SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName: B. Tech		**Assignment Type: Lab**	**AcademicYear:** 2025-2026
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	*ExpectedTime to complete*
1	Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals **Lab Objectives:** • 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):**	Week3 - Wednesday

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

Instructions:

- **Initialize class with attributes like name, roll no, marks**
- **Method to display student details**
- **Method to calculate grade based on marks (A:>=90, B: >=75, C: >=60, else Fail)**

Start Writing code and auto complete using any AI tool

INPUT:

```python
class Student:
    def __init__(self, name, roll_no, marks):
        self.name = name
        self.roll_no = roll_no
        self.marks = marks

    def display_details(self):
        print(f"Name: {self.name}")
        print(f"Roll No: {self.roll_no}")
        print(f"Marks: {self.marks}")

    def calculate_grade(self):
        if self.marks >= 90:
            return 'A'
        elif self.marks >= 75:
            return 'B'
        elif self.marks >= 60:
            return 'C'
        else:
            return 'Fail'

# Test the class
if __name__ == "__main__":
    student1 = Student("Alice", 101, 85)
    student1.display_details()
    print(f"Grade: {student1.calculate_grade()}")

    student2 = Student("Bob", 102, 55)
    student2.display_details()
    print(f"Grade: {student2.calculate_grade()}")
```

OUTPUT:

```
PS C:\Users\welcome\Desktop\AIAC\lab 6.3> & 'c:\Users\welcome\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\welcome\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '52139' '--' 'c:\Users\welcome\Desktop\AIAC\lab 6.3\student.py'
Name: Alice
Roll No: 101
Marks: 85
Grade: B
Name: Bob
Roll No: 102
Marks: 55
Grade: Fail
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
```

Expected Output#1

- Class with constructor and display_details() method

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

Write code using **For** Loop, later complete code using **While** Loop

INPUT:

```python
def print_multiples_for(n):
    """
    Prints the first 10 multiples of the given number n using a for loop.
    """
    for i in range(1, 11):
        print(n * i)

def print_multiples_while(n):
    """
    Prints the first 10 multiples of the given number n using a while loop.
    """
    i = 1
    while i <= 10:
        print(n * i)
        i += 1

if __name__ == "__main__":
    number = int(input("Enter a number: "))
    print(f"\nFirst 10 multiples of {number} using for loop:")
    print_multiples_for(number)

    print(f"\nFirst 10 multiples of {number} using while loop:")
    print_multiples_while(number)
5
```

OUTPUT:

```
PS C:\Users\welcome\Desktop\AIAC\lab 6.3> ^C
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>  c:; cd 'c:\Users\welcome\Desktop\AIAC\lab 6.3'; & 'c:\Users\welcome\AppData\Local\Programs\Python\Python313\python.exe' 'c:
sers\welcome\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64249' '--' 'c:\Users\welcome\Desktop\AIAC\lab 6.3\TASK2.PY'
Enter a number: 5

First 10 multiples of 5 using for loop:
5
10
15
20
25
30
35
40
45
50

First 10 multiples of 5 using while loop:
5
10
15
20
25
30
35
40
45
50
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
```

E [Lab-6.3-1 assignment.docx](#)xpected Output#2

- Correct loop-based implementation

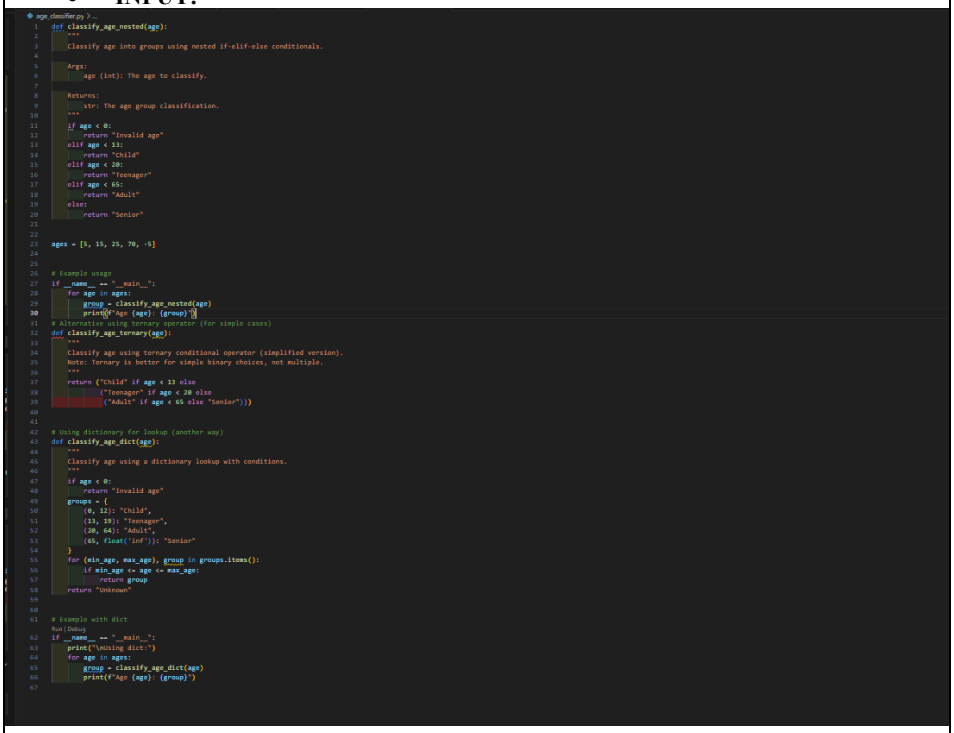
Task Description#3 (Conditional Statements)

- 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

Table: Age Group Classification Logic

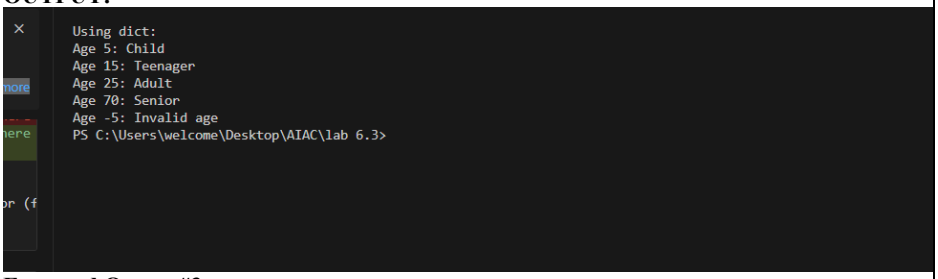
Age Range	Age Group
0 – 12 years	Child
13 – 19 years	Teen
20 – 59 years	Adult
60 years & above	Senior

-
- **INPUT:**



```python
def classify_age_nested(age):
    """
    Classify age into groups using nested if-elif-else conditionals.

    Args:
        age (int): The age to classify.

    Returns:
        str: The age group classification.
    """
    if age < 0:
        return "Invalid age"
    elif age < 13:
        return "Child"
    elif age < 20:
        return "Teenager"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"


ages = [5, 15, 25, 70, -5]

# Example usage
if __name__ == "__main__":
    for age in ages:
        group = classify_age_nested(age)
        print(f"Age {age}: {group}")
# Alternative using ternary operator (for simple cases)
def classify_age_ternary(age):
    """
    Classify age using ternary conditional operator (simplified version).
    Note: Ternary is better for simple binary choices, not multiple.
    """
    return ("Child" if age < 13 else
            ("Teenager" if age < 20 else
             ("Adult" if age < 65 else "Senior")))


# Using dictionary for lookup (another way)
def classify_age_dict(age):
    """
    Classify age using a dictionary lookup with conditions.
    """
    if age < 0:
        return "Invalid age"
    groups = {
        (0, 12): "Child",
        (13, 19): "Teenager",
        (20, 64): "Adult",
        (65, float('inf')): "Senior"
    }
    for (min_age, max_age), group in groups.items():
        if min_age <= age <= max_age:
            return group
    return "Unknown"


# Example with dict
if __name__ == "__main__":
    print("\nUsing dict:")
    for age in ages:
        group = classify_age_dict(age)
        print(f"Age {age}: {group}")
```

OUTPUT:



```
Using dict:
Age 5: Child
Age 15: Teenager
Age 25: Adult
Age 70: Senior
Age -5: Invalid age
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
```

Expected Output#3

- Age classification function with appropriate conditions and with explanation

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

INPUT:

```python
def sum_to_n_for(n):
    """Sum of 1 to n using a for loop."""
    total = 0
    for i in range(1, n + 1):
        total += i
    return total

def sum_to_n_while(n):
    """Sum of 1 to n using a while loop."""
    total = 0
    i = 1
    while i <= n:
        total += i
        i += 1
    return total

def sum_to_n_recursive(n):
    """Sum of 1 to n using recursion."""
    if n == 1:
        return 1
    return n + sum_to_n_recursive(n - 1)

def sum_to_n_formula(n):
    """Sum of 1 to n using the direct formula."""
    return n * (n + 1) // 2

if __name__ == "__main__":
    n = int(input("Enter a positive integer n: "))
    print(f"Using for loop: {sum_to_n_for(n)}")
    print(f"Using while loop: {sum_to_n_while(n)}")
    print(f"Using recursion: {sum_to_n_recursive(n)}")
    print(f"Using formula: {sum_to_n_formula(n)}")

```

OUTPUT:

```
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>  c:; cd 'c:\Users\welcome\Desktop\AIAC\lab 6.3'; & 'C:\Users\welcome\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\welcome\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '65078' '--' 'c:\Users\welcome\Desktop\AIAC\lab 6.3\TASK4.PY'
Enter a positive integer n: 4
Using for loop: 10
Using while loop: 10
Using recursion: 10
Using formula: 10
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
```

Expected Output#4

- Python code with explanation

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

Instructions

- **Initialize BankAccount class with attributes like name, balance**
- **Method to deposit amount**
- **Method to withdraw amount**
- **Method to check balance**

INPUT:

bank_account.py > Pylance > BankAccount

```python
class BankAccount:
    """
    A simple BankAccount class to manage a bank account with basic operations.
    """

    def __init__(self, name, balance=0):
        """
        Initialize the BankAccount with a name and an optional initial balance.

        Args:
            name (str): The name of the account holder.
            balance (float): The initial balance of the account. Defaults to 0.
        """
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        """
        Deposit a specified amount into the account.

        Args:
            amount (float): The amount to deposit. Must be positive.

        Returns:
            None
        """
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount:.2f}. New balance: ${self.balance:.2f}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """
        Withdraw a specified amount from the account if sufficient funds are available.

        Args:
            amount (float): The amount to withdraw. Must be positive and not exceed balance.

        Returns:
            None
        """
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ${amount:.2f}. "
                  f"New balance: ${self.balance:.2f}")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def get_balance(self):
        """
        Get the current balance of the account.

        Returns:
            float: The current balance.
        """
        return self.balance

# Example usage
if __name__ == "__main__":
    account = BankAccount("John Doe", 1000)
    print(f"Account holder: {account.name}")
    print(f"Initial balance: ${account.get_balance():.2f}")

    account.deposit(500)
    account.withdraw(200)
    account.withdraw(1500)  # Should fail due to insufficient funds
    print(f"Final balance: ${account.get_balance():.2f}")
```

OUTPUT:

```
de\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58073' '--' 'c:\Users\welcome\Desktop\AIAC\lab 6.3\bank_account.py'

Account holder: John Doe
Initial balance: $1000.00
Deposited $500.00. New balance: $1500.00
Withdrew $200.00. New balance: $1300.00
Invalid withdrawal amount or insufficient funds.
Final balance: $1300.00
PS C:\Users\welcome\Desktop\AIAC\lab 6.3>
```

Expected Output#5

- Python code with explanation

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

Evaluation Criteria:

Criteria	Max Marks
Class	1.0
Loops	1.0

	Conditional Statements	0.5		
	Total	**2.5 Marks**		