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

ANS:

# Negative Indexing in Python:

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

last = list\_items[-1]

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

last = numbers[-1]

print(last)

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.

ANS:

Python is a very flexible language where a single task can be performed in a number of ways, for example initializing lists can be performed in many ways. However, there are subtle differences in these seemingly similar methods. Python which is popular for its simplicity and readability is equally infamous for being slow compared to C++ or Java. The ‘for’ loop is especially known to be slow whereas methods like map() and filter() are known to be faster because they are written in C. Knowing the better and faster way to initialize lists might give you a slight edge in competitive programming.

The following are some of the ways to initialize lists(we create lists of size 1000 and initialize with zeros) in Python.

**Using a for loop and append()**  
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)

Using a while loop with a counter variable  
This is similar to the above method. However we use while loop instead.

arr = []

i = 0

while(i<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.)

ANS:

In Python, list slicing is a common practice and it is the most used technique for programmers to solve efficient problems. Consider a python list, In-order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:)

With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

**Syntax:**

Lst**[** Initial **:** End **:** IndexJump **]**

Q4. Explain the distinctions between indexing and slicing.

ANS:

“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?

ANS:

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?

ANS:

What happens when a function tries to assign or change a value of a variable that has been defined at the module level? Python creates a temporary variable with the same name, and the value of that variable exists only within the scope of the function.

Q7. What is the concept of an unbalanced matrix?

ANS:

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?

ANS:

List Comprehension

Python features functional programming tools like map and filter for mapping operations over sequences and collecting results.

Since this is such a common task in Python coding, Python made a new expression: the **list comprehension** which is more flexible than **map** and **filter**. List comprehensions apply an **arbitrary expression** to items in an iterable rather than applying function. It provides a compact way of mapping a list into another list by applying a function to each of the elements of the list.