1. What exactly is []?

**Ans.** [] this is an empty list value, which is a list value that contains no items.

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.)

**Ans.** spam = [2, 4, 6, 8, 10]

spam[2] = ‘hello’

spam

**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)]?

**Ans.** **‘d’** (Note that '3' \* 2 is the string '33', which is passed to int() before being divided by 11. This eventually evaluates to 3. Expressions can be used wherever values are used.)

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

**Ans.** The value of spam[-1] is **‘d’**.

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

**Ans.** The value of spam[:2] is **[‘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’)?

**Ans.** The value of bacon.index(‘cat’) is **1**.

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

**Ans.** bacon.append(99) change the look of the list value in bacon like :

[3.14, ‘cat’, 11, ‘cat’, True, 99].

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

**Ans.** bacon.remove(‘cat’) change the look of the list value in bacon like :

[3.14, 11, ‘cat’, True].

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

**Ans.** In Python, the list concatenation operator is ‘+’, which combines two or more lists into a single list. The list replication operator is ‘\*’, which creates a new list by repeating elements of a list specified number of times.

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

**Ans.** The ‘append()’ method used to add an item to the end of a list, while the ‘insert()’ method is used to insert an item at a specified index in the list. ‘append()’ only takes one argument, the item to be added, while ‘insert()’ takes two arguments, the index at which the item should be inserted, and the item to be inserted.

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

**Ans.** The two methods for removing items from a list are ‘list.remove(element)’ and ‘del list[index]’.

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

**Ans.** List values and string values are not identical, but they can have some similarities.

Both list values and string values are data structures used to store collections of items. Lists store elements that can be of any type, including other lists, while strings store sequences of characters.

One similarity between the two is that both lists and strings can be indexed, meaning individual elements can be accessed by their position in the collection. Additionally, both lists and strings can be sliced, meaning sub-collections of elements can be extracted from them.

However, there are also important differences between the two. Lists are mutable, meaning their contents can be changed, while strings are immutable, meaning their contents cannot be changed once created. Lists can store elements

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

**Ans.** Tuples and lists are both data structures in Python used to store collections of items. However, there are several differences between them:

**Immutability:** Tuples are immutable, meaning that once you create a tuple, you cannot change its contents. Lists are mutable, which means you can add, remove or modify elements after creation.

**Syntax:** Tuples use parentheses (), while lists use square brackets [].

**Accessing elements:** Elements in both tuples and lists can be accessed by indexing, but tuples are faster than lists when accessing elements.

**Use cases:** Tuples are used when you want to group together related data and ensure that the data remains unchanged. Lists are used when you need to manipulate data, or when you want to store a collection of elements that can change.

Overall, the choice between tuples and lists depends on your use case and the constraints you have in terms of data modification**.**

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

**Ans.** In Python, you can create a tuple that contains only the integer 42 as follows:

**my\_tuple = (42,)**

Note the comma after 42. This is necessary because without it, Python would interpret the expression ‘(42)’ as a parentheses-enclosed expression rather than a tuple. The comma creates a single-element tuple.

15. How do you get a list values tuple form? How do you get a tuple values list form?

**Ans.** To convert a list into tuple, I can use tuple() function:

my\_list = [1,2,3]

my\_tuple = tuple(my\_list)

To convert a tuple into list, I can use list() function:

my\_tuple = (1,2,3)

my\_list = list(my\_tuple)

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

**Ans**. Variables that "contain" list values contain references to lists, not the lists themselves.

In Python, variables do not actually contain values, they contain references to values. When you assign a list to a variable, you are creating a reference to that list, not a new copy of the list. This means that changes to the list will be reflected in the variable, and that multiple variables can refer to the same list.

**For example:**

my\_list = [1, 2, 3]

my\_other\_list = my\_list

**# Both my\_list and my\_other\_list refer to the same list**

my\_list is my\_other\_list **# returns True**

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

**Ans.** In Python, copy.copy() and copy.deepcopy() are two functions from the copy module that are used to create copies of objects.

The copy.copy() function creates a shallow copy of an object. A shallow copy is a copy of the object that references the original object's elements, rather than copying them. This means that changes to the elements of the original object will be reflected in the shallow copy, and vice versa.

The copy.deepcopy() function creates a deep copy of an object. A deep copy is a complete copy of the object, including all of its elements. Changes to the original object will not affect the deep copy, and vice versa.

In general, you should use copy.deepcopy() when you want to create a completely independent copy of an object, and copy.copy() when you want to create a copy that references the original object's elements. The choice between the two will depend on the specific requirements of your use case.