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

Ans.it is a blank list

**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.** spam.insert[2]=’hello’

**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.3

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

Ans.’d’

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

Ans.‘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.1

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

Ans [3.14, 'cat,' 11, 'cat,' True’,99]

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

Ans. [3.14, 'cat,' 11, ' True’,99]

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

Ans. In Python, the **list concatenation operator** is the plus sign (+), and it is used to combine two or more lists into a single list. When you use the plus operator to concatenate lists, a new list is created that contains all the elements from the original lists in the order they were concatenated.

the **list replication operator** is the asterisk (\*) symbol, and it is used to create a new list by repeating the elements of an existing list a specified number of times.

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

Ans. **The append() method** is used to add an element at the end of a list. It takes a single argument, which is the element to be added. The element is appended to the end of the list, increasing the list's length by one.

**insert() method:** The insert() method is used to add an element at a specific index position within a list. It takes two arguments: the index at which to insert the element and the element itself.

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

Ans. remove() method: The remove() method is used to remove the first occurrence of a specified element from a list. It takes a single argument, which is the element to be removed. If the element is found in the list, it is removed.

The pop() method is used to remove an element from a list at a specified index position. It takes an optional argument, which is the index of the element to be removed. If no index is specified, pop() removes and returns the last element in the list.

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

Ans. lists and strings are identical in the following ways:-

**Sequential and Index-Based Access**: Both lists and strings allow sequential access to their elements using indexing. You can access individual elements of a list or a string by referring to their respective indices. The first element has an index of 0, the second has an index of 1, and so on.

**Iteration:** You can iterate over both lists and strings using loops, such as the for loop. This allows you to process each element or character in the list or string.

**Slicing:** Both lists and strings support slicing, which means you can extract a subset of elements or characters from them by specifying a range of indices.

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

Ans. **Mutability**: Tuples are immutable, while lists are mutable. Once a tuple is created, its elements cannot be modified. In contrast, elements in a list can be modified, added, or removed.

**Size**: Tuples generally require less memory compared to lists. If you have a sequence of values that won't change, using a tuple can be more memory-efficient than a list.

**Performance**: Tuples are generally faster to access and iterate over than lists. This is because tuples are immutable, allowing for more optimizations by the interpreter. Lists, being mutable, require additional memory allocation and management.

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

**Ans. intup = (42)**

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

Ans. In the above example, the tuple() function is used to convert the list my\_list into a tuple my\_tuple. The resulting tuple contains the same elements as the original list.

For example:-

mylist=[“helo”,1,2,4,”delhi”]

tuple = tuple(mylist)

print(tuple)

To convert a tuple value into a list, you can use the list() function. It takes an iterable, such as a tuple, and returns a list containing the same elements in the same order. Here's an example:

mytuple=(1,2,4,5)

list= list(mytuple)

print(tuple)

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

Ans. Variables that "contain" list values in Python are not necessarily lists themselves. Instead, they contain references or pointers to the list objects.In Python, variables are essentially names that reference objects. When you assign a list to a variable, the variable stores a reference to the memory location where the list object is stored. This reference allows you to access and manipulate the list through the variable.

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

Ans. In Python, the copy module provides two methods, copy() and deepcopy(), for creating copies of objects. Here's how you can distinguish between them:

**copy() method**: The copy() method creates a shallow copy of an object. It constructs a new object and populates it with references to the same elements as the original object. If the original object contains mutable elements, such as nested lists or objects, the copied object will also contain references to the same nested elements. Changes made to these mutable elements in the copied object will be reflected in the original object and vice versa. For example

import copy

original\_list = [1, 2, [3, 4]]

copied\_list = copy.copy(original\_list)

original\_list[2].append(5)

print(copied\_list)

output:- [1, 2, [3, 4, 5]]

**deepcopy() method**: The deepcopy() method creates a deep copy of an object. It constructs a new object and recursively copies all the elements within the object, including nested objects or elements. This means that changes made to the elements in the copied object will not affect the original object, and vice versa.

Example:

import copy

original\_list = [1, 2, [3, 4]]

copied\_list = copy.deepcopy(original\_list)

original\_list[2].append(5)

print(copied\_list)

output:- [1, 2, [3, 4]]