Department of Computer Engineering
University of Peradeniya
CO523- Programming Languages
Lab 06: Expressions, Control Structures, and Exception Handling

E/20/449
Wijewardhana A.R.S.S.

Task 01

Code :

```python
def factorial_recursive(n):
    """Calculates factorial using recursion."""
    # Base Case: factorial of 0 or 1 is 1
    if n == 0 or n == 1:
        return 1
    # Recursive Case: n * factorial of (n-1)
    return n * factorial_recursive(n - 1)

def sum_of_digits_loop(n):
    """Calculates sum of digits using a loop."""
    n = abs(n)  # Convert to positive to handle digits correctly
    total_sum = 0
    while n > 0:
        total_sum += n % 10  # Get the last digit
        n //= 10             # Remove the last digit
    return total_sum

def main():
    try:
        # Taking integer input
        user_input = input("Enter an integer: ")
        num = int(user_input)

        if num > 0:
            # If positive, find factorial using recursion
            result = factorial_recursive(num)
            print(f"The number is positive. Factorial of {num} is: {result}")

        elif num < 0:
            # If negative, find sum of digits using a loop
            result = sum_of_digits_loop(num)
```

```
            print(f"The number is negative. Sum of its digits is: {result}")

        else:
            # If zero
            print("The number is zero.")

    except ValueError:
        # Exception handling for invalid (non-integer) input
        print("Error: Invalid input. Please enter a valid integer.")

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

Output :

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Taks01.py"
Enter an integer: 4
The number is positive. Factorial of 4 is: 24
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Taks01.py"
Enter an integer: -4
The number is negative. Sum of its digits is: 4
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Taks01.py"
Enter an integer: 0
The number is zero.
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Taks01.py"
Enter an integer: s
Error: Invalid input. Please enter a valid integer.
PS D:\Sem7\CO523-programming Languages>
```

Explanation :

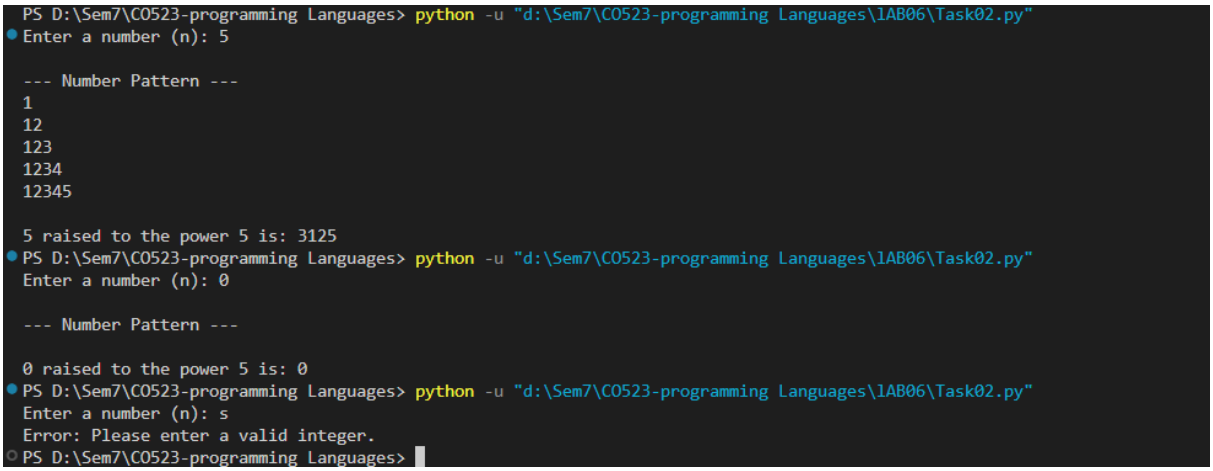
- **Recursion (Factorial):** As defined in Section 4, the factorial_recursive function uses a **base case** (n=0 or n=1) to stop the execution and a **recursive case** where the function calls itself with a modified argument (n-1).
- **Loop (Sum of Digits):** For negative numbers, the program uses a while loop. It uses the modulo operator (%) to extract digits and integer division (//) to reduce the number until the condition n > 0 becomes false.
- **Exception Handling:** The code is wrapped in a try-except block. If a user enters a string or a decimal (which cannot be converted via int()), a ValueError is triggered, and the except block handles it gracefully instead of crashing the program.
- **Conditional Structures:** An if-elif-else structure is used to decide which logic to execute based on whether the input is positive, negative, or zero.

Task 02

Code :

```python
def power_recursive(base, exponent):
    """Calculates base raised to the power of exponent using recursion."""
    # Base Case: Any number raised to the power of 0 is 1
    if exponent == 0:
        return 1
    # Recursive Case: base * (base ^ (exponent - 1))
    return base * power_recursive(base, exponent - 1)

def main():
    try:
        # Taking number n as input
        n = int(input("Enter a number (n): "))

        print("\n--- Number Pattern ---")
        # Outer loop for rows from 1 up to n
        for i in range(1, n + 1):
            # Inner loop for numbers in each row
            for j in range(1, i + 1):
                print(j, end="")
            # Move to next line after inner loop completes
            print()

        # Calculate n^5 using recursion
        power_result = power_recursive(n, 5)
        print(f"\n{n} raised to the power 5 is: {power_result}")

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

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

Output :



Explanation :

- **Nested Loops for Patterns**:
  - The **outer loop** (for i in range(1, n + 1)) controls the number of rows to be printed.
  - The **inner loop** (for j in range(1, i + 1)) iterates to print numbers from 1 up to the current row number on the same line.
  - The print() statement after the inner loop ensures the program moves to a new line to form the triangular shape.
- **Recursion for Power Calculation**:
  - **Base Case**: The recursion stops when the exponent reaches 0, returning 1.
  - **Recursive Case**: The function calls itself with a reduced exponent (exponent - 1), breaking the problem into smaller pieces.
- **Execution Flow**:
  - Loops are generally faster and more memory-efficient for simple repetitive tasks like the pattern.
  - Recursion provides an elegant solution for the power calculation but consumes more memory on the call stack for each iteration.

Task 03

Code :

```
import sys

def factorial_loop(n):
    """Calculates factorial using a for loop."""
    res = 1
    for i in range(1, n + 1):
        res *= i
    return res

def factorial_recursion(n):
    """Calculates factorial using recursion."""
    if n == 0 or n == 1: # Base case [cite: 173, 197]
        return 1
    return n * factorial_recursion(n - 1) # Recursive case [cite: 174, 199]

def reverse_loop(n):
    """Reverses a number using a while loop."""
    rev = 0
    n = abs(n)
    while n > 0:
        rev = (rev * 10) + (n % 10)
        n //= 10
    return rev

def reverse_recursion(n, rev=0):
    """Reverses a number using recursion."""
    if n == 0: # Base case
        return rev
    return reverse_recursion(n // 10, rev * 10 + n % 10) # Recursive case

def main():
    while True: # Infinite loop for menu repetition [cite: 106, 107]
        print("\n--- Menu ---")
        print("1. Find factorial (loop)")
        print("2. Find factorial (recursion)")
        print("3. Reverse a number (loop)")
        print("4. Reverse a number (recursion)")
        print("5. Exit")

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

```
            if choice == '5':
                print("Exiting program...")
                break # Break statement to exit loop [cite: 161]

            if choice not in ['1', '2', '3', '4']:
                print("Invalid choice! Please select 1-5.")
                continue # Skip to next iteration [cite: 150]

            num = int(input("Enter an integer: "))

            if choice == '1':
                print(f"Factorial (loop): {factorial_loop(num)}")
            elif choice == '2':
                if num < 0:
                    print("Factorial not defined for negative numbers.")
                else:
                    print(f"Factorial (recursion): {factorial_recursion(num)}")
            elif choice == '3':
                print(f"Reversed (loop): {reverse_loop(num)}")
            elif choice == '4':
                print(f"Reversed (recursion): {reverse_recursion(abs(num))}")

        except ValueError as e:
            # Handling invalid input types [cite: 241, 242]
            print(f"Error: Invalid input. {e}")

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

Output :

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Task03.py"

--- Menu ---
1. Find factorial (loop)
2. Find factorial (recursion)
3. Reverse a number (loop)
4. Reverse a number (recursion)
5. Exit
Enter your choice (1-5): 1
Enter an integer: 10
Factorial (loop): 3628800

--- Menu ---
1. Find factorial (loop)
2. Find factorial (recursion)
3. Reverse a number (loop)
4. Reverse a number (recursion)
5. Exit
Enter your choice (1-5): 2
Enter an integer: 10
Factorial (recursion): 3628800

--- Menu ---
1. Find factorial (loop)
2. Find factorial (recursion)
3. Reverse a number (loop)
4. Reverse a number (recursion)
5. Exit
Enter your choice (1-5): 3
Enter an integer: 10
Reversed (loop): 1

--- Menu ---
1. Find factorial (loop)
2. Find factorial (recursion)
3. Reverse a number (loop)
4. Reverse a number (recursion)
5. Exit
Enter your choice (1-5): 4
Enter an integer: 10
Reversed (recursion): 1

--- Menu ---
1. Find factorial (loop)
2. Find factorial (recursion)
3. Reverse a number (loop)
4. Reverse a number (recursion)
5. Exit
Enter your choice (1-5): 5
Exiting program...
PS D:\Sem7\CO523-programming Languages>
```

Explanation :

- Menu Selection: Uses an if-elif-else structure to direct the program flow based on user input.
- Repetition: A while (True) loop is used to keep the menu active until the user explicitly chooses to exit.
- Loop Control:
    - The break statement is used to terminate the menu loop when choice '5' is selected.
    - The continue statement is used to skip the rest of the loop if an invalid menu choice is made.
- Recursion vs. Iteration:
    - Iteration (Loops): Generally faster and uses constant memory.
    - Recursion: Breaks problems into smaller instances. It requires a base case to prevent stack overflow.
- Exception Handling: The entire input process is wrapped in a try-except block to catch ValueError if the user enters non-integer data.

Task 04

Code :

```
def main():
    print("--- Simple Calculator ---")

    while True: # Loop to continue until user exits
        try:
            # Menu for operations
            print("\nOptions: +, -, *, /, or 'exit' to quit")
            operation = input("Enter operation: ").strip().lower()

            if operation == 'exit':
                print("Exiting calculator. Goodbye!")
                break # Exit the loop [cite: 161, 287]

            if operation not in ['+', '-', '*', '/']:
                print("Invalid operation! Please choose +, -, *, or /.")
                continue # Skip to next iteration [cite: 150]

            # Taking numerical inputs
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))

            # Expressions for calculations [cite: 14, 293]
            if operation == '+':
                result = num1 + num2
                print(f"Result: {num1} + {num2} = {result}")
            elif operation == '-':
                result = num1 - num2
                print(f"Result: {num1} - {num2} = {result}")
            elif operation == '*':
                result = num1 * num2
                print(f"Result: {num1} * {num2} = {result}")
            elif operation == '/':
                # The try-except block will catch division by zero
                result = num1 / num2
                print(f"Result: {num1} / {num2} = {result}")

        except ZeroDivisionError:
            # Handling division by zero [cite: 241, 292]
            print("Error: Cannot divide by zero.")
        except ValueError:
            # Handling invalid input (e.g., entering letters instead of numbers)
[cite: 241, 292]
```

```
            print("Error: Invalid input. Please enter numeric values.")

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

Output :

```
PS D:\Sem7\CO523-programming Languages> python -u "d:\Sem7\CO523-programming Languages\lAB06\Task04.py"
--- Simple Calculator ---

Options: +, -, *, /, or 'exit' to quit
Enter operation: 2+3
Invalid operation! Please choose +, -, *, or /.

Options: +, -, *, /, or 'exit' to quit
Enter operation: +
Enter first number: 4
Enter second number: 5
Result: 4.0 + 5.0 = 9.0

Options: +, -, *, /, or 'exit' to quit
Enter operation: /
Enter first number: 9
Enter second number: 0
Error: Cannot divide by zero.

Options: +, -, *, /, or 'exit' to quit
Enter operation: s
Invalid operation! Please choose +, -, *, or /.

Options: +, -, *, /, or 'exit' to quit
Enter operation: exit
Exiting calculator. Goodbye!
PS D:\Sem7\CO523-programming Languages>
```

Explanation :

- **Expressions**: The program uses arithmetic expressions (e.g num1 + num2, num1 / num2) to produce calculation results. These are the basic building blocks used to calculate the results requested in the task.
- **Looping for Continuity**: A while (True) loop is implemented to ensure the program keeps running until the user explicitly types 'exit'. The break statement is used to exit the loop prematurely when the exit condition is met.
- **Exception Handling**:
    - **ZeroDivisionError**: If the user attempts to divide by zero, the program catches this specific exception to prevent a crash.
    - **ValueError**: If the user inputs non-numeric characters when a number is expected, the ValueError block provides a clear error message instead of failing.
- **Control Flow**: The if-elif-else structure acts as the logic gate to determine which arithmetic operation to perform based on user selection.

Task 05

Explanation :

- Recursion requires two main components: a **Base Case** and a **Recursive Case**. **The Base Case** acts as a stopping condition.
- If a recursive function lacks this condition, or if the condition is never met, the function will call itself indefinitely.
- Each recursive call adds a new layer to the **call stack**, consuming memory.
- Eventually, the program will exhaust the available stack space, leading to a **Stack Overflow** error.

```python
def infinite_recursion():
    """A recursive function with no base case."""
    print("Calling function...")
    # Recursive Case: Function calls itself without a stopping condition
    return infinite_recursion()

# Executing the function
try:
    infinite_recursion()
except RecursionError as e:
    print(f"\nResult: {e}")
```

```
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...
Calling function...

Result: maximum recursion depth exceeded while calling a Python object
PS D:\Sem7\CO523-programming Languages>
```

Result:

- The function infinite_recursion() is called and immediately calls itself again.
- Because there is no **base case** (stopping condition) to exit the loop, the stack continues to grow with each call.
- Python has a built-in limit for the maximum depth of the call stack to protect against system crashes.
- Once this limit is reached, the program terminates and throws a **RecursionError: maximum recursion depth exceeded**.