**Module – 2**

**(Fundamentals of python)**

**Name:- Vijay Ratnotar Mo.No.9429099686**

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

**ANS.**

Python program to check if a number is positive, negative, or zero:

def check\_no(num):

if num > 0:

print("Positive")

elif num < 0:

print("Negative")

else:

print("Zero")

# Example usage

check\_no(5)

check\_no(-2)

check\_no(0)

1. **Write a Python program to get the Factorial number of given number.**

**ANS.**

Python program to get the factorial of a given number:

def factorial(num):

if num == 0:

return 1

else:

return num \* factorial (num - 1)

# Example usage

print (factorial (5))

1. **Write a Python program to get the Fibonacci series of given range.**

**ANS.**

Python program to get the Fibonacci series within a given range:

def fibonacci\_series(n):

fib\_series = [0, 1]

while fib\_series[-1] + fib\_series[-2] <= n:

fib\_series.append(fib\_series[-1] + fib\_series[-2])

return fib\_series

# Example usage

print(fibonacci\_series(100))

1. **How memory is managed in Python?**

**ANS.**

Python automates memory management, letting the Python memory manager handle memory allocation and release. It uses a private heap for memory control. The Python interpreter performs these tasks, not the programmer. Additionally, Python employs a garbage collector that recycles unneeded memory, optimizing memory usage.

1. **What is the purpose continue statement in python?**

**ANS.**

When using loops in Python, you can use the `continue` statement to skip the current loop iteration and move on to the next one. This is usually helpful when you need to ignore certain iterations based on a condition without stopping the entire loop prematurely.

1. **Write a python program to sum of the first n positive integers.**

**ANS.**

Swap two numbers with and without a temporary variable:

# With temp variable

def swap\_with(a, b):

temp = a

a = b

b = temp

return a, b

# Without temp variable

def swap\_without(a, b):

a = a + b

b = a - b

a = a - b

return a, b

# Test

num1 = 5

num2 = 10

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

num1, num2 = swap\_with(num1, num2)

print("After swapping with temp variable:", num1, num2)

num1, num2 = swap\_without(num1, num2)

print("After swapping without temp variable:", num1, num2)

1. **Write a Python program to calculate the length of a string.**

**ANS.**

Certainly! Here's a simple Python program to calculate the length of a string:

def calculate\_string\_length(string):

length = 0

for char in string:

length += 1

return length

# Test

input\_string = "Hello, world!"

print("Length of the string:", calculate\_string\_length(input\_string))

The program goes through each letter in a given text, counting how many there are to find out how long the text is. You can try out different texts by changing what's in `input\_string`.

1. **Write a Python program to count the number of characters (character frequency) in a string**

**ANS.**

Here's a Python program to count the number of characters (character frequency) in a string:

python

def count\_Number(string):

frequency = {}

for char in string:

if char in frequency:

frequency[char] += 1

else:

frequency[char] = 1

return frequency

# Example and Output

input\_str = "hello world"

result = count\_Number(input\_str)

print("Input string:", input\_str)

print("Character frequency:", result)

The program examines each character in the input text. It maintains a database (`frequency`) where each entry corresponds to a distinct character, and the associated value reflects the character's frequency within the text. Ultimately, the program delivers this database, which contains information about the frequencies of each character.

**Output:**

Input string: hello world

Character frequency: {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}

Using this program, you can input any string, such as "hello world" in the example provided. It will then analyze the string and count the frequency of each unique character within it. You can experiment with different strings to observe the program's behavior.

1. **What are negative indexes and why are they used?**

**ANS.**

In Python, negative numbers are used as indexes to retrieve elements from the end of a sequence, such as strings, lists, or tuples. Unlike positive indexes that start from the beginning, negative indexes begin from the end. The last element has an index of -1, followed by -2 for the second-to-last element, and so on. Using negative indexes offers a flexible way to access elements without determining the sequence's length beforehand. It allows for convenient manipulation of elements from the end of the sequence, making it particularly useful when dealing with sequences of unknown or varying lengths.

Here's an example to illustrate the usage of negative indexes:

my\_string = "Hello, world!"

# Accessing the last character

print("Last character:", my\_string[-1])

# Accessing the second-to-last character

print("Second-to-last character:", my\_string[-2])

# Accessing a range of characters from the end

print("Characters from the end:", my\_string[-6:-1])

**Output:**

Last character: !

Second-to-last character: d

Characters from the end: world

In this example, instead of using positive indexes to access characters from the beginning, negative indexes are used to access characters from the end of the string `my\_string`. `-1` points to the last character, `-2` points to the second-to-last character, and `-6:-1` points to a range of characters beginning with the sixth character from the end and going until, but not including, the last character. Using negative indexes simplifies working with sequences since it eliminates the need to know their lengths.

1. **Write a Python program to count occurrences of a substring in a string.**

**ANS.**

Sure! Here's a Python program that counts the occurrences of a substring within a given string:

def count\_sub(string, substring):

count = 0

start = 0

while True:

start = string.find(substring, start) # Find the next occurrence of substring

if start == -1: # If no more occurrences are found, break the loop

break

count += 1

start += 1 # Move start index to search for next occurrence

return count

# Example

if \_\_name\_\_ == "\_\_main\_\_":

main\_string = "This is an example string. Example is a good way to learn."

sub\_string = "example"

occurrences = count\_sub(main\_string, sub\_string)

print(f"The substring '{sub\_string}' occurs {occurrences} times in the main string.")

"The `count\_sub` function counts how many times a substring appears within a main string. It takes the main string and the substring as inputs. The function iteratively goes through the main string, finding occurrences of the substring using `find()`. It then tallies these occurrences and returns the count. As an example, if we want to count how often 'example' appears in a string, the function will do so and display the result."

1. **Write a Python program to count the occurrences of each word in a given sentence**

**ANS.**

Certainly! Here's a Python program that counts the occurrences of each word in a given sentence:

def count\_word(sentence):

word\_count = {}

words = sentence.split() # Split the sentence into words

for word in words:

word\_count[word] = word\_count.get(word, 0) + 1 # Increment the count for each word

return word\_count

**# Example**

if \_\_name\_\_ == "\_\_main\_\_":

example\_sentence = "This is an example sentence. This sentence contains example words."

word\_occurrences = count\_word(example\_sentence)

print("Word occurrences in the given sentence:")

for word, count in word\_occurrences.items():

print(f"'{word}': {count}")

In this program, the `count\_word` function takes a sentence as input and returns a dictionary where the keys are the unique words in the sentence and the values are the counts of how many times each word appears.

We split the input sentence into individual words using the `split()` method. Then, we iterate over each word, incrementing its count in the dictionary. Finally, we print out each word along with its count.

In the provided example, the program counts the occurrences of each word in the given sentence and prints the results.

1. **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.**

Here's a Python program that accomplishes what you're asking for:

python

def swap\_first(str1, str2):

new\_str1 = str2[:2] + str1[2:] # Swap first two characters of str1 with str2

new\_str2 = str1[:2] + str2[2:] # Swap first two characters of str2 with str1

return new\_str1 + ' ' + new\_str2 # Combine the two strings separated by a space

# Example

if \_\_name\_\_ == "\_\_main\_\_":

str1 = "hello"

str2 = "world"

result = swap\_first(str1, str2)

print("Resulting string after swapping first two characters of each string and combining them with a space:")

print(result)

In this program:

The `swap\_first` function swaps the first two characters of two input strings and then concatenates them with a space. For example, with the strings "hello" and "world," the function swaps "he" with "wo," resulting in "wollo" and "herld." The final output is "wollo herld."

1. **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.**

Here's a Python program to accomplish the task:

def add\_ing\_ly(string):

if len(string) < 3:

return string

elif string.endswith('ing'):

return string + 'ly'

else:

return string + 'ing'

**# Example and Output**

input\_str1 = "play"

input\_str2 = "swimming"

input\_str3 = "go"

result1 = add\_ing\_ly(input\_str1)

result2 = add\_ing\_ly(input\_str2)

result3 = add\_ing\_ly(input\_str3)

print("Original string 1:", input\_str1)

print("Resulting string 1:", result1)

print("Original string 2:", input\_str2)

print("Resulting string 2:", result2)

print("Original string 3:", input\_str3)

print("Resulting string 3:", result3)

The program examines the string's length. If it's below three, the string remains untouched. Next, it determines if the string concludes with "ing." If true, "ly" is appended. If not, "ing" is added to the string's end.

**Output:**

Original string 1: play

Resulting string 1: playing

Original string 2: swimming

Resulting string 2: swimmingly

Original string 3: go

Resulting string 3: go

In this example, the first string is "play", so "ing" is added to make it "playing". The second string already ends with "ing", so "ly" is added to make it "swimmingly". The third string is "go", which is less than 3 characters long, so it remains unchanged. You can test it with different input strings to see how it behaves.

1. **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.**

Here's a Python program to achieve the specified task:

def replace\_str(string):

not\_index = string.find('not')

poor\_index = string.find('poor')

if not\_index != -1 and poor\_index != -1 and not\_index < poor\_index:

return string[:not\_index] + 'good' + string[poor\_index + 4:]

else:

return string

**# Example and Output**

input\_str = "The weather is not so poor today."

result = replace\_str(input\_str)

print("Original string:", input\_str)

print("Resulting string:", result)

This program uses the `find()` method to locate the indices of the first appearances of the substrings 'not' and 'poor'. Then, it checks if 'not' appears before 'poor' and if both substrings are found. If so, it replaces the substring 'not'...'poor' with 'good'. Otherwise, it returns the original string unchanged.

**Output:**

Original string: The weather is not so poor today.

Resulting string: The weather is good today.

In this example, the substring 'not' appears before 'poor', so it is replaced with 'good'. You can test it with different input strings to see how it behaves.

1. **Write a Python function that takes a list of words and returns the length of the longest one.**

**ANS.**

Certainly! Here's a Python function that takes a list of words and returns the length of the longest one:

def longest\_length(word\_list):

if not word\_list:

return 0 # Return 0 if the list is empty

max\_length = len(word\_list[0]) # Initialize with the length of the first word

for word in word\_list:

if len(word) > max\_length:

max\_length = len(word)

return max\_length

**# Example and Output**

words = ["apple", "banana", "kiwi", "strawberry", "orange"]

result = longest\_length(words)

print("List of words:", words)

print("Length of the longest word:", result)

This function iterates through the list of words, updating the `max\_length` variable whenever it encounters a word longer than the current maximum length. Finally, it returns the length of the longest word.

**Output:**

List of words: ['apple', 'banana', 'kiwi', 'strawberry', 'orange']

Length of the longest word: 10

In this example, the longest word is "strawberry", which has a length of 10 characters. You can test it with different lists of words to see how it behaves.

1. **Write a Python function to reverses a string if its length is a multiple of 4.**

**ANS.**

Here's a Python function that reverses a string if its length is a multiple of 4:

def reverse\_string(input\_string):

if len(input\_string) % 4 == 0:

return input\_string[::-1]

else:

return input\_string

**# Example and Output**

input\_str = "Vijay Ratnottar"

result = reverse\_string(input\_str)

print("Input string:", input\_str)

print("Result:", result)

This function first checks if the length of the input string is a multiple of 4. If it is, it reverses the string using slicing (`[::-1]`). If not, it returns the original string unchanged.

**Output:**

Input string: Vijay Ratnottar

Result: Vijay Ratnottar

In this example, the input string "Vijay Ratnottar" has a length that's not divisible by 4. Therefore, the function retains the original string without any alterations. You can test this function using various input strings to observe its behavior.

1. **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.**

Here's a Python program to achieve the specified task along

def get\_first\_and\_last\_two\_chars(input\_string):

if len(input\_string) < 2:

return ""

else:

return input\_string[:2] + input\_string[-2:]

# Example and Output

input\_str = "Python"

result = get\_first\_and\_last\_two\_chars(input\_str)

print("Input string:", input\_str)

print("Result:", result)

The program determines whether the input string has fewer than two characters. If so, it returns an empty string. If the string has more than two characters, it combines the first and last two characters and returns them as the output.

**Output:**

Input string: Python

Result: Pyon

You can test this program with different input strings to see how it works.

1. **Write a Python function to insert a string in the middle of a string.**

**ANS.**

Here's a Python function that inserts a string into the middle of another string:

def insert\_string\_middle(main\_string, insert\_string):

middle\_index = len(main\_string) // 2

result = main\_string[:middle\_index] + insert\_string + main\_string[middle\_index:]

return result

# Test

main\_str = "Hello, world!"

insert\_str = "Python"

result\_str = insert\_string\_middle(main\_str, insert\_str)

print("Result:", result\_str)

This function finds the midpoint of a given main string, splits the string into two parts at that midpoint, and inserts a specified string between the two parts. Experiment by using different main strings and insert strings to observe the function's operation.