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

Solution - Yes, you can create a program or function that employs both positive and negative indexing in Python. Positive indexing starts from 0 for the first element, while negative indexing starts from -1 for the last element. Using both types of indexing allows you to access elements from both ends of the sequence.

```
def access_elements(sequence):
    # Access elements using positive indexing
    for i in range(len(sequence)):
        print("Positive index:", i, "->", sequence[i])

    print("-------------------")

    # Access elements using negative indexing
    for i in range(-1, -(len(sequence) + 1), -1):
        print("Negative index:", i, "->", sequence[i])
```

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.

Solution - 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 or the * operator.

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.)

Solution - To slice a list and extract elements at specific intervals, such as the first, third, fifth, seventh, and so on, you can use slice notation with a step parameter. Here's how you can achieve that:

Q4. Explain the distinctions between indexing and slicing.

Solution - Indexing refers to the process of accessing an individual element or character within a sequence by specifying its position using an index value. The index value represents the location of the element in the sequence, starting from 0 for the first element.

Slicing, on the other hand, allows you to extract a portion or a subsequence from the original sequence by specifying a range of indices. The syntax for slicing is sequence[start:end], where start is the index of the starting element (inclusive) and end is the index of the ending element (exclusive).

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

Solution - If one of the slicing expression's indexes is out of range, Python will handle it gracefully and return the valid slice without raising an error. Here's what happens in different scenarios:

If the start index is out of range:
If the start index is greater than the length of the list, an empty list is returned.
If the start index is negative and greater than the negative length of the list, the entire list is returned.
If the stop index is out of range:

If the stop index is greater than the length of the list, the slice will include all elements until the end of the list.
If the stop index is negative and greater than the negative length of the list, an empty list is returned.
If the step parameter is out of range:

If the step parameter is zero, Python will raise a ValueError with the message "slice step cannot be zero."
If the step parameter is positive, but greater than the length of the list, an empty list is returned.
If the step parameter is negative, but greater than the negative length of the list, the entire list is returned.

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?

Solution - If you want a function to be able to change the values of a list passed to it, you should avoid reassigning the list itself to a new object within the function. In other words, you should avoid using the assignment operator (=) to assign a new list to the parameter that receives the list.

Q7. What is the concept of an unbalanced matrix?

Solution - In the context of machine learning or data analysis, an unbalanced matrix refers to a dataset or a confusion matrix where the number of samples or instances across different classes is significantly imbalanced. In other words, the classes in the dataset have unequal representation.

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

Solution - When creating arbitrarily large matrices, it becomes impractical and inefficient to manually specify each element of the matrix. Using list comprehension or a loop allows for a more dynamic and scalable approach to generate the matrix.