1. **What exactly is []?**

The empty square brackets [] are used to denote an empty list.

1. **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.)**

spam = [2, 4, 6, 8, 10]

spam[2] = “hello”

print(spam) = [2, 4, ‘hello’, 8, 10]

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

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

spam[int(int('3' \* 2) / 11)]?

Solution :

3 will be the result of this [int(int('3' \* 2) / 11)]

**Then ; spam[3] = ‘d’** #’d’ is the answer for this.

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

spam[-1] = ‘d’

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

spam[:2] = [‘a’,’b’]

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

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

bacon.index('cat') = 1

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

bacon.append(99)

bacon = [3.14, 'cat', 11, 'cat', True, 99]

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

bacon.remove('cat')

bacon **=**  [3.14, 11,’cat’,True]

1. **What are the list concatenation and list replication operators?**

In Python, the list concatenation operator is the plus sign (+), and the list replication operator is the asterisk (\*) symbol.

(a).

list1 = [1, 2, 3]

list2 = [4, 5, 6]

concatenated\_list = list1 + list2

print(concatenated\_list) = [1, 2, 3, 4, 5, 6]

list1 = [1, 2, 3]

replicated\_list = list1 \* 3

print(replicated\_list) = [1, 2, 3, 1, 2, 3, 1, 2, 3]

1. **What is difference between the list methods append() and insert()?**

The append() method is used to add an element to the end of a list.

# List.append(Element)

The insert() method is used to insert an element at a specific position in a list

# List.insert(Index,Element)

1. **What are the two methods for removing items from a list?**
2. Remove()
3. Pop()
4. **Describe how list values and string values are identical.**

List values and string values in Python share certain similarities and characteristics.

**Sequential Data:** Both lists and strings represent sequential data structures. They consist of an ordered collection of individual elements. In a string, the elements are characters, while in a list , the elements can be of any data type, including other lists.

**Indexing:** Both lists and strings support indexing.

**Slicing :** Lists and strings also support slicing, which allows you to extract a portion of the sequence by specifying a range of indices. Slicing is done using the colon (:) operator.

**Immutable vs. Mutable:**

strings are immutable, while lists are mutable.

**Iteration:** Lists and strings can be iterated over using loops or comprehensions. You can process each element or character one by one by iterating over the sequence.

**Concatenation:** Both lists and strings support concatenation, which means you can combine two lists or two strings to create a new one. In Python, you can use the + operator for concatenation.

1. **What's the difference between tuples and lists?**

Tuples and lists are used to store collections of items, but they have some fundamental differences. Here are the main differences between tuples and lists:

**Mutability**: Tuples are immutable, meaning their elements cannot be modified once they are created. In contrast, lists are mutable, allowing you to add, remove, or modify elements after creation.

**Syntax**: Tuples are defined using parentheses (), whereas lists are defined using square brackets [].

**Length**: Tuples generally require less memory compared to lists.

**Operations**: Lists offer more built-in methods and operations compared to tuples due to their mutability. With lists, you can append, remove, insert, sort, and perform other modifications directly. Tuples have a limited set of operations, mostly related to indexing and slicing.

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

my\_tuple = (42,)

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

Let , “Li” is a with having some elements (Values) :

Li =[1,2,3,4,5,6,7]

Convert the list value in Tuple form :

t = tuple([1,2,3,4,5,6,7]) or t = tuple([Li])

same as it is , convert the tuple for in List:

Let,” t” is tuple having some values:

t= (1,2,3,4,5,6,7,8)

Convert tuple “t” in List:

List=list((t))

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

Variables that "contain" list values are actually referring to the memory address where the list is stored. These variables are known as references or pointers to the list object.

When you assign a list to a variable, the variable stores the memory address of the list rather than the actual list data itself. This means that the variable doesn't directly contain the list values, but it holds a reference to the location in memory where the list is stored.

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

The copy.copy() and copy.deepcopy() functions are used to create copies of objects. The key difference between them is how they handle object references.

**copy.copy():**

The copy.copy() function performs a shallow copy of an object.

It creates a new object and then copies the references of the original object's elements to the new object.

If the elements of the object are mutable (e.g., lists, dictionaries), changes to those elements in the copied object will be reflected in the original object and vice versa.

However, if the elements are immutable (e.g., numbers, strings), the copied object will have its own independent copy of those elements.

The copy created by copy.copy() is considered a "one level deep" copy.

**copy.deepcopy():**

The copy.deepcopy() function performs a deep copy of an object.

It creates a completely independent copy of the original object and all its nested objects.

This means that changes to any element in the copied object will not affect the original object, and vice versa.

Both mutable and immutable elements of the original object are copied, resulting in a new object with its own copies of all elements, regardless of their mutability.

The copy created by copy.deepcopy() is considered a "fully independent" copy.