Shows account holder and account number - User-friendly output format <p>6. display_account_info():</p> <ul style="list-style-type: none"> - Displays complete account information - Formatted output for account summary <p>KEY CONCEPTS:</p> <p>1. ENCAPSULATION:</p> <ul style="list-style-type: none"> - Account data (balance, account_number, account_holder) is stored as instance variables - Methods provide controlled access to account data - Prevents direct modification of balance without validation <p>2. DATA VALIDATION:</p> <ul style="list-style-type: none"> - All methods validate input before processing - Prevents invalid operations (negative deposits, overdraft, etc.) - Provides error messages for invalid operations <p>3. METHOD DESIGN:</p> <ul style="list-style-type: none"> - Each method has a single responsibility - Methods return boolean values for success/failure - User-friendly error messages <p>4. OOP PRINCIPLES:</p> <ul style="list-style-type: none"> - Encapsulation: Data and methods bundled together - Abstraction: Hides implementation details - Data hiding: Balance accessed through methods <p>TIME COMPLEXITY:</p> <p>-----</p> <ul style="list-style-type: none"> - deposit(): O(1) - Constant time operation - withdraw(): O(1) - Constant time operation - get_balance(): O(1) - Constant time operation - All operations are constant time, making them efficient <p>SPACE COMPLEXITY:</p> <p>-----</p> <ul style="list-style-type: none"> - O(1) - Only stores account data (3 attributes) <p>Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots</p> <p>Evaluation Criteria:</p>	
--	---	--

	Criteria	Max Marks		
Class		1.0		
Loops		1.0		
Conditional Statements		0.5		
Total		2.5 Marks		