SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name		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)	
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week3 – Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
Assignment Number:5.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices **Lab Objectives:** • To explore the ethical risks associated with AI-generated code.	Week3 - Monday

- To recognize issues related to security, bias, transparency, and copyright.
- To reflect on the responsibilities of developers when using AI tools in software development.
- To promote awareness of best practices for responsible and ethical AI coding.

Lab Outcomes (LOs):
After completing this lab, students will be able to:

- Identify and avoid insecure coding patterns generated by AI tools.
- Detect and analyze potential bias or discriminatory logic in AI-generated outputs.
- Evaluate originality and licensing concerns in reused AI-generated code.
- Understand the importance of explainability and transparency in AI-assisted programming.
- Reflect on accountability and the human role in ethical AI coding practices..

Task Description #1 (Privacy in API Usage)
Task: Use an AI tool to generate a Python program that connects to a weather API.
Prompt:
"Generate code to fetch weather data securely without exposing API keys in the code."
Expected Output:

- Original AI code (check if keys are hardcoded).
- Secure version using environment variables.

Code:

```
import requests
from config import API_KEY

def fetch_weather(city):
    url =
f"http://api.openweathermap.org/data/2.5/weather?q={city}&app
id={API_KEY}"
    response = requests.get(url)
    data = response.json()
    if response.status_code != 200:
```

```
        raise Exception(f"Error fetching weather data:
{data.get('message', 'Unknown error')}")

    return  {
        'city': data['name'],
        'temperature': data['main']['temp'],
        'weather': data['weather'][0]['description']
    }


if __name__ == "__main__":
    city = input("Enter the city name: ")
    weather_data = fetch_weather(city)
    print(weather_data)
```

Output :
Enter the city name: London
{'city': 'London', 'temperature': 290.45, 'weather': 'broken clouds'}

Task Description #2 (Privacy & Security in File Handling)
Task: Use an AI tool to generate a Python script that stores user data (name, email, password) in a file.
Analyse: Check if the AI stores sensitive data in plain text or without encryption.
Expected Output:

- Identified privacy risks.
- Revised version with encrypted password storage (e.g., hashing).

Prompt : write a python program to generate a script that stores user data (name, email, password) in a file with and without security.
Code:
-Without security

```
def store_user_data():
    """Store user data in a file"""
    print("User Registration System")
    print("========================")

    # Get user input
    name = input("Enter your name: ")
    email = input("Enter your email: ")
    password = input("Enter your password: ")

    # Store data in file
```

```
    with open("users.txt", "a") as file:
        file.write(f"Name: {name}, Email: {email}, Password:
{password}\n")

    print("User data stored successfully!")

def read_user_data():
    """Read and display user data from file"""
    try:
        with open("users.txt", "r") as file:
            print("\nStored User Data:")
            print("==================")
            for line in file:
                print(line.strip())
    except FileNotFoundError:
        print("No user data found.")

# Main program
if __name__ == "__main__":
    while True:
        print("\nOptions:")
        print("1. Register new user")
        print("2. View all users")
        print("3. Exit")

        choice = input("Enter your choice (1-3): ")

        if choice == "1":
            store_user_data()
        elif choice == "2":
            read_user_data()
        elif choice == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

Output:
Options:
1. Register new user
2. View all users
3. Exit
Enter your choice (1-3): 1
User Registration System

==========

Enter your name: hello
Enter your email: pqr@gmail.com
Enter your password: hello@pqr
User data stored successfully!

Options:
1. Register new user
2. View all users
3. Exit
Enter your choice (1-3): 2

Stored User Data:
==========

Name: hello, Email: pqr@gmail.com, Password: hello@pqr

Options:
1. Register new user
2. View all users
3. Exit
Enter your choice (1-3): 3
Goodbye!

-With security

```
import bcrypt
import re
import os
from getpass import getpass

def validate_email(email):
    """Validate email format using regex"""
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-
Z]{2,}$'
    return re.match(pattern, email) is not None

def hash_password(password):
    """Hash password using bcrypt with salt"""
    salt = bcrypt.gensalt()
    hashed_password = bcrypt.hashpw(password.encode('utf-8'),
salt)
    return hashed_password.decode('utf-8')
```

```
def verify_password(stored_hash, password):
    """Verify password against stored hash"""
    return bcrypt.checkpw(password.encode('utf-8'),
stored_hash.encode('utf-8'))

def store_user_data_secure():
    """Store user data securely with hashed password"""
    print("Secure User Registration System")
    print("==============================")

    # Get user input with validation
    name = input("Enter your name: ").strip()
    if not name:
        print("Name cannot be empty!")
        return

    email = input("Enter your email: ").strip()
    if not validate_email(email):
        print("Invalid email format!")
        return

    # Use getpass for secure password input (doesn't echo to
screen)
    password = getpass("Enter your password: ")
    if len(password) < 8:
        print("Password must be at least 8 characters long!")
        return

    # Hash the password
    hashed_password = hash_password(password)

    # Store data in file (only store hashed password)
    try:
        with open("users_secure.txt", "a") as file:
            file.write(f"Name: {name}, Email: {email},
PasswordHash: {hashed_password}\n")

        print("User data stored securely!")

    except IOError as e:
        print(f"Error storing data: {e}")

def read_user_data_secure():
    """Read and display user data from secure file"""
    try:
```

```
        with open("users_secure.txt", "r") as file:
            print("\nStored User Data (Secure):")
            print("==========================")
            for line in file:
                # Mask the password hash for display
                if "PasswordHash:" in line:
                    parts = line.split("PasswordHash:")
                    display_line = parts[0] + "PasswordHash:
[HASHED]"
                    print(display_line.strip())
                else:
                    print(line.strip())
    except FileNotFoundError:
        print("No secure user data found.")

def verify_user_login():
    """Verify user login credentials"""
    print("\nUser Login Verification")
    print("=======================")

    email = input("Enter your email: ").strip()
    password = getpass("Enter your password: ")

    try:
        with open("users_secure.txt", "r") as file:
            for line in file:
                if f"Email: {email}" in line:
                    # Extract the stored hash
                    hash_start = line.find("PasswordHash: ")
+ len("PasswordHash: ")
                    stored_hash = line[hash_start:].strip()

                    if verify_password(stored_hash,
password):
                        print("Login successful!")
                        return
                    else:
                        print("Invalid password!")
                        return

        print("User not found!")

    except FileNotFoundError:
        print("No user data found.")
    except Exception as e:
```

```
        print(f"Error during login: {e}")

# Main program
if __name__ == "__main__":
    while True:
        print("\nSecure Options:")
        print("1. Register new user (secure)")
        print("2. View all users (secure display)")
        print("3. Verify user login")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            store_user_data_secure()
        elif choice == "2":
            read_user_data_secure()
        elif choice == "3":
            verify_user_login()
        elif choice == "4":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

Output :

Secure Options:
1. Register new user (secure)
2. View all users (secure display)
3. Verify user login
4. Exit
Enter your choice (1-4): 1
Secure User Registration System
==========
Enter your name: hi
Enter your email: xyz@gmail.com
Enter your password:
User data stored securely!

Secure Options:
1. Register new user (secure)
2. View all users (secure display)
3. Verify user login

4. Exit
Enter your choice (1-4): 3

User Login Verification
==========
Enter your email: xyz@gmail.com
Enter your password:
Login successful!

Secure Options:
1. Register new user (secure)
2. View all users (secure display)
3. Verify user login
4. Exit
Enter your choice (1-4): 2

Stored User Data (Secure):
==========
Name: hi, Email: xyz@gmail.com, PasswordHash: [HASHED]

Secure Options:
1. Register new user (secure)
2. View all users (secure display)
3. Verify user login
4. Exit
Enter your choice (1-4): 4
Goodbye!

Task Description #3 (Transparency in Algorithm Design)
Objective: Use AI to generate an Armstrong number checking function with comments and explanations.
Instructions:

1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

- Transparent, commented code.
- Correct, easy-to-understand explanation.

Prompt: Write a Python program to generate an Armstrong number checking function with comments and explanations.

Code:

```
def is_armstrong_number(num):
    """
    Check if a number is an Armstrong number.

    An Armstrong number (also known as a narcissistic number) is a number that is equal to the sum of its own digits each raised to the power of the number of digits.

    Parameters:
    num (int): The number to check.

    Returns:
    bool: True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to a string to easily iterate over digits
    digits = str(num)
    num_digits = len(digits)  # Get the number of digits

    # Calculate the sum of each digit raised to the power of the number of digits
    armstrong_sum = sum(int(digit) ** num_digits for digit in digits)

    # Check if the calculated sum is equal to the original number
    return armstrong_sum == num

# Example usage
if __name__ == "__main__":
    test_number = 153
    if is_armstrong_number(test_number):
        print(f"{test_number} is an Armstrong number.")
    else:
        print(f"{test_number} is not an Armstrong number.")
```

Output :

153 is an Armstrong number.

Task Description #4 (Transparency in Algorithm Comparison)
Task: Use AI to implement two sorting algorithms (e.g., QuickSort and BubbleSort).
Prompt:
"Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ."
Expected Output:

- Code for both algorithms.
- Transparent, comparative explanation of their logic and efficiency.

Code :
Quick Sort

```
def quicksort(arr):
    """
    QuickSort Algorithm Implementation

    QuickSort is a divide-and-conquer algorithm that works
by:
    1. Selecting a 'pivot' element from the array
    2. Partitioning the array around the pivot (elements <
pivot go left, elements > pivot go right)
    3. Recursively applying the same process to the left and
right sub-arrays

    Time Complexity:
    - Best/Average case: O(n log n)
    - Worst case: O(n²) - when pivot is always the smallest
or largest element

    Space Complexity: O(log n) - due to recursion stack

    Characteristics:
    - In-place sorting (modifies the original array)
    - Unstable sort (may change relative order of equal
elements)
    - Efficient for large datasets
    """
```

```
    def _quicksort(arr, low, high):
        if low < high:
            # Partition the array and get the pivot index
            pivot_index = partition(arr, low, high)

            # Recursively sort elements before and after
partition
            _quicksort(arr, low, pivot_index - 1)
            _quicksort(arr, pivot_index + 1, high)

    def partition(arr, low, high):
        """
        Partition function that places the pivot element in
its correct position
        and arranges all smaller elements to the left and
larger to the right
        """
        # Choose the rightmost element as pivot
        pivot = arr[high]

        # Index of smaller element (indicates right position
of pivot)
        i = low - 1

        for j in range(low, high):
            # If current element is smaller than or equal to
pivot
            if arr[j] <= pivot:
                i += 1
                # Swap arr[i] and arr[j]
                arr[i], arr[j] = arr[j], arr[i]

        # Place pivot in correct position
        arr[i + 1], arr[high] = arr[high], arr[i + 1]
        return i + 1

    # Start the sorting process
    _quicksort(arr, 0, len(arr) - 1)
    return arr


# Example usage and testing
if __name__ == "__main__":
    # Test the quicksort algorithm
    test_array = [64, 34, 25, 12, 22, 11, 90, 5]
```

```
    print("Original array:", test_array)

    sorted_array = quicksort(test_array.copy())
    print("QuickSorted array:", sorted_array)

    # Performance comparison with BubbleSort
    print("\n--- QuickSort vs BubbleSort Comparison ---")
    print("QuickSort Advantages:")
    print("- Much faster for large datasets (O(n log n) vs
O(n²))")
    print("- More cache-efficient due to good locality of
reference")
    print("- In-place sorting with low memory overhead")

    print("\nQuickSort Disadvantages:")
    print("- Worst-case performance can be O(n²)")
    print("- Not stable (may change order of equal
elements)")
    print("- Recursive implementation can cause stack
overflow for very large arrays")
```

Output :
Original array: [64, 34, 25, 12, 22, 11, 90, 5]
QuickSorted array: [5, 11, 12, 22, 25, 34, 64, 90]
--- QuickSort vs BubbleSort Comparison ---
QuickSort Advantages:
- Much faster for large datasets (O(n log n) vs O(n²))
- More cache-efficient due to good locality of reference
- In-place sorting with low memory overhead

QuickSort Disadvantages:
- Worst-case performance can be O(n²)
- Not stable (may change order of equal elements)
- Recursive implementation can cause stack overflow for very large arrays

Bubble Sort –

```
def bubblesort(arr):
    """
    BubbleSort Algorithm Implementation

    BubbleSort is a simple comparison-based sorting algorithm
that works by:
```

```
    1. Repeatedly stepping through the list
    2. Comparing adjacent elements and swapping them if they
are in the wrong order
    3. The pass through the list is repeated until no swaps
are needed

    Time Complexity:
    - Best case: O(n) - when array is already sorted
    - Average/Worst case: O(n²)

    Space Complexity: O(1) - only requires a constant amount
of additional space

    Characteristics:
    - In-place sorting (modifies the original array)
    - Stable sort (preserves relative order of equal
elements)
    - Simple to understand and implement
    - Inefficient for large datasets
    """

    n = len(arr)

    # Traverse through all array elements
    for i in range(n):
        # Flag to optimize - if no swapping occurs, array is
sorted
        swapped = False

        # Last i elements are already in place
        for j in range(0, n - i - 1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the
next element
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # If no two elements were swapped in inner loop, then
array is sorted
        if not swapped:
            break

    return arr
```

```python
def bubblesort_optimized(arr):
    """
    Optimized version of BubbleSort with early termination
    This version stops early if the array becomes sorted
    """
    n = len(arr)

    for i in range(n):
        swapped = False
        # Reduce the range by i each iteration since the last
i elements are sorted
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        # Early termination if no swaps occurred
        if not swapped:
            break

    return arr


# Example usage and testing
if __name__ == "__main__":
    # Test the bubblesort algorithm
    test_array = [64, 34, 25, 12, 22, 11, 90, 5]
    print("Original array:", test_array)

    sorted_array = bubblesort(test_array.copy())
    print("BubbleSorted array:", sorted_array)

    # Test optimized version
    test_array2 = [1, 2, 3, 4, 5]  # Already sorted
    print("\nTesting optimized version on already sorted
array:", test_array2)
    optimized_sorted =
bubblesort_optimized(test_array2.copy())
    print("Optimized BubbleSort result:", optimized_sorted)

    # Algorithm comparison
    print("\n--- BubbleSort Characteristics ---")
    print("BubbleSort Advantages:")
```

```
    print("- Simple to understand and implement")
    print("- Stable sorting algorithm")
    print("- In-place sorting with O(1) space complexity")
    print("- Good for small datasets or nearly sorted
arrays")

    print("\nBubbleSort Disadvantages:")
    print("- Very slow for large datasets (O(n²) time
complexity)")
    print("- Poor cache performance")
    print("- Many unnecessary comparisons even in best case")

    print("\n--- When to use BubbleSort ---")
    print("- Educational purposes (teaching sorting
concepts)")
    print("- Very small datasets (n < 10)")
    print("- When simplicity is more important than
performance")
    print("- When stability is required and dataset is
small")
```

Output :

```
Original array: [64, 34, 25, 12, 22, 11, 90, 5]
BubbleSorted array: [5, 11, 12, 22, 25, 34, 64, 90]

Testing optimized version on already sorted array: [1, 2, 3, 4, 5]
Optimized BubbleSort result: [1, 2, 3, 4, 5]

--- BubbleSort Characteristics ---
BubbleSort Advantages:
- Simple to understand and implement
- Stable sorting algorithm
- In-place sorting with O(1) space complexity
- Good for small datasets or nearly sorted arrays

BubbleSort Disadvantages:
- Very slow for large datasets (O(n²) time complexity)
- Poor cache performance
- Many unnecessary comparisons even in best case

--- When to use BubbleSort ---
- Educational purposes (teaching sorting concepts)
```

- Very small datasets (n < 10)
- When simplicity is more important than performance
- When stability is required and dataset is small

Task Description #5 (Transparency in AI Recommendations)
Task: Use AI to create a product recommendation system.
Prompt:
"Generate a recommendation system that also provides reasons for each suggestion."
Expected Output:

- Code with explainable recommendations.
- Evaluation of whether explanations are understandable.

Code :

```
import sys
import subprocess

def check_dependencies():
    """Check if required dependencies are installed"""
    dependencies = ['numpy', 'pandas', 'scikit-learn',
'scipy']
    missing = []

    for dep in dependencies:
        try:
            __import__(dep)
        except ImportError:
            missing.append(dep)

    return missing

def run_simplified_system():
    """Run the simplified recommendation system"""
    print("=" * 70)
    print("RUNNING SIMPLIFIED RECOMMENDATION SYSTEM")
    print("=" * 70)
    print("This version uses only built-in Python libraries")
    print("-" * 70)

    # Import and run the simplified system
    from simplified_recommendation_system import main as
simplified_main
    simplified_main()
```

```
def run_full_system():
    """Run the full recommendation system if dependencies are
available"""
    print("\n" + "=" * 70)
    print("RUNNING FULL RECOMMENDATION SYSTEM")
    print("=" * 70)
    print("This version uses numpy, pandas, and scikit-
learn")
    print("-" * 70)

    try:
        # Try to import the full system
        from recommendation_system import
ExplainableRecommendationSystem
        from data_generator import generate_data

        # Generate data and run the system
        data = generate_data(num_users=30, num_items=20)
        rec_system = ExplainableRecommendationSystem()
        rec_system.fit(data)

        # Test with a user
        recommendations = rec_system.recommend_for_user(0,
top_n=3)

        print("Recommendations for User 0:")
        print("-" * 40)
        for i, rec in enumerate(recommendations, 1):
            print(f"{i}. Item {rec['item_id']} (score:
{rec['score']:.2f})")
            print("   Explanation:")
            for explanation in rec['explanation']:
                print(f"   - {explanation}")
            print()

        # Evaluate explanations
        evaluation =
rec_system.evaluate_explanation_quality(0, recommendations)
        print(f"Explanation Quality Score:
{evaluation['average_quality_score']:.2f}")
        print(f"Feedback: {evaluation['overall_feedback']}")

    except ImportError as e:
        print(f"Could not run full system: {e}")
```

```
        print("Please install the required dependencies:")
        print("pip install numpy pandas scikit-learn scipy")

def main():
    """Main demonstration function"""
    print("EXPLAINABLE RECOMMENDATION SYSTEM DEMONSTRATION")
    print("=" * 70)

    # Check dependencies
    missing_deps = check_dependencies()

    if missing_deps:
        print(f"Missing dependencies: {',
'.join(missing_deps)}")
        print("Running simplified version only...")
        run_simplified_system()
    else:
        print("All dependencies available!")
        print("Running both simplified and full versions...")
        run_simplified_system()
        run_full_system()
```

Output :

Generating sample data...
Generated 100 interactions

Training recommendation system...

Generating recommendations for 3 users...

Recommendations for User 0:
---
1. Item 10 (score: 4.09)
   Explanation:
   - Similar user 9 rated this item 4/5 (similarity: 0.52)
   - Similar user 11 rated this item 5/5 (similarity: 0.40)

2. Item 14 (score: 1.27)
   Explanation:
   - This item is popular among users with similar preferences

3. Item 11 (score: 1.21)

Explanation:
- This item is popular among users with similar preferences

Recommendations for User 1:
---
1. Item 9 (score: 2.30)
Explanation:
- Similar user 19 rated this item 4/5 (similarity: 0.58)

Recommendations for User 2:
---
1. Item 10 (score: 2.84)
Explanation:
- Similar user 16 rated this item 4/5 (similarity: 0.38)
- Similar user 14 rated this item 5/5 (similarity: 0.26)

2. Item 5 (score: 2.49)
Explanation:
- Similar user 17 rated this item 4/5 (similarity: 0.34)
- Similar user 10 rated this item 4/5 (similarity: 0.29)

3. Item 6 (score: 1.30)
Explanation:
- Similar user 14 rated this item 5/5 (similarity: 0.26)

==========
EXPLANATION QUALITY EVALUATION
==========
Average Quality Score: 0.74
Feedback: Excellent explanations - clear and specific

Task Description #6 (Transparent Code Generation)
Task: Ask AI to generate a Python function for calculating factorial using recursion.
Prompt:
"Generate a recursive factorial function with comments that explain each line and a final summary of the algorithm's flow."
Expected Output:

- Fully commented code.
- Clear documentation of how recursion works.

Code :

```
def factorial(n):
    """
    Calculate the factorial of a non-negative integer using recursion.

    Factorial of n (n!) is the product of all positive integers from 1 to n.
    By definition: 0! = 1

    Args:
        n (int): A non-negative integer

    Returns:
        int: The factorial of n

    Raises:
        ValueError: If n is negative
        TypeError: If n is not an integer
    """

    # Base case 1: Check if input is an integer
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")

    # Base case 2: Check for negative numbers
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    # Base case 3: Factorial of 0 is 1 (termination condition)
    if n == 0:
        return 1
```

```
    # Recursive case: n! = n * (n-1)!
    # The function calls itself with a smaller value (n-1)
    # This creates a chain of recursive calls until we reach
the base case
    return n * factorial(n - 1)


# Example usage and demonstration
if __name__ == "__main__":
    # Test the factorial function with various inputs
    test_values = [0, 1, 5, 7, 10]

    print("Factorial Calculation Examples:")
    print("=" * 30)

    for num in test_values:
        result = factorial(num)
        print(f"{num}! = {result}")

    print("\n" + "=" * 30)
    print("ALGORITHM FLOW SUMMARY:")
    print("=" * 30)
    print("""
1. INPUT VALIDATION:
   - First checks if input is an integer
   - Then verifies it's non-negative
   - These are the first base cases that prevent invalid
operations

2. TERMINATION CONDITION:
   - When n == 0, returns 1 (0! = 1 by definition)
   - This stops the recursion and starts the unwinding
process

3. RECURSIVE PROCESS:
   - For n > 0: n! = n * (n-1)!
   - Each recursive call reduces the problem size by 1
   - Creates a call stack: factorial(5) → factorial(4) →
factorial(3) → etc.

4. UNWINDING PHASE:
   - Once base case is reached (n=0), the calls start
returning
   - Each function returns its result to the previous caller
   - The results are multiplied in reverse order of the calls
```

```
Example for factorial(3):
- factorial(3) calls factorial(2)
- factorial(2) calls factorial(1)
- factorial(1) calls factorial(0)
- factorial(0) returns 1
- factorial(1) returns 1 * 1 = 1
- factorial(2) returns 2 * 1 = 2
- factorial(3) returns 3 * 2 = 6

Time Complexity: O(n) - linear time
Space Complexity: O(n) - due to call stack (each recursive
call adds to stack)
""")
```

Output :

Factorial Calculation Examples:

==========

0! = 1

1! = 1

5! = 120

7! = 5040

10! = 3628800

==========

ALGORITHM FLOW SUMMARY:

==========

1. INPUT VALIDATION:
   - First checks if input is an integer
   - Then verifies it's non-negative
   - These are the first base cases that prevent invalid operations

2. TERMINATION CONDITION:
   - When n == 0, returns 1 (0! = 1 by definition)
   - This stops the recursion and starts the unwinding process

3. RECURSIVE PROCESS:
   - For n > 0: n! = n * (n-1)!
   - Each recursive call reduces the problem size by 1
   - Creates a call stack: factorial(5) → factorial(4) → factorial(3) → etc.

4. UNWINDING PHASE:
- Once base case is reached (n=0), the calls start returning
- Each function returns its result to the previous caller
- The results are multiplied in reverse order of the calls

Example for factorial(3):
- factorial(3) calls factorial(2)
- factorial(2) calls factorial(1)
- factorial(1) calls factorial(0)
- factorial(0) returns 1
- factorial(1) returns 1 * 1 = 1
- factorial(2) returns 2 * 1 = 2
- factorial(3) returns 3 * 2 = 6

Time Complexity: O(n) - linear time
Space Complexity: O(n) - due to call stack (each recursive call adds to stack

Task Description #7 (Inclusiveness in Customer Support)
Code Snippet:

```
def support_reply(name, gender):
    if gender.lower() == "male":
        prefix = "Mr."
    else:
        prefix = "Mrs."
    return f"Dear {prefix} {name}, we have resolved your issue."
```

Task:
Regenerate the code so that support messages use neutral language (e.g., “Dear {name}”) and optionally accept preferred titles.
Expected Output:
- Neutral, user-friendly support responses.

Code :

```
def support_reply(name, title=None):
    if title:
        prefix = title
    else:
        prefix = "Dear"
```

```
    return f"{prefix} {name}, we have resolved your issue."

def main():
    from support import support_reply

    name = input("Enter your name: ")
    title = input("Enter your title : ")

    response = support_reply(name, title)
    print(response)

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

Output :
Enter your name: Bob
Enter your gender (male/female): male
Dear Mr. Bob, we have resolved your issue.

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
Transparency	1
Inclusiveness	0.5
Data security and Privacy	1
Total	2.5 Marks