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

Ans. This denotes creation of a list, and it contains values of a particular 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.)**

Ans. Spam[3]=”hello”

**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’

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

Ans. ‘d

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

Ans. [‘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. 1

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

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

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

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

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

Ans. Concatenation is the process of joining the elements of a particular data-structure in an end-to-end manner. Concatenation is done between the same data types only. ‘+’ operator is used to concatenate two lists. Eg. L1=[1,2,3] and l2=[4,5,6], then l1+l2 will concatenate both giving [1,2,3,4,5,6].

Sequences datatypes (both mutable and immutable) support a **repetition**operator \* The repetition operator \* will make multiple copies of that particular object and combines them together. When \* is used with an integer it performs multiplication but with list, tuple or strings it performs a repetition.eg. l=[1,2,3], then l\*2 will give [1,2,3,1,2,3].

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

Ans. append() will add the value passed in it at the end of the list. Syntax – l.append(object)

While insert() will add the value passed in it at the index value location passed with it. Syntax – l.insert(index, object).

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

Ans. clear(): to remove all items

Pop(): to remove an item by index and return get its value

Remove(): to remove an item by value

Del l[]: to remove items by index or slice

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

Ans. Strings can be defined as sequential collections of characters enclosed between ‘’ or “”.

A **list** is a sequential collection of Python data values, where each value is identified by an index. The values that make up a list are called its **elements**. Lists are similar to strings, which are ordered collections of characters, except that the elements of a list can have any type and for any one list, the items can be of different types.

Note: list are mutable and string are immutable.

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

Ans. A **tuple**, like a list, is a sequence of items of any type. The printed representation of a tuple is a comma-separated sequence of values, enclosed in parentheses. In other words, the representation is just like lists, except with parentheses () instead of square brackets [].

The key difference between lists and tuples is that a tuple is immutable

Note: To create a tuple with a single element (but you’re probably not likely to do that too often), we have to include the final comma, because without the final comma, Python treats the (5) below as an integer in parentheses

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

**Ans.** t =(42,), the use of “,” is must otherwise single value inside tuple will be considered as an integer value.

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

Ans. suppose l is a list, then to get a tuple form – tuple(l)

Suppose t is a tuple, then to get a list form – list(t)

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

Ans. The list values inside a variable could be of different data types, it could be a string, int, list, tuple, set, dictionary etc

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

Ans. Essentially, sometimes you may want to have the original values unchanged and only modify the new values or vice versa. In Python, there are two ways to create copies:

1. Shallow Copy
2. Deep Copy

To make use of these operations we import copy module.

**Shallow copy**: it creates a new object which store the reference of the original elements.

It does not create a copy of nested objects, instead it just copies the reference of nested objects. This means , a copy process does not recurse or create copies of nested objects itself.

it will create new and independent object with same content

so in this the new list contains references to original nested objects stored in old list. Then we add the new list into old list. This new sublist was not copied to the new list. However, when you change any nested objects in old list , the changes appear in new list. This is because, both lists share the reference of same nested objects.

**Deep copy**: creates a new object and recursively adds the copies of nested objects present in the original elements. It creates independent copy of original object and all its nested objects. Any changes to any nested objects in original object i.e., old list , you will see no changes to the copy i.e., new list. And also if any new value is assigned to old list , only old list will be modified. This means both the old list and the new list are independent.