PES UNIVERSITY, BENGALURU

Perseverance | Excellence | Service

UE25CS151A – PYTHON FOR COMPUTATIONAL PROBLEM SOLVING LAB MANUAL

WEEK 4

TOPICS:

Programs on Control Strutures in Python

OBEJCTIVE:

- Utilize **conditional statements (if, elif, else)** to implement decision-making logic and control program flow based on dynamic conditions or user inputs.
- Apply **for and while loops** to execute repetitive tasks and process data iteratively, enabling efficient solutions for problems like sequence generation and pattern creation.
- Integrate **branching and looping constructs** to develop robust Python programs that solve complex problems, such as validating inputs, generating patterns, or performing iterative calculations.

Problem Statement 1:

Given an integer n, print numbers from 1 to n.

However, for multiples of three, print "Fizz" instead of the number.

For multiples of five, print "Buzz".

For numbers which are multiples of both three and five, print "FizzBuzz".

```
Enter an integer n: 15
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Problem Statement 2:

Given an integer, reverse its digits. The sign of the number should be preserved.

Example:

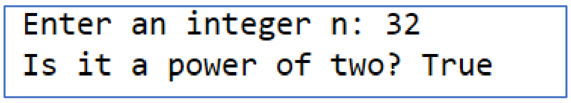
- Input: -123
- Output: -321

Problem Statement 3:

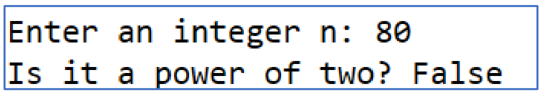
Given an integer n, determine if it is a power of two. An integer is a power of two if there exists an integer x such that n == 2^x.

Example:

- Input: 16
- Output: True (Because 16 = 2^4)
- Input: 18
- Output: False



```
Enter an integer n: 32
Is it a power of two? True
```



```
Enter an integer n: 80
Is it a power of two? False
```

Problem Statement 4:

Given a positive integer num, write a program that returns True if num is a perfect square, and False otherwise without using built in functions.(math.sqrt())

Examples:

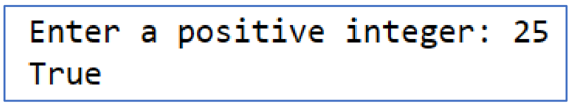
- Input: 16
- Output: True (since 4 * 4 = 16)
- Input: 14
- Output: False

A perfect square is a whole number that is the result of multiplying another whole number by itself

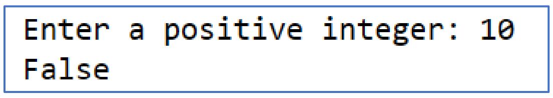
The simplest way to think about it is through multiplication.

- 9 is a perfect square because 3 × 3 = 9.
- 16 is a perfect square because 4 × 4 = 16.
- 25 is a perfect square because 5 × 5 = 25.
- 1 is a perfect square because 1 × 1 = 1.

A number like 10 is not a perfect square because you can't multiply any single whole number by itself to get 10.



```
Enter a positive integer: 25
True
```



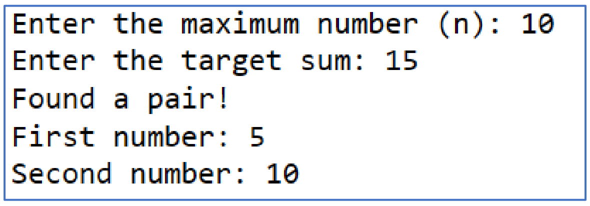
```
Enter a positive integer: 10
False
```

Problem Statement 5:

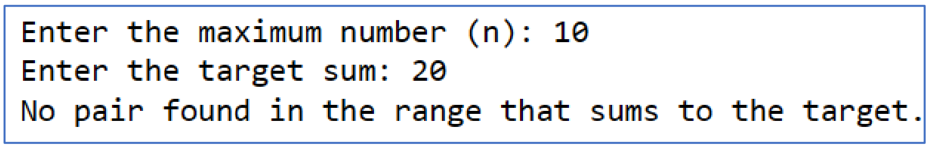
Given a maximum number n and a target number, find the first pair of two different integers between 1 and n (inclusive) that add up to the target.

Example:

- Input: n = 10, target = 12
- Output: First number: 2, Second number: 10
- Reason: The first pair the program finds is 2 + 10, which equals 12. (Although 3+9, 4+8, and 5+7 also work, the program stops after finding the first one).



```
Enter the maximum number (n): 10
Enter the target sum: 15
Found a pair!
First number: 5
Second number: 10
```



```
Enter the maximum number (n): 10
Enter the target sum: 20
No pair found in the range that sums to the target.
```

Practice Programs:

1. Number Guessing Game

Objective: Create a simple game where the user tries to guess a secret number.

Description: Hard-code a "secret number" in your program (e.g., secret_number = 42). Use a while loop to repeatedly ask the user to guess the number. Inside the loop, tell the user if their guess is "Too high" or "Too low". The loop should continue until the user guesses correctly, at which point you print "Congratulations!" and the loop ends.

Example:

(Assuming secret_number is 42)

Input: 50 -> **Output:** Too high

Input: 30 -> **Output:** Too low

Input: 42 -> **Output:** Congratulations!

Concepts to Use: while loop, if-elif-else, break statement.

2. Print Patterns:

1)Right angled triangle

```
*
* *
* * *
* * * *
* * * * *
```

2) Pyramid pattern

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

3) Diamond pattern

```
        *
       ***
      *****
     *******
    *********
     *******
      *****
       ***
        *
```

4) Hollow square

```
* * * * *
*       *
*       *
*       *
* * * * *
```

5) Number pattern

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

3. **Armstrong Number Checker**
   Objective: Check if a number is an Armstrong number.
   Description: An Armstrong number (of order n) is a number that is equal to the sum of its own digits each raised to the power of the number of digits. For example, **153** is an Armstrong number because it has 3 digits, and $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.
   Task: Ask the user for an integer. First, you'll need a loop to count how many digits are in the number. Then, you'll need another loop to extract each digit and calculate the sum of the digits raised to the power of the digit count. Finally, compare the sum to the original number.

- **Example:**
  - **Input:** 153
  - **Output:** 153 is an Armstrong number.
  - **Input:** 120
  - **Output:** 120 is not an Armstrong number.

- **Concepts to Use:** while loops, modulo (%) and integer division (//) to manipulate digits.

Every solved problem adds a brick to your foundation of knowledge