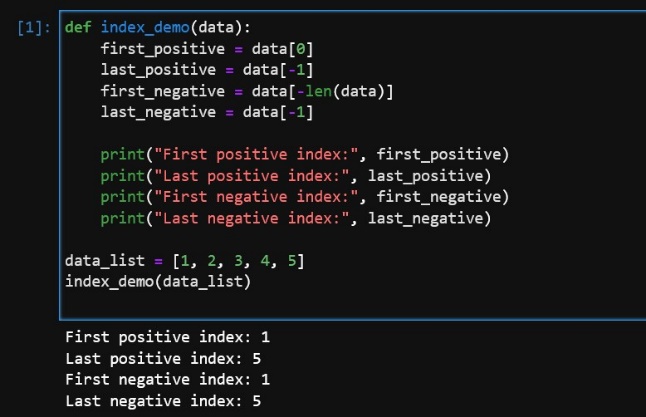
Q1. Can you create a programme or function that employs both positive and negative indexing? Is there any repercussion if you do so?



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.

my\_list = [initial\_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.)

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

Q4. Explain the distinctions between indexing and slicing.

Indexing refers to accessing a single element from a list using its position. Slicing involves extracting a portion of a list by specifying a range of indices. Indexing returns a single element, while slicing returns a new list containing a subset of the original list.

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

If one of the slicing expression's indexes is out of range, Python will not raise an error. Instead, it will automatically adjust the index to the nearest valid value. For example, if you attempt to slice a list with an index that is greater than or equal to its length, the slice will include only the elements up to the end of the list.

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?

To allow a function to modify the values of a list passed as an argument, you should avoid reassigning the list variable inside the function. Instead, modify the list in place using methods like append(), extend(), pop(), etc. Modifying the list variable inside the function will create a new local reference, and the original list outside the function will remain unchanged.

Q7. What is the concept of an unbalanced matrix?

An unbalanced matrix is a matrix where the number of elements in each row is not equal. In other words, the sublists within the main list have different lengths.

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

List comprehension or loops are necessary to create arbitrarily large matrices because they allow you to generate and populate matrix elements dynamically. Without these constructs, you would need to manually write repetitive code for each element, which is not practical for large matrices. List comprehension and loops provide a concise and efficient way to create and fill matrix elements based on patterns or conditions.