Q1. Can you create a programme or function that employs both positive and negative indexing? Is there any repercussion if you do so?

Sol:-

def print\_characters(string):

length = len(string)

for i in range(-length, length):

print(f"Character at index {i}: {string[i]}")

# Example usage

text = "Hello, World!"

print\_characters(text)

One potential repercussion of using both positive and negative indexing is the possibility of accessing characters outside the valid index range. For example, if you attempt to access string[-1] in a string with an empty or single character, it will raise an IndexError. It's important to handle such cases and ensure that the indices are within the valid range to avoid errors.

Q2. What is the most effective way of starting with 1,000 elements in a Python list? Assume that all elements should be set to the same value.

Sol:-

value = "initial\_value"

my\_list = [value] \* 1000

Q3. How do you slice a list to get any other part while missing the rest? (For example, suppose you want to make a new list with the elements first, third, fifth, seventh, and so on.)

Sol:-

my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

new\_list = my\_list[1::2]

print(new\_list)

[2, 4, 6, 8, 10]

Q4. Explain the distinctions between indexing and slicing.

Sol:-

Indexing:

Indexing refers to the process of accessing a specific element within a sequence using its position or index.

It uses square brackets [] notation to specify the index of the desired element.

The index starts from 0 for the first element and goes up to len(sequence) - 1.

Indexing returns a single element at the specified index.

Slicing:

Slicing refers to the process of extracting a subsequence or a portion of a sequence by specifying a range of indices.

It uses the colon : notation to specify the start, stop, and step values of the slice.

The start index is inclusive, the stop index is exclusive, and the step value determines the increment between indices.

Slicing returns a new sequence that contains the elements within the specified range.

The syntax for slicing is [start:stop:step].

my\_list = [1, 2, 3, 4, 5]

# Indexing

element = my\_list[2]

print(element) # Output: 3

# Slicing

sublist = my\_list[1:4]

print(sublist) # Output: [2, 3, 4]

# Slicing with step

sublist\_with\_step = my\_list[0:5:2]

print(sublist\_with\_step) # Output: [1, 3, 5]

Q5. What happens if one of the slicing expression's indexes is out of range?

Sol:-

Index Error:

If the start index or stop index provided is outside the valid range of indices for the sequence, Python raises an IndexError with an error message indicating that the index is out of range.

For example, if you try to slice a list with a start index greater than the length of the list or a stop index beyond the maximum valid index, an IndexError will be raised.

Partial Slicing:

If the start index is out of range but the stop index is within the valid range, or vice versa, Python will handle it gracefully.

In such cases, Python will return a subsequence containing the available elements up to the valid index.

For example, if you try to slice a list with a start index beyond its length but a valid stop index, Python will return a subsequence starting from the first element up to the stop index.

my\_list = [1, 2, 3, 4, 5]

# Index Error

print(my\_list[5]) # Raises IndexError: list index out of range

# Partial Slicing

print(my\_list[2:10]) # Output: [3, 4, 5]

print(my\_list[10:15]) # Output: []

Q6. If you pass a list to a function, and if you want the function to be able to change the values of the list—so that the list is different after the function returns—what action should you avoid?

Sol:-

def modify\_list\_bad(lst):

# This is incorrect - creates a new local reference to a different list

lst = [4, 5, 6]

def modify\_list\_good(lst):

# This is correct - modifies the elements within the list

lst[0] = 4

lst[1] = 5

lst[2] = 6

my\_list = [1, 2, 3]

# Incorrect - does not modify the original list

modify\_list\_bad(my\_list)

print(my\_list) # Output: [1, 2, 3]

# Correct - modifies the original list

modify\_list\_good(my\_list)

print(my\_list) # Output: [4, 5, 6]

Q7. What is the concept of an unbalanced matrix?

Sol:-

In the context of matrices, an unbalanced matrix refers to a matrix where the number of rows is not equal to the number of columns. In other words, it is a matrix that does not have an equal number of rows and columns.

Q8. Why is it necessary to use either list comprehension or a loop to create arbitrarily large matrices?

Sol:-

It is necessary to use list comprehension or a loop to create arbitrarily large matrices because they provide a way to dynamically generate the elements of the matrix based on a pattern or set of rules. This allows us to handle matrices of any size, including very large ones, without having to manually write each element.