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

Yes, we can create a programme or function that employs both positive and negative indexing. But it can be confusing in a more complicated programme.

Code:

t=[]

n=int(input("enter how many elements: "))

for i in range(0,n):

a=(input("enter element: "))

t.append(a)

for num in t[::-1]: # Here t is a list, you can use tuple in the likewise manner.

print(t.index(num)-len(t),t.index(num),num)

print('.................................')

for num in t[::1]: # Here t is a list, you can use tuple in the likewise manner.

print(t.index(num)-len(t),t.index(num),num)

Output:

enter how many elements: 3

enter element: a

enter element: b

enter element: c

-1 2 c

-2 1 b

-3 0 a

.................................

-3 0 a

-2 1 b

-1 2 c

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.

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.

Code:

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.

Code:

arr = []

i = 0

while(i<1000):

arr.append(0)

Using list comprehensions:

It consists of square brackets containing an expression followed by a for clause and further followed by an optional if clause. The expression can be any type of object that we want to put on the list. Since we are initializing the list with zeros, our expression will just be 0.

Code:

arr = [0 for i in range(1000)]

Using the \* operator:

The \* operator can be used as [object]\*n where n is the no of elements in the array.

Code:

arr = [0]\*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.)

Syntax:

Lst[ Initial : End : IndexJump ]

Code:

Lst = [50, 70, 30, 20, 90, 10, 50]

# Display list

print(Lst[1:5:2])

Q4. Explain the distinctions between indexing and slicing.

Slicing in Python refers to extracting a subset or specific part of the sequence list, tuple, or string in a specific range. While indexing refers to accessing a single element from an array, it is used to get slices of arrays.

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?

We should avoid giving the wrong indexes for the items in the list which we wish to change.

Q7. What is the concept of an unbalanced matrix?

A matrix is balanced if all cells in the matrix are balanced and a cell of the matrix is balanced if the number of cells in that matrix that are adjacent to that cell is strictly greater than the value written in this cell.

Adjacent cell means cells in the top, down, left, and right cell of each cell if it exists.

Approach:

* Traverse the given matrix mat[][].
* For each cell of the matrix check if all the adjacent cells i.e., mat[i+1][j], mat[i][j+1], mat[i-1][j], mat[i][j-1] are strictly smaller than the current cell.
* For the corner cells of the matrix, there are only two adjacent cells i.e., mat[i+1][j] and mat[i][j+1] check if all these adjacent cells are strictly smaller than the corner cell.
* For border cell of the matrix, there are 3 adjacent cells i.e., mat[i-1][j], mat[i+1][j], and mat[i][j+1] check if all these adjacent cells are strictly smaller than the border cell.
* If all the above conditions are true for all the cells of the matrix then print “Balanced” else print “Unbalanced”.

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

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.

We can also implement loops for creating large matrices.

This way we don’t have to give input of each matrix elements individually.