EXPERIMENT NO:03

Title: Implement a python program to demonstrate the use of following conditional statements: if, if…else, if…elif…else and nested if statements

Prior Concepts: Basic Python syntax, how to define variables and use operators (arithmetic, comparison), understanding of logic and flow control in programming and knowledge of indentation and block structure in Python.

Objective: To implement and understand the usage of conditional statements in Python, specifically:

if, if..else, if..elif..else, Nested if statements

Theory:

Conditional statements allow the flow of a Python program to change based on whether a condition evaluates to True or False.

1. **if Statement**: The simplest form of decision-making statement. It evaluates a condition and executes the block of code only if the condition is true.

   Syntax:

   if condition:

   # code block to execute if condition is True

Example:

```
#Check if a number is positive.
     number = 10
     if number > 0:
          print(f"{number} is positive.")
```

2. **if..else Statement**: Allows two possibilities—if the condition is True, one block is executed; if False, the other block is executed.

   Syntax:

   if condition:

   # code block to execute if condition is True

   else:

   # code block to execute if condition is False

```
Example:
#Check if a number is positive or negative.
number = -5
     if number > 0:
         print(f"{number} is positive.")
     else:
         print(f"{number} is negative.")
```

3. **if..elif..else Statement**: Extends if..else to allow multiple conditions. If the first condition is False, the next elif condition is checked, and so on.

   Syntax:

   if condition1:

      # code block to execute if condition1 is True

   elif condition2:

      # code block to execute if condition2 is True

   else:

      # code block to execute if none of the conditions is True

```
Example:
#Classify a number as positive, negative, or zero.
     number = 0
     if number > 0:
         print(f"{number} is positive.")
     elif number < 0:
         print(f"{number} is negative.")
     else:
         print("The number is zero.")
```

4. **Nested if Statements**: When if statements are placed inside another if or else block, creating more complex decision-making logic.

Syntax:

if condition1:

   if condition2:

      # code block to execute if both condition1 and condition2 are True

Example:

```
#Check if a person is eligible to vote based on their age and citizenship.
   age = 20
   citizenship = 'USA'
   if age >= 18:
       if citizenship == 'USA':
           print("You are eligible to vote.")
       else:
           print("You are not eligible to vote in the USA.")
   else:
       print("You are not eligible to vote due to age.")
```

Procedure

1. **Step 1**: Open your Python development environment (IDE) such as IDLE, VSCode, or Jupyter Notebook.
2. **Step 2**: Create a new Python file.
3. **Step 3**: Write Python programs for each type of conditional statement.
4. **Step 4**: Run each program and observe the output to understand how each condition is evaluated.

Conclusion: Thus, we have learned the use and implementation of conditional statements in Python.

Questions

1. Write a Python program to calculate the bonus for an employee based on the following conditions:

- If the employee has more than 10 years of experience, they receive a 20% bonus on their salary.
- If the employee has between 5 and 10 years of experience, they receive a 10% bonus.
- If the employee has less than 5 years of experience, they receive a 5% bonus.
- If the employee's performance rating is below 3 (on a scale of 1 to 5), they are not eligible for a bonus, regardless of experience.

  Input:

  Salary (integer)

  Years of experience (integer)

Performance rating (float)

Expected Output: The bonus amount and final salary after adding the bonus.

2. Write a program that takes the lengths of three sides of a triangle and determines the type of triangle:

   If all sides are equal, it is an **Equilateral Triangle**.

   If two sides are equal, it is an **Isosceles Triangle**.

   If no sides are equal, it is a **Scalene Triangle**.

   If the sum of any two sides is not greater than the third side, it is **not a triangle**.

   Input: Three positive integers representing the sides of a triangle.

   Expected Output: The type of triangle or a message saying "Not a triangle."

3. Write a program to determine if a given year is a leap year. However, add the following extra conditions:

   A year is a leap year if it is divisible by 4 but not divisible by 100 unless it is also divisible by 400.

   If the year is in the 21st century (from 2001 to 2100), print a message indicating whether it is a "21st Century Leap Year" or not.

   If the year is before the 21st century or after it, print a simple message whether it's a leap year or not.

   Input: A single integer representing the year.

   Expected Output: A message indicating whether the year is a leap year or not, with the extra condition for the 21st century.

EXPERIMENT NO: 04

Title: Implement a python program to demonstrate the use of following looping statements: while and for loop, nested loops

Prior Concepts: Basic syntax of Python, Variables and basic data types in Python, Knowledge of indentation and code blocks in Python, Conditional statements (e.g., if, elif, else).

Objective: To demonstrate and understand the usage of: while loop, for loop, nested loops.

Theory:

1. while Loop: A while loop repeatedly executes a block of code as long as a specified condition is True. The loop may run indefinitely if the condition never becomes False.

Syntax:

```
while condition:
    # Code to execute
```

2. for Loop: A for loop iterates over a sequence (such as a list, string, or range) and executes a block of code for each item in the sequence.

Syntax:

```
for item in sequence:
    # Code to execute
```

3. Nested Loops: A loop inside another loop is known as a nested loop. The inner loop runs completely every time the outer loop runs once.

Syntax:

```
for item1 in sequence1:
    for item2 in sequence2:
        # Code to execute
```

Procedure

Step 1: Open a Python IDE like IDLE, VS Code, or Jupyter Notebook.

Step 2: Create a Python file or open a new Python script.

Step 3: Implement a while loop to print numbers from 1 to 5.

```
# Example: Using a while loop to print numbers from 1 to 5
number = 1
while number <= 5:
    print(f"While Loop: {number}")
    number += 1
```

Step 4: Implement a for loop to iterate through a list of strings and print each item.

```
# Example: Using a for loop to print elements of a list
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(f"For Loop: {fruit}")
```

Step 5: Implement a nested for loop to print a pattern (e.g., a multiplication table).

```
# Example: Nested for loop to print multiplication table
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i} x {j} = {i * j}", end='\t')
    print()  # New line after each inner loop iteration
```

Step 6: Implement a nested while loop to print a right-angle triangle pattern.

```
# Example: Nested while loop to print a right-angle triangle
rows = 5
i = 1
while i <= rows:
    j = 1
    while j <= i:
        print('*', end=' ')
```

```
        j += 1
    print()  # New line after each row
    i += 1
```

Step 7: Execute each program to observe the outputs and understand how the loops work.

Conclusion: Hence, we have successfully implemented and learned about different types of loops in Python (while, for, and nested loops).

Questions:

1. Write a Python program that generates Pascal's Triangle up to a specified number of rows using nested loops. Each number in the triangle is the sum of the two directly above it.

   Input: An integer representing the number of rows in Pascal's triangle.

   Expected Output: For n = 5, the output should be:

   1

   1 1

   1 2 1

   1 3 3 1

   1 4 6 4 1

2. Write a Python program using a `while` loop to find all prime numbers between two given numbers `a` and `b` (inclusive). A prime number is a number that is divisible only by 1 and itself.
   Input: Two integers `a` and `b`.
   Expected Output: A list of prime numbers in the range `[a, b]`.

3. Write a Python program using a for loop to calculate the sum of all even numbers from 1 to 100.