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

In Python, you can start indexing from the end of an iterable. This is known as negative indexing.

numbers = [1, 2, 3, 4, 5]

last = numbers[-1]

print(last)

This means you can use both positive and negative indexes to access iterables.

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.)

List = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Show original list

print("\nOriginal List:\n", List)

print("\nSliced Lists: ")

# Display sliced list

print(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?

In the following simplistic example variables x and y are assigned and passed into a function, where, under new names, they’re modified, summed and the result returned:

x,y = 2,3

def some\_function(a,b):

z = a\*2 + b\*2

return z

z = some\_function(x,y)

print(x, y, z)

> 2 3 10

Q7. What is the concept of an unbalanced matrix?

Whenever the cost matrix of an assignment problem is not a square matrix, that is, whenever the number of sources is not equal to the number of destinations, the assignment problem is called an unbalanced assignment problem.

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

List comprehensions collect the result of applying an arbitrary expression to a sequence and return them in a new list. The effect is similar to that of the for loop and the map call. List comprehensions become more convenient when we need to apply an arbitrary expression to a sequence.