1. What exactly is []?

:- [] is another way of defining an empty list in python. eg. x = []

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.)

:- spam[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)]?

:- d

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

:-d

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

:-['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')?

:1

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

:- [3.14, 'cat', 11, 'cat', True, 99]

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

:- [3.14, 11, 'cat', True, 99]

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

:- Two lists can be concatenated with + operator whereas list can be replicated with \* operator

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

:- append() method will add item at the end of the list whereas insert() method adds item at the given index in the list.

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

:- list.remove() (removes first matching item from list) and list.pop() (removes item from the given index from the list) are the two methods for removing items from a list.

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

:- Both are sequences: Strings and lists are both sequences of items in Python. A string is a sequence of characters, while a list is a sequence of any data type.

Indexing and slicing: Both strings and lists can be indexed and sliced. You can access individual elements in a string or list using their index, and you can extract a subset of a string or list using slicing.

Concatenation: Both strings and lists can be concatenated using the + operator. For example, "hello" + "world" produces the string "helloworld", and [1, 2, 3] + [4, 5, 6] produces the list [1, 2, 3, 4, 5, 6].

Replication: Both strings and lists can be replicated using the \* operator. For example, "hello" \* 3 produces the string "hellohellohello", and [1, 2, 3] \* 2 produces the list [1, 2, 3, 1, 2, 3].

Membership: You can use the in operator to check if an item is present in a string or list. For example, "e" in "hello" returns True, and 2 in [1, 2, 3] returns True.

Length: Both strings and lists have a len() function that returns the number of items in the sequence.

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

:- Mutability: Lists are mutable, which means that you can add, remove, or modify elements in a list after it has been created. Tuples, on the other hand, are immutable, which means that once a tuple is created, you cannot add, remove, or modify elements in it.

Syntax: Lists are enclosed in square brackets [], while tuples are enclosed in parentheses ().

Use case: Lists are generally used for storing collections of related items where the order of the items may change, and you need to modify the list as you go along. Tuples are often used for grouping related data together, where the individual elements have a specific meaning, and the order of the elements is important.

Performance: Since tuples are immutable, they can be faster to access and process than lists in certain situations. Tuples also use less memory than lists, which can be beneficial in situations where memory usage is a concern.

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

:- my\_tuple = (42,) (Here comma is necessary to define tuple with only one item)

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

:- To get a list value's tuple form, you can use the tuple() function. eg tuple([1,2,3])

To get a tuple value's list form, you can use the list() function. eg. list((1,2,3))

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

:- variables that "contain" list values in Python actually contain references to list objects in memory, rather than the list values themselves

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

:- copy.copy(): This function creates a shallow copy of an object. A shallow copy creates a new object, but it does not create new copies of the nested objects within the original object.

copy.deepcopy(): This function creates a deep copy of an object. A deep copy creates a new object and also creates new copies of all the nested objects within the original object.