**ASSIGNMENT – 13**

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

**Ans:**

Yes, you can create a program or function that employs both positive and negative indexing in Python. Positive indexing starts from the beginning of the string or list and uses the index values 0, 1, 2, 3, and so on, while negative indexing starts from the end of the string or list and uses the index values -1, -2, -3, and so on.

Here is an example of a Python function that employs both positive and negative indexing to return a substring of a given string:

def get\_substring(s, start, end):

if start >= 0 and end >= 0:

return s[start:end]

elif start < 0 and end < 0:

return s[start:end:-1][::-1]

else:

return s[:end] if start == 0 else s[start:]

1. **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:**

The most effective way to start with 1,000 elements in a Python list, all set to the same value, is to use the multiplication operator (`\*`) with a list containing the single value.

my\_list = [0] \* 1000

In this example, `my\_list` is a list of 1,000 elements, all set to the value `0`. The `\*` operator is used to repeat the single value `0` 1,000 times, effectively creating a list of 1,000 `0` values.

1. **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:**

To slice a list to get every other element, you can use the slice notation with a step value of 2.

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

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

print(new\_list) # Output: [1, 3, 5, 7, 9]

In this example, `my\_list` is the original list containing 10 elements. To get every other element starting with the first element (index 0), we use the slice notation `my\_list[::2]`. The `::2` indicates that we want to start at the first element, step by 2, and include every other element in the list.

The resulting `new\_list` contains only the elements at odd positions in `my\_list`: the first, third, fifth, seventh, and ninth elements.

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

**Ans:**

Indexing and slicing are two ways of accessing elements in a Python list or other sequence data types such as strings and tuples.

Indexing refers to accessing a single element in a sequence using its position in the sequence, which is represented by an integer index. In Python, indexing starts at 0 for the first element, -1 for the last element, and so on. Indexing can be performed using square brackets (`[]`) after the sequence object, with the index of the desired element inside the brackets. For example, to get the second element of a list, you can use `my\_list[1]`, since the first element has index 0 and the second element has index 1.

Slicing, on the other hand, refers to accessing a subsequence of a sequence by specifying a range of indices. A slice is created by specifying two indices separated by a colon `:` inside the square brackets. The first index indicates the starting point of the slice (inclusive), and the second index indicates the ending point of the slice (exclusive). For example, to get a slice of a list containing the first three elements, you can use `my\_list[0:3]`. This will return a new list with the elements at indices 0, 1, and 2.

1. **What happens if one of the slicing expression’s indexes is out of range?**

**Ans:**

If one of the slicing expression's indexes is out of range, i.e., it is less than 0 or greater than or equal to the length of the sequence, then Python raises an `IndexError` exception.

For example, consider the following list of integers:

my\_list = [1, 2, 3, 4, 5]

If we try to slice the list using an index that is out of range, we will get an `IndexError` exception. For instance, trying to slice the list from index 0 to index 10, which is out of range, will result in an exception:

>>> my\_list[0:10]

IndexError: list index out of range

To avoid this error, it's important to ensure that the indices used for slicing are within the bounds of the sequence.

1. **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:**

If you want a function to be able to change the values of a list passed to it, you should avoid creating a new list object inside the function and assigning it to the name of the original list, as this will re-bind the name to a new object, leaving the original list unchanged.

For example, consider the following function that takes a list of numbers as an argument and returns a new list containing the squares of the numbers:

def square\_list(numbers):

new\_list = [x\*\*2 for x in numbers]

return new\_list

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

Ans:

An unbalanced matrix is a matrix where the number of elements in each row or column is not equal. In other words, the matrix does not have the same number of rows and columns or the same number of elements in each row or column. For example, a matrix with two rows and three columns is unbalanced if the first row has four elements and the second row has three elements. Unbalanced matrices can be difficult to work with because they do not have the same structure as balanced matrices, and operations that require balanced matrices may not work correctly on unbalanced matrices.

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

**Ans:**

It is necessary to use either list comprehension or a loop to create arbitrarily large matrices because it allows you to generate the matrix elements dynamically based on a certain pattern or algorithm. With list comprehension, you can generate a matrix in a concise and readable way, making it easy to understand and modify the matrix generation algorithm. With loops, you have more control over the generation of the matrix elements and can apply more complex operations to generate the elements. Without using list comprehension or loops, you would need to manually define every element in the matrix, which would be impractical for large matrices.