Assignment 13 Solutions

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

Ans: Below is a basic program showing positive and negative indexing. NO.

In [1]:

```python
my_list = [1,2,3,4,5,6,6,7,8,9,10,11,12,13]
def bi_index(in_list,position):
    return in_list[position]
print('Positive Indexing ->',bi_index(my_list,5))
print('Negative Indexing ->',bi_index(my_list,-1))
```

```
Positive Indexing -> 6
Negative Indexing -> 13
```

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.

In [2]:

```python
start_list = [1 for x in range(1001)] # Quick Way to Create a List Using List Comprehension
print(start_list)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1]
```

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.)

In [3]:

```python
my_list = [x for x in range(1,15)]
print(f'my_list -> {my_list}')
sliced_list = my_list[::2]
print(f'sliced_list -> {sliced_list}')
```

```
my_list -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
sliced_list -> [1, 3, 5, 7, 9, 11, 13]
```

Q4. Explain the distinctions between indexing and slicing.

Ans: Indexing is used when we have to work on index level. While slicing are used over a range of items.

In [4]:

```python
my_list = [x for x in range(1,15)]
print(f'my_list -> {my_list}')
print(f'Example of indexing -> {my_list[1], my_list[5]}')
print(f'Example of slicing -> {my_list[1:5]}')
```

```
my_list -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Example of indexing -> (2, 6)
Example of slicing -> [2, 3, 4, 5]
```

Q5. What happens if one of the slicing expression's indexes is out of range?

Ans: If start index is out of range then it will return empty entity.

In [6]:

```python
my_list = [x for x in range(1,15)]
my_list = [x for x in range(1,15)]
print(f'my_list -> {my_list}')
print(f'Case #1 -> {my_list[20:]}')
print(f'Case #2 -> {my_list[10:100]}')
```

```
my_list -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Case #1 -> []
Case #2 -> [11, 12, 13, 14]
```

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 do?

Ans: Always use `return` statement, if we want the see the changes in the input list.

In [8]:

```python
my_list = [1,2,3,4,5,6]
def modify_list(in_list):
    in_list.append(100)
    return in_list
print(modify_list(my_list))
```

```
[1, 2, 3, 4, 5, 6, 100]
```

Q7. What is the concept of an unbalanced matrix?

Ans: In Unbalanced Matrix number of rows is not same as number of columns.

In [10]:

```
# Python3 program for the above approach
# Define the size of the matrix
N = 4
M = 4

# Function to check given matrix
# balanced or unbalanced
def balancedMatrix(mat):

    # Flag for check matrix is balanced
    # or unbalanced
    is_balanced = True

    # Iterate row until condition is true
    i = 0
    while i < N and is_balanced:

        # Iterate cols until condition is true
        j = 0
        while j < N and is_balanced:

            # Check for corner edge elements
            if ((i == 0 or i == N - 1) and
                (j == 0 or j == M - 1)):
                if mat[i][j] >= 2:
                    isbalanced = False

            # Check for border elements
            elif (i == 0 or i == N - 1 or
                j == 0 or j == M - 1):
                if mat[i][j] >= 3:
                    is_balanced = False

            # Check for the middle ones
            else:
                if mat[i][j] >= 4:
                    is_balanced = False

            j += 1
        i += 1

    # Return balanced or not
    if is_balanced:
        return "Balanced"
    else:
        return "Unbalanced"

# Driver code

# Given matrix mat[][]
mat = [ [ 1, 2, 3, 4 ],
        [ 3, 5, 2, 6 ],
        [ 5, 3, 6, 1 ],
        [ 9, 5, 6, 0 ] ]

# Function call
print(balancedMatrix(mat))
```

```
Unbalanced
```

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

Ans: List comprehension or a Loop helps creation of large matrices easy. it also helps to implemeent and avoid manual errors. It also makes reading code easy. Also lot of time for manual feeding is reduced.