Assignment-4

1. **What exactly is []?**
   Ans.
   It is the parenthesis, which is used as an empty list. That means it doesn't have any element in it.

2. **In a list of values stored in a variable called spam, how would you assign the value 'hello' as the third value? (Assume [2, 4, 6, 8, 10] are in spam.)**
   Ans.

```
In [1]: spam = [2, 4, 6, 8, 10]
        spam
Out[1]: [2, 4, 6, 8, 10]

In [2]: spam[2] = 'Hello'

In [3]: spam
Out[3]: [2, 4, 'Hello', 8, 10]
```

   Let's pretend the spam includes the list ['a', 'b', 'c', 'd'] for the next three queries.

3. **What is the value of spam[int(int('3'* 2) / 11)]?**
   Ans.

```
In [4]: spam = ['a', 'b', 'c', 'd']

In [5]: spam
Out[5]: ['a', 'b', 'c', 'd']

In [9]: spam[int(int('3'* 2) / 11)]
Out[9]: 'd'
```

4. **What is the value of spam[-1]?**
   Ans.

```
In [7]: spam
Out[7]: ['a', 'b', 'c', 'd']

In [8]: spam[-1]
Out[8]: 'd'
```

5. **What is the value of spam[:2]?**
   Ans.

```
In [9]: spam
Out[9]: ['a', 'b', 'c', 'd']

In [10]: spam[:2]
Out[10]: ['a', 'b']
```

Let's pretend bacon has the list [3.14, 'cat', '11', 'cat', True] for the next three questions.

6. **What is the value of bacon.index('cat')?**
   Ans.

```
In [11]: bacon = [3.14, 'cat', '11', 'cat', True]
         bacon
Out[11]: [3.14, 'cat', '11', 'cat', True]

In [12]: bacon.index('cat')
Out[12]: 1
```

7. **How does bacon.append(99) change the look of the list value in bacon?**
   Ans.

```
In [14]: bacon
Out[14]: [3.14, 'cat', '11', 'cat', True]

In [15]: bacon.append(99)

In [16]: bacon
Out[16]: [3.14, 'cat', '11', 'cat', True, 99]
```

8. **How does bacon.remove('cat') change the look of the list in bacon?**
Ans.

```
In [16]: bacon
Out[16]: [3.14, 'cat', '11', 'cat', True, 99]

In [17]: bacon.remove('cat')

In [18]: bacon
Out[18]: [3.14, '11', 'cat', True, 99]
```

9. **What are the list concatenation and list replication operators?**
Ans.
List Concatenation operators:

+ operator:
It's very easy to use + operator for string concatenation. This operator can be used to add multiple strings together. However, the arguments must be a string.

Ex.

```
In [21]: var1 = "Hello"
         var2 = "World"

         var3 = var1 + " " + var2
         print(var3)

         Hello World
```

% operator:
% operator is used for string formatting, it can also be used for string concatenation. It's useful when we want to concatenate strings and perform simple formatting.

Ex.

```
In [22]: var1 = "Hello"
         var2 = "iNeuron"

         print("%s %s" %(var1, var2))

         Hello iNeuron
```

List Replication operators:

*** operator:**

Multiplication operator is use to create multiple copies of a string. Actually, it creates a replication of the string.

Ex.

```
In [23]: a = "Ciao"

         b = a * 3
         print(f"Original string: {a}")
         print(f"New string: {b}")

         Original string: Ciao
         New string: CiaoCiaoCiao
```

10. What is difference between the list methods append() and insert()?
Ans.

append()	**insert()**
It adds an element at the end of the list.	This method can be used to insert value at any desired position.
In append() method, the argument passed is the single element at the end of the list.	It takes two arguments-element and the index at which the element has to be inserted.
Syntax: list_name.append(element)	Syntax: List_name(index, element)

Ex.
`In [21]: l = ['Captain America']`
`l.append('Iron Man')`
`l.append('Thor')`
`print(l)`
`['Captain America', 'Iron Man', 'Thor']`
	Ex.
	`In [22]: l`
	`Out[22]: ['Captain America', 'Iron Man', 'Thor']`
	`In [23]: l.insert(1, 'Dr. Strange')`
	`print(l)`
	`['Captain America', 'Dr. Strange', 'Iron Man', 'Thor']`

11. What are the two methods for removing items from a list?
Ans.

a) **del[a:b]**
This method deletes all the elements in range starting from index ‘a’ till ‘b’ mentioned in arguments.

Ex.

```
In [31]: list1 = [2, 1, 3, 5, 4, 3, 8]

         del list1[2:5]

         print("List of elements after deleting: ", end="")
         for i in range(0, len(list1)):
             print(list1[i], end=" ")

         List of elements after deleting: 2 1 3 8
```

b) pop()

This method deletes the element at the position mentioned in its argument.

Ex.

```
In [40]: list1

Out[40]: [2, 1, 3, 8]

In [41]: list1.pop(2)

         print("List of elements after popping: ", end="")
         for i in range(0, len(list1)):
             print(list1[i], end=" ")

         List of elements after popping: 2 1 8
```

c) remove()

This method is used to delete the first occurrence of number mentioned in its arguments.

Ex.

```
In [45]: lis = [2, 6, 45, 3, 76, 3, 7, 5, 6]

         lis.remove(3)
         print("List of elements after deleting 3: ", end="")
         for i in range(0, len(lis)):
             print(lis[i], end=" ")

         List of elements after deleting 3: 2 6 45 76 3 7 5 6
```

d) clear()

This function is used to erase all the elements of list. After this operation, list becomes empty.

Ex.

```
In [47]: lis1 = [2, 6, 45, 3, 76]
         lis1

Out[47]: [2, 6, 45, 3, 76]

In [48]: lis1.clear()

         print("List of elements after clearing: ", end=" ")
         for i in range(0, len(lis1)):
             print(lis1[i], end=" ")

         List of elements after clearing:
```

12. Describe how list values and string values are identical.

Ans.

- The similarity between Lists and Strings in Python is that both are sequences.
- The differences between them are that firstly, Lists are mutable but Strings are immutable.
- Secondly, elements of a list can be of different data types whereas a String only contains characters that are all of String type.

13. What is the difference between tuples and lists?

Ans.

List	Tuple
Lists are mutable	Tuples are immutable
List has a dynamic charactics.	Tuple is static in nature which makes it faster than List.
Implication of iterations is Time-comsuming	The implication of iteration is comparatively Faster
Lists consume more memory	Tuple consume less memory as compared to the list.
Lists have several built-in methods	Tuple does not have many built-in methods.
The unexpected changes and errors are more likely to occur.	In tuple, it is hard to take place.

14. How do you type a tuple value that only contains the integer 42?

Ans. (42,)

15. **How do you get a list value's tuple form? How do you get a tuple value's list form?**

Ans.

List to Tuple:

```
In [8]: lis = [1, 34, 4, 56, 2]
        lis

Out[8]: [1, 34, 4, 56, 2]

In [9]: print(tuple(lis))

        (1, 34, 4, 56, 2)
```

Tuple to List:

```
In [12]: tup = ('Racheal', 'Monica', 'Pheobe',
                'Joey', 'Chandler', 'Ross')

         tup

Out[12]: ('Racheal', 'Monica', 'Pheobe', 'Joey', 'Chandler', 'Ross')

In [14]: print(list(tup))

         ['Racheal', 'Monica', 'Pheobe', 'Joey', 'Chandler', 'Ross']
```

16. **Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?**

Ans.

Variables will contain references to list values themselves.

17. **How do you distinguish between copy.copy() and copy.deepcopy()?**

Ans.

The copy.copy() method will do a shallow copy of a list, while the copy.deepcopy() function will do a deep copy of a list. That is, only copy.deepcopy() will duplicate any lists inside the list.