Module 2

Q.1: Write a Python program to check if a number is positive, negative or zero.

Ans:
```
number = float(input(“Enter a number: “))


If number > 0:
  print(“The number is positive”)
elif number < 0:
  print(“The number is negative”)
else:
  print(“The number is zero”)
```

Q.2: Write a Python program to get the Factorial number of given number.

Ans:
```
def factorial_iterative(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

Function to calculate factorial recursively:

```
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)
```

#Get input from the user:

```
number = int(input("Enter a number: "))
```

#Calculate factorial using both methods:

```
factorial_iter = factorial_iterative(number)
factorial_rec = factorial_recursive(number)
```

#Print the result:

```
print(f"The factorial of {number} (calculated
iteratively) is: {factorial_iter}")
print(f"The factorial of {number} (calculated
recursively) is: {factorial_rec}")
```

Q.3: Write a Python program to get the Fibonacci series of given range.

Ans: #Function to generate Fibonacci series within a given range iteratively
```
def fibonacci_series_iterative(n):
    fib_series = []
    a, b = 0, 1
    while a < n:
        fib_series.append(a)
        a, b = b, a + b
    return fib_series
```

#Get input from the user
```
range_limit = int(input("Enter the upper limit
of the range: "))
```

#Generate Fibonacci series using the iterative function
```
fibonacci_series =
fibonacci_series_iterative(range_limit)
```

#Print the generated Fibonacci series
```
print("Fibonacci series within the given
range:")
print(fibonacci_series)
```

Q.4: How memory is managed in Python?

Ans: Memory in Python is managed primarily through a combination of techniques including automatic garbage collection, reference counting, and dynamic memory allocation.

1. **Reference Counting**: Python uses a reference counting mechanism to keep track of the number of references to an object in memory.
2. **Automatic Garbage Collection**: Python employs a cyclic garbage collector to identify and collect object that are no longer reachable, this helps in freeing up memory occupied by objects that have circular references or are otherwise not accessible from the main program.
3. **Memory Allocation and Deallocation**: Python's memory manager handles dynamic memory allocation and deallocation. It requests memory from the OS when needed and releases memory when objects are no longer in use.
4. **Memory Pools**: Python employs a memory pool allocator that manages blocks of memory of specific sizes. This helps in reducing memory fragmentation and improves the efficiency of memory allocation and deallocation.
5. **Optimization**: Python uses various optimizations to manage more effectively.

6. **Memory Views and Buffers**: Python provides memory views and buffer objects that allow direct access to the internal data of certain objects without making copies.
7. **Memory Profiling Tools**: Python offers various tools and libraries for memory profiling, which help developers identify memory leaks and areas of the code that may be consuming excessive memory.

Q.5: What is the purpose continue statement in python?

Ans: The “continue” statement in Python is used within loops to skip the current iteration of the loop and proceed to the next iteration.

I allows you to control the flow of the loop by skipping certain steps within the loop body based on a specific condition.

The primary purpose of “continue” statement is to provide a way to bypass certain code execution within the loop while still continuing the loop with the next iteration.

Q.6: Write python program that swap two number with temp variable and without temp variable.

Ans: #Using a Temporary Variable:

```
def swap_with_temp(a, b):
    temp = a
    a = b
    b = temp
    return a, b

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

print("Before swapping: num1 =", num1, "num2 =", num2)

num1, num2 = swap_with_temp(num1, num2)

print("After swapping: num1 =", num1, "num2 =", num2)
```

#Without Using a Temporary Variable

```
def swap_without_temp(a, b):
    a = a + b
    b = a - b
    a = a - b
    return a, b

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: 
"))

print("Before swapping: num1 =", num1, "num2 
=", num2)

num1, num2 = swap_without_temp(num1, num2)

print("After swapping: num1 =", num1, "num2 =", 
num2)
```

Q.7: Write a Python program to find whether a given number is even or odd, print out an appropriate message to the user.

Ans:

```
number = int(input("Enter a number: "))

if number % 2 == 0:
    print(f"{number} is an even number.")
else:
    print(f"{number} is an odd number.")
```

Q.8: Write a Python program to test whether a passed letter is a vowel or not.

Ans:

```
letter = input("Enter a letter: ")


letter = letter.lower()


if letter in ['a', 'e', 'i', 'o', 'u']:
    print(f"The letter '{letter}' is a vowel.")
else:
    print(f"The letter '{letter}' is not a
vowel.")
```

Q.9: Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.

Ans:

```
num1 = int(input("Enter the first integer: "))
num2 = int(input("Enter the second integer: "))
num3 = int(input("Enter the third integer: "))

 #Check for equal values
if num1 == num2 or num2 == num3 or num1 ==
num3:
    sum_result = 0
else:
    sum_result = num1 + num2 + num3
```

```
 #Print the result
print("Sum:", sum_result)
```

Q.10: Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.

Ans:

```
 #Function to check the conditions
def check_condition(a, b):
    return a == b or abs(a - b) == 5 or a + b
== 5

 #Get input from the user
num1 = int(input("Enter the first integer: "))
num2 = int(input("Enter the second integer: "))

 #Check the conditions using the function
result = check_condition(num1, num2)

 #Print the result
if result:
    print("True")
```

```
else:
    print("False")
```

Q.11: Write a python program to sum of the first n positive integers.

Ans:

```
 #Get input from the user
n = int(input("Enter a positive integer: "))

 #Validate the input
if n <= 0:
    print("Please enter a positive integer.")
else:
     #Calculate the sum of the first n positive
integers
    sum_result = (n * (n + 1)) // 2

     #Print the result
    print(f"The sum of the first {n} positive
integers is:", sum_result)
```

Q.12: Write a Python program to calculate the length of a string.

Ans:

```
 #Get input from the user
string = input("Enter a string: ")

 #Calculate the length of the string
length = len(string)

 #Print the result
print(f"The length of the string '{string}'
is:", length)
```

Q.13: Write a Python program to count the number of characters (character frequency) in a stringWrite a Python program to count the number of characters (character frequency) in a string.

Ans:
```
 #Get input from the user

string = input("Enter a string: ")


 #Create an empty dictionary to store character
frequencies

char_frequency = {}


 #Count the frequency of each character

for char in string:
    if char in char_frequency:
        char_frequency[char] += 1
    else:
        char_frequency[char] = 1

 #Print the character frequencies
print("Character frequencies:")
for char, freq in char_frequency.items():
    print(f"'{char}': {freq}")
```

Q.14: What are negative indexes and why are they used?

Ans: Negative indexes in programming, including in Python, are a way to access elements in a sequence (such as a list, string, or tuple) starting from the end of the sequence rather than the beginning. In other words, the last element has an index of -1, the second-to-last has an index of -2, and so on.

In Python, using negative indexes can be useful for various reasons:

1. **Reverse Access:** Negative indexes make it easy to access elements in reverse order

without explicitly calculating the index of the last element.

2. **Convenient Slicing:** When using slicing to extract a portion of a sequence, negative indexes can be used to specify ranges relative to the end of the sequence.
3. **Finding Last Elements:** Negative indexes are often used to quickly access the last elements of a sequence, especially when the length of the sequence is not known.

Q.15: Write a Python program to count occurrences of a substring in a string.

Ans:

```
 #Function to count occurrences of a substring in a string
def count_substring_occurrences(main_string, substring):
    count = 0
    index = 0
    while index < len(main_string):
        index = main_string.find(substring, index)
        if index == -1:
            break
        count += 1
        index += len(substring)
    return count

 #Get input from the user
main_string = input("Enter the main string: ")
substring = input("Enter the substring to
count: ")

 #Calculate and print the count of occurrences
occurrences =
count_substring_occurrences(main_string,
substring)
print(f"The substring '{substring}' occurs
{occurrences} times in the main string.")
```

Q.16: Write a Python program to count the occurrences of each word in a given sentence.

Ans:
```
def count_words(sentence):
     #Split the sentence into words
    words = sentence.split()

     #Create an empty dictionary to store word
frequencies
    word_count = {}
    #Iterate through the words and count their
occurrences
    for word in words:
        #Remove punctuation and convert to
lowercase to ensure accurate counting
        word = word.strip('.,!?').lower()

        #Check if the word is already in the
dictionary
        if word in word_count:
            #If it is, increment the count
            word_count[word] += 1
        else:
            # If it's not, add it to the
dictionary with a count of 1
            word_count[word] = 1

    return word_count

#Input sentence
sentence = "This is a sample sentence. This
sentence contains some words, and it's a simple
example."
```

```
 #Count the occurrences of each word in the
sentence
word_frequency = count_words(sentence)

 #Print the word frequencies
for word, count in word_frequency.items():
    print(f"'{word}': {count}")
```

Q.17: Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

Ans:

```
 #Get input from the user
string1 = input("Enter the first string: ")
string2 = input("Enter the second string: ")

 #Check if both strings have at least two
characters
if len(string1) >= 2 and len(string2) >= 2:
     #Swap the first two characters of each
string
    new_string1 = string2[:2] + string1[2:]
    new_string2 = string1[:2] + string2[2:]
```

#Combine the modified strings with a space in between

```
    result_string = new_string1 + ' ' +
new_string2
```

#Print the result

```
    print("Resulting string:", result_string)
else:
    print("Both strings must have at least two
characters.")
```

Q.18: Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead if the string length of the given string is less than 3, leave it unchanged.

Ans: #Get input from the user

```
input_string = input("Enter a string: ")
```

#Check the length of the string

```
if len(input_string) >= 3:
     #Check if the string already ends with 'ing'
    if input_string.endswith("ing"):
        result_string = input_string + "ly"
    else:
        result_string = input_string + "ing"
else:
     #If the string has less than 3 characters, leave it unchanged
    result_string = input_string

 #Print the result
print("Resulting string:", result_string)
```

Q.19: Write a Python program to find the first appearance of the substring 'not' and 'poor' from a given string, if 'not' follows the 'poor', replace the whole 'not'...'poor' substring with 'good'. Return the resulting string.

```
Ans:  #Function to replace 'not'...'poor' with
'good'
def replace_not_poor_with_good(input_string):
     #Find the indexes of 'not' and 'poor' in
the string
    index_not = input_string.find('not')
    index_poor = input_string.find('poor')

     #Check if 'not' and 'poor' both exist in
the string
    if index_not != -1 and index_poor != -1 and
index_not < index_poor:
         #Replace the 'not'...'poor' substring
with 'good'
        return input_string[:index_not] +
'good' + input_string[index_poor + 4:]
    else:
        return input_string  # Return the
original string

 #Get input from the user
input_string = input("Enter a string: ")

 #Call the function to perform the replacement
```

```
result_string =
replace_not_poor_with_good(input_string)
```

 #Print the resulting string

```
print("Resulting string:", result_string)
```

Q.20: Write a Python function that takes a list of words and returns the length of the longest one.

Ans:
```
def find_longest_word(word_list):
    if not word_list:
        return 0  # Return 0 if the list is
empty
    else:
        longest_word_length = len(word_list[0])
# Initialize with the length of the first word
        for word in word_list:
            current_length = len(word)
            if current_length >
longest_word_length:
                longest_word_length =
current_length
```

```
        return longest_word_length
```

#Example:

```
words = ["apple", "banana", "cherry", "date", "elderberry"]

result = find_longest_word(words)

print("The length of the longest word is:", result)
```

Q.21: Write a Python function to reverses a string if its length is a multiple of 4.

Ans:
```
def reverse_string_if_multiple_of_4(input_string):
    if len(input_string) % 4 == 0:
        return input_string[::-1]  # Reverse the string using slicing
    else:
        return input_string  # Return the original string
```

#Example:

```
input_str = "python"
```

```
result =
reverse_string_if_multiple_of_4(input_str)
print("Result:", result)
```

Q.22: Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.

Ans: #Function to get the new string

```
def get_new_string(input_string):
    if len(input_string) < 2:
        return ""  # Return an empty string if
length is less than 2
    else:
        return input_string[:2] +
input_string[-2:]  # Combine the first 2 and
last 2 characters
```

#Example:

```
input_str = input("Enter a string: ")
result = get_new_string(input_str)
print("Result:", result)
```

Q.23: Write a Python function to insert a string in the middle of a string.

Ans: def
insert_string_in_middle(original_string,
string_to_insert):

#Calculate the midpoint of the original string

```
    middle_index = len(original_string) // 2
```

#Insert the string in the middle

```
    result_string =
original_string[:middle_index] +
string_to_insert +
original_string[middle_index:]

    return result_string
```

#Example:

```
original_str = "Hello, world!"

string_to_insert = "Python"

result = insert_string_in_middle(original_str,
string_to_insert)
```

```
print("Result:", result)
```