1. What exactly is []?

**Ans.1** It is an empty **list**. It represents a collection of items enclosed within square brackets, but in this case, it indicates that the list contains no elements.

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.)

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

**Ans.2**

**spam** = [2, 4, 6, 8, 10] **# original list**

**spam[2**] = 'hello' **# assign 'hello' as the third value**

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

**Ans.3** the value of spam[int(int('3' \* 2) / 11)] is evaluates to **3**.

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

**Ans.4** The value of **spam[-1]** refers to the last element in the list **spam**. In the given list **spam** (['a', 'b', 'c', 'd']), the last element is **'d'**. Therefore, the value of **spam[-1]** is **'d'**.

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

**Ans.5** The expression **spam[:2]** represents a list slice that retrieves the elements from index 0 up to (but not including) index 2 in the list **spam**. In the given list **spam** (['a', 'b', 'c', 'd']), the elements at indexes 0 and 1 are **'a'** and **'b'**, respectively. Thus, **spam[:2]** will return **['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.6** The value of bacon.index('cat') evaluates to **1**.

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

**Ans.7** After executing **bacon.append(99)**, the updated **bacon** list will be **[3.14, 'cat', 11, 'cat', True, 99]**.

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

Ans.8 When you call **bacon.remove('cat')**, it will modify the **bacon** list by removing the first occurrence of the value **'cat'**. After executing **bacon.remove('cat')**, the updated **bacon** list will be **[3.14, 11, 'cat', True]**.

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

Ans.9

1. The list **concatenation operator** in Python is **+**, and it is used to combine two lists into a single list. For example:

list1 = [1, 2, 3]

list2 = [4, 5, 6]

concatenated = list1 + list2

print(concatenated)

1. The list **replication operator** is **\***, and it is used to repeat a list a certain number of times. For example:

list1 = [1, 2, 3]

repeated = list1 \* 3

print(repeated)

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

**Ans.10**

1. my\_list = [1, 2, 3]

my\_list.append(4)

print(my\_list) **# Output: [1, 2, 3, 4]**

1. my\_list.insert(0, 0)

print(my\_list) **# Output: [0, 1, 2, 3, 4]**

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

**Ans.11**

1. my\_list = [1, 2, 3, 4, 3]

my\_list.**remove**(3)

print(my\_list) **# Output: [1, 2, 4, 3]**

(b) my\_list = [1, 2, 3, 4]

removed\_item = my\_list.pop(2)

print(my\_list) **# Output: [1, 2, 4]**

print(removed\_item) **# Output: 3**

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

**Ans.12**

List values and string values share some similarities, but they are not identical. Here are some common characteristics:

* Both lists and strings are sequence types in Python.
* They can be indexed and sliced to access specific elements or sub-sequences.
* They support various operations, such as concatenation, replication, and iteration.
* Both can be looped over using a **for** loop.

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

**Ans.13**

Tuples and lists are both sequence types in Python, but they have some key differences:

* Lists are mutable, meaning their elements can be modified, added, or removed after creation. Tuples, on the other hand, are immutable, and their elements cannot be changed once defined.
* Lists are defined using square brackets (**[]**), while tuples are defined using parentheses (**()**).
* Lists have a variable length, and their size can change dynamically. Tuples have a fixed length, and their size is determined at the time of creation.
* Lists use more memory compared to tuples because they have additional functionality due to their mutability.

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

**Ans.14** my\_tuple = (42,)

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

**Ans.15**

1. my\_list = [1, 2, 3]

my\_tuple = tuple(my\_list)

print(my\_tuple) **# Output: (1, 2, 3)**

(b) my\_tuple = (1, 2, 3)

my\_list = list(my\_tuple)

print(my\_list) **# Output: [1, 2, 3]**

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

**Ans.16** Variables that "contain" list values in Python are not actually lists themselves. Instead, they contain references or pointers to the list objects in memory. This means that the variable stores the memory address where the list is stored, allowing you to access and modify the list through that variable. This behavior is a result of Python's object-oriented nature and its use of references**.**

**For example, consider the following code:**

**my\_list = [1, 2, 3]**

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

Ans.17

* **copy.copy()**: This function performs a shallow copy of an object. It creates a new object and copies the references to the child objects in the original object. However, if the child objects are mutable, changes made to them will be reflected in both the original and copied objects.
* **copy.deepcopy()**: This function performs a deep copy of an object. It creates a completely independent copy of the original object and all its child objects. Even if the child objects are mutable, changes made to them in the copied object will not affect the original object.