# Q1. Programme or Function with Positive and Negative Indexing

def positive\_negative\_indexing\_example():

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

# Positive indexing

print("Positive Indexing:", my\_list[2]) # Output: 3

# Negative indexing

print("Negative Indexing:", my\_list[-2]) # Output: 9

positive\_negative\_indexing\_example()

# Q2. Starting with 1,000 Elements in a Python List

elements = [42] \* 1000 # Creates a list with 1,000 elements, all set to the value 42

# Q3. Slicing to Get Specific Elements

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

new\_list = original\_list[::2] # Gets elements at even indices

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

# Q4. Distinctions Between Indexing and Slicing

# - Indexing retrieves a single element.

# - Slicing retrieves a portion (substring or sublist) of the original list.

# Q5. Out of Range Index in Slicing Expression

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

result = my\_list[2:10] # Elements beyond the list length are ignored

print(result) # Output: [3, 4, 5]

# Q6. Avoid Mutating a List Passed to a Function

def avoid\_mutating\_list(my\_list):

# Do not modify the list in-place

new\_list = my\_list[:] # Create a copy if modification is needed

new\_list[0] = 42

return new\_list

original\_list = [1, 2, 3]

modified\_list = avoid\_mutating\_list(original\_list)

print(original\_list) # Output: [1, 2, 3] (unchanged)

# Q7. Concept of an Unbalanced Matrix

# An unbalanced matrix is a matrix where the number of elements in each row is not equal.

# Q8. Use List Comprehension or Loop for Arbitrarily Large Matrices

# List comprehension or loops are used to dynamically generate matrix elements based on certain patterns or conditions,

# making it suitable for creating matrices of arbitrary size.