ASSIGNMENT 13

# Q1

Yes, we can create a program or function that employs both positive and negative indexing in Python. Positive indexing starts from 0 and goes up to the length of the list minus 1, while negative indexing starts from -1 and goes down to -length of the list.

def index\_example():

my\_list = ['a', 'b', 'c', 'd', 'e']

# Positive indexing

print(my\_list[0]) # Output: 'a'

print(my\_list[3]) # Output: 'd'

# Negative indexing

print(my\_list[-1]) # Output: 'e'

print(my\_list[-4]) # Output: 'b'

# Q2

The most effective way to start with 1,000 elements in a Python list, all set to the same value, is to use list comprehension with the \* operator for repetition

my\_list = [initial\_value] \* 1000

This creates a new list with 1,000 elements, where each element is set to the initial\_value specified. The \* operator is used to repeat the initial\_value 1,000 times in the list comprehension.

# Q3

To slice a list and get specific elements while skipping others, we can use the step parameter in slicing. For example, to create a new list with elements at odd indices (1st, 3rd, 5th, etc.), we can specify a step of 2:

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

new\_list = my\_list[::2] # Slice with step 2

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

Here, my\_list[::2] returns a new list containing elements from my\_list starting from index 0 and moving with a step of 2, effectively skipping every other element.

# Q4

Indexing: It refers to accessing a specific element in a list by its position, using an index value enclosed in square brackets []. Indexing allows us to retrieve a single element from the list.

Slicing: It refers to extracting a portion of a list by specifying a range of indices using the slicing operator :. Slicing returns a new list that contains the specified portion of the original list. It allows us to retrieve multiple elements or a sublist from the list.

# Q5

If one of the indexes in a slicing expression is out of range (either too small or too large), Python will handle it gracefully. If the index is too small, it is treated as if it starts from the first element of the list. If the index is too large, it is treated as if it ends at the last element of the list. This behaviour prevents the program from raising an IndexError and allows us to slice as much as possible within the valid range of indices.

# Q6

If we pass a list to a function and we want the function to be able to change the values of the list (mutate the list), we don't need to avoid any specific action. In Python, lists are mutable objects, meaning we can modify their contents directly. Therefore, any modifications made to the list within the function will be reflected outside the function.

# Q7

The concept of an unbalanced matrix typically refers to a matrix where the number of elements in each row is not equal. In other words, the rows of the matrix have different lengths. This can happen when dealing with irregular or sparse data structures. Python allows we to create lists of lists to represent matrices, and each inner list can have a different length

matrix = [[1, 2, 3],

[4, 5],

[6, 7, 8, 9]]

Here, the first row has 3 elements, the second row has 2 elements, and the third row has 4 elements.

# Q8

to create arbitrarily large matrices, it is necessary to use either list comprehension or a loop. List comprehension allows us to generate lists dynamically based on a specific pattern or condition. By using list comprehension or a loop, we can control the size and content of the matrix based on our requirements. This approach is suitable for handling large matrices efficiently without the need to manually define each element.For example, to create a

1000x1000 matrix filled with zeros using list comprehension, we can use the following code:

matrix = [[0] \* 1000 for \_ in range(1000)]

This creates a 1000x1000 matrix where each element is initialized to 0. The use of list comprehension with the \* operator allows for efficient creation of large matrices without having to explicitly specify each element.