

# 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?

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

```
Positive Indexing -> 5
Negative Indexing -> 10
```

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

[illegible]

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 [4]: 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 [5]: 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 [12]: 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[30:]}')
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 to see the changes in the input list.

```
In [13]: my_list = [1,2,3,4,5,6]
def modify_list(in_list):
    in_list.append(200)
    return in_list
print(modify_list(my_list))
```

```
[1, 2, 3, 4, 5, 6, 200]
```

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

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

## 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 implement and avoid manual error. it also makes reading code easy. Also lot of time for manual feeding is reduced.

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js