

## PES UNIVERSITY, BENGALURU

Perseverance | Excellence | Service

# UE25CS151A – PYTHON FOR COMPUTATIONAL PROBLEM SOLVING LAB MANUAL

## WEEK 11

### TOPICS:

### Programs on Functions and Recursion

### OBEJCTIVE:

To gain hands-on experience in designing and implementing **user-defined functions** for modular and reusable code, and to explore the concept of **recursion** as a problem-solving technique for repetitive and divide-and-conquer tasks in Python programming.

### Problem Statement 1:

Write a Python program using a function that takes an integer n as input and returns a list named solution (1-indexed) such that:

- `solution[i] = "FizzBuzz"` if i is divisible by both 3 and 5.
- `solution[i] = "Fizz"` if i is divisible by 3.
- `solution[i] = "Buzz"` if i is divisible by 5.
- `solution[i] = string representation of i` if none of the above conditions are true.

### Expected Output:

#### **Example 1:**

Input: n = 3

Output: ["1", "2", "Fizz"]

#### **Example 2:**

Input: n = 5

Output: ["1", "2", "Fizz", "4", "Buzz"]

#### **Example 3:**

Input: n = 15

Output:

["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz"]

## Problem Statement 2:

Write a Python program using a **function** that takes a list of integers `nums` as input and moves all the **even integers** to the beginning of the list, followed by all the **odd integers**.

The function should return a new list that satisfies this condition.

Any valid order that places all even numbers before all odd numbers is acceptable.

### **Example 1:**

Input: `nums = [3, 1, 2, 4]`

Output: `[2, 4, 3, 1]`

Explanation: `[4, 2, 3, 1]`, `[2, 4, 1, 3]`, and `[4, 2, 1, 3]` would also be accepted.

### **Example 2:**

Input: `nums = [0, 1, 0, 3, 12, 3, 4]`

Output: `[0, 0, 12, 4, 1, 3, 3]`

## Problem Statement 3:

Write a Python program using a **function** that takes a sentence and a word as input and returns how many times that word appears in the sentence.

### **Function Specification:**

```
def count_word_occurrence(sentence, word):
    # returns number of times 'word' appears in 'sentence'
```

### **Example Input:**

Enter a sentence: "This is a test. This test is simple."

Enter the word to count: "test"

### **Expected Output:**

The word 'test' appears 2 times.

## Problem Statement 4:

**Write a Python function `isPalindrome` that takes a single string `s` as its parameter.**

- Your function must determine if the string is a palindrome.
- A phrase is a palindrome if it reads the same forwards and backward after it has been "cleaned".  
The "cleaning" process involves two steps:
  - Converting all uppercase letters into lowercase.
  - Removing all non-alphanumeric characters (i.e., keeping only letters and numbers).

- Your isPalindrome function should perform this cleaning process and then check if the cleaned string reads the same forwards and backward.
- Return True if it is a palindrome, or False otherwise.

### Expected output:

**Enter a string:** A man, a plan, a canal: Panama

**Cleaned String:** amanaplanacanalpanama

**Reversed String:** amanaplanacanalpanama

**Yes, it is a palindrome.**

### Problem Statement 5:

**The Fibonacci sequence is a famous mathematical sequence where each new number is the sum of the two preceding ones.**

The sequence starts with 0 and 1.

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n - 1) + F(n - 2), \text{ for } n > 1.$$

Write a recursive function fib(n) that takes an integer n and returns the nth Fibonacci number.

#### **Example 1:**

Input: n = 2

Output: 1

Explanation:  $\text{fib}(2) = \text{fib}(1) + \text{fib}(0) = 1 + 0 = 1.$

#### **Example 2:**

Input: n = 3

Output: 2

Explanation:  $\text{fib}(3) = \text{fib}(2) + \text{fib}(1) = 1 + 1 = 2.$

#### **Example 3:**

Input: n = 4

Output: 3

Explanation:  $\text{fib}(4) = \text{fib}(3) + \text{fib}(2) = 2 + 1 = 3.$

## Problem Statement 6:

Write a Python program using **recursion** to flatten a **nested list**, that is, convert a list that may contain other lists as elements into a single one-dimensional list containing all the values.

The program should use a function named `flatten_list()` that takes a nested list as input and returns a new flat list containing all elements in order.

### **Function Specification:**

```
def flatten_list(nested_list):  
    # returns a flat list containing all elements of nested_list
```

#### **Example1:**

Input: [1, [2, 3], 4]

Output: [1, 2, 3, 4]

#### **Example 2:**

Input: [1, [2, [3, 4]], 5, [6]]

Output: [1, 2, 3, 4, 5, 6]

---

The best code is not the one that works, but the one that's easy to understand.