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

In summary, Python allows positive indexing starting at zero and negative indexing starting at -1. In Python, negative indexing denotes that the indexing process begins at the end of the iterable. The final element is located at index -1, the next-to-last at index -2, and so on.

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.

We create an empty an list and run a for loop for n times using the append() method to add elements to the list.

arr = []

for i in range(1000):

arr.append(0)

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.)

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

print(my\_list[::2])

Q4. Explain the distinctions between indexing and slicing.

“Indexing” means referring to an element of an iterable by its position within the iterable. “Slicing” means getting a subset of elements from an iterable based on their indices

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

The slicing operation doesn't raise an error if both your start and stop indices are larger than the sequence length. This is in contrast to simple indexing—if you index an element that is out of bounds, Python will throw an index out of bounds error. However, with slicing it simply returns an empty sequence.

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?

The original lists are not and cannot be immutable.

Q7. What is the concept of an unbalanced matrix?

An unbalanced matrix is a matrix in which the rows or columns have different lengths. This can be a problem in machine learning applications, where the goal is often to predict a value for a given input.

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

List comprehensions are often faster than loops because they use a more optimized internal mechanism for iterating over the collection. Additionally, list comprehensions allow you to perform transformations and filtering in a single statement, which can lead to more efficient code.