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

**ANS:-**

**Python supports negative indexing of an iterable object. We can access the elements of a object from the end by indexes starting from -1, -2, -3... There is no repercussion if we use either positive or negative indexing.**

**def indexing(data, typ):**

**if typ == 'positive':**

**print(data[0:])**

**print(data[4])**

**print(data[2:len(data)])**

**print(data[3:len(data)-1])**

**if typ == 'negative':**

**print(data[-2])**

**print(data[-1:0:-1])**

**print(data[-6:-2])**

**print(data[-3:-1])**

**indexing("abcdefg",'positive')**

**indexing([1,2,3,4,5,6,7],'positive')**

**abcdefg**

**e**

**cdefg**

**def**

**[1, 2, 3, 4, 5, 6, 7]**

**5**

**[3, 4, 5, 6, 7]**

**[4, 5, 6]**

**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:-**

**We can use \* operator for this.**

**my\_list = [12] \* 1000**

**print("Length of the list is::",len(my\_list))**

**print(my\_list)**

**Length of the list is:: 1000**

**[12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12]**

**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:-**

**We use list slicing operations to acheive this.**

**my\_list = [1,2,3,4,5,6,7,8,9,10,11,12]**

**new\_list = my\_list[1:len(my\_list):2]**

**print(new\_list)**

**[2, 4, 6, 8, 10, 12]**

**Q4. Explain the distinctions between indexing and slicing.**

**ANS:-**

**Indexing is the way of accessing a single element of the object by its index or position, where as, slicing enables us to access the sub parts of object. Slicing can be done using indexes.**

**my\_list = [1,2,3,4,5,6,7,8,9,10,11,12]**

**new\_list = my\_list[1:len(my\_list):2] *# this is list sclicing.***

**print(new\_list)**

**print(new\_list[2]) *# this is list indexing***

**[2, 4, 6, 8, 10, 12]**

**6**

**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 one of the indices are out of range. This is in not in the case of 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 exisiting accessed elements. It stops accessing elements after the end of the object is reached.**

**print(new\_list[0:12])**

**[2, 4, 6, 8, 10, 12]**

**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:-**

**We should avoid passing the list as an argument in the function by value. If we do so, it will point to different memory location and will not be able alter the list.**

**def fun(l1):**

**for i in range(len(l1)):**

**l1[i] = l1[i]\*\*2**

**return l1**

**l = [1,2,3,4]**

**print("Original list is",l)**

**print("Call by reference alters list to",fun(l))**

**def fun1(l2):**

**print('Before doing anything, list is::',l2)**

**l2 = [100,200,300,400,500]**

**print("After assigning new values, list is::",l2)**

**l = [1,2,3,4]**

**print("\n Original list is",l)**

**fun1(l)**

**print("After calling function, list is::",l)**

**Original list is [1, 2, 3, 4]**

**Call by reference alters list to [1, 4, 9, 16]**

**Original list is [1, 2, 3, 4]**

**Before doing anything, list is:: [1, 2, 3, 4]**

**After assigning new values, list is:: [100, 200, 300, 400, 500]**

**After calling function, list is:: [1, 2, 3, 4]**

**Q7. What is the concept of an unbalanced matrix?**

**ANS:-**

**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.**

**import numpy as np**

**mat = np.matrix([[1, 2, 3],[4, 5, 6],[7, 8, 9]])**

**mat**

***# This is unbalanced matrix***

**matrix([[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 9]])**

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

**ANS:-**

**In Python 3, we can use numpy library to create large matrices instaed of list comprehension or a loop.**