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

Yes, you can create a program or function that employs both positive and negative indexing in Python.

Positive indexing refers to accessing elements in a sequence using indices starting from 0 and increasing sequentially. For example, string[0] accesses the first character of a string.

Negative indexing, on the other hand, allows you to access elements from the end of a sequence by using negative indices. The last element is accessed using -1, the second-last element using -2, and so on. For example, string[-1] accesses the last character of a string

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.

The most effective way to create a Python list with 1,000 elements, all set to the same value, is to use a list comprehension. Here's an example:

value = 42 # The value to be assigned to all elements

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.)

To slice a list and select every other element, you can use the slice notation with a step value of 2. Here's an example:

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

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

Q4. Explain the distinctions between indexing and slicing.

Indexing and slicing are both ways to access elements from a sequence like a list or a string in Python, but they have some distinct differences.

Indexing:

Indexing refers to accessing a specific element in a sequence by its position.

It uses square brackets [] notation with the index value inside, e.g., my\_list[index]

The index starts from 0 for the first element and increases sequentially

Slicing

Slicing refers to accessing a portion of a sequence by specifying a range of indices

It uses the colon : notation with start, stop, and optional step values inside square brackets, e.g., my\_list[start:stop:step]

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

If one of the indexes specified in a slicing expression is out of range, Python handles it in a graceful manner. Here's what happens

When the start index is out of range:

When the stop index is out of range:

When the step value is out of range:

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?

By avoiding the reassignment of the list parameter, you allow the function to have the desired effect of modifying the list's values even after the function returns.

Q7. What is the concept of an unbalanced matrix?

The term "unbalanced matrix" is not a standard terminology in the context of matrices. It seems to be a misnomer or an incorrect usage of the term.

In linear algebra, a matrix is typically referred to as balanced or unbalanced based on its dimensions. A balanced matrix is one where the number of rows is equal to the number of columns, resulting in a square matrix. An unbalanced matrix, on the other hand, is one where the number of rows is not equal to the number of columns, resulting in a rectangular matrix

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

When creating arbitrarily large matrices, it is necessary to use either list comprehension or a loop because these methods allow for dynamic and efficient generation of matrix elements.

List Comprehension: List comprehension is a concise way to create lists in Python. It allows you to generate a list by specifying the elements and the logic to generate them in a single line. When used for matrix creation, list comprehension can generate the elements row by row or column by column, depending on the desired matrix structure.

Loop: Using a loop, such as a for loop, allows you to iterate over the desired range of rows and columns and generate matrix elements incrementally.