1. What exactly is []?

ANS :- In Python, [] represents an empty list. It is a built-in data type used to store a collection of items. A list is an ordered sequence of elements, and [] denotes an empty list with 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 :- i ) spam = [2, 4, 6, 8, 10]

spam[2] = 'hello'

print(spam)

ii ) spam = ['a', 'b', 'c', 'd']

spam[2] = 'hello'

print(spam)

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

ANS :- list i.e. spam[3]

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

ANS :- It will return the last element of list spam

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

ANS :- spam[:2] would return a new list containing the elements from the start of the list up to (but not including) the element at index 2. So, it would include elements at indices 0 and 1. Therefore, the value of spam[:2] would be [1, 2].

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, 11, 'cat,' True, 99]

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

ANS :- The list concatenation operator in Python is `+`, and it is used to combine two or more lists into a single list.The list replication operator in Python is `\*`, 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, whereas the insert() method is used to insert an element at a specific position in a list.

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

ANS :- The two methods for removing items from a list are:

1. remove(): Removes the first occurrence of a specific value from the list.

2. pop(): Removes an element at a specified index from the list and returns its value

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

ANS :- List values and string values share some similarities in how they are structured and accessed: 1. Sequence of Elements: Both lists and strings are sequences of elements. In a list, the elements can be of any data type, while in a string, the elements are individual characters. 2. Indexing: Elements in both lists and strings can be accessed using indexing. Each element has a specific index indicating its position within the sequence. Indexing starts from 0 for the first element. 3. Slicing: Both lists and strings support slicing, which allows you to extract a portion of the sequence by specifying a range of indices. 4. Iteration: We can iterate over both lists and strings using loops to process each element or character sequentially

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

ANS :- The main differences between tuples and lists in Python are as follows:

1. Mutability: Tuples are immutable, while lists are mutable. This means that the elements of a tuple cannot be modified once it is created, whereas elements in a list can be modified, added, or removed.

2. Syntax: Tuples are defined using parentheses `()`, while lists are defined using square brackets `[]`. For example, `(1, 2, 3)` is a tuple, whereas `[1, 2, 3]` is a list.

3. Usage: Tuples are typically used for grouping related values together, where the order and values themselves are significant. Lists, on the other hand, are more versatile and commonly used when you need to store a collection of items that may change or require modification.

4. Operations: Lists offer more flexibility and functionality compared to tuples. Lists have various built-in methods for adding, removing, and modifying elements. Tuples have fewer methods due to their immutability.

5. Performance: Tuples are generally more memory-efficient and faster to process compared to lists. This is because tuples are immutable, allowing for optimizations in memory allocation and data processing.

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

ANS :- To create a tuple value that only contains the integer 42, we can use the following syntax:

```python

my\_tuple = (42,)

```

Note the presence of a comma `,` after the integer 42. It is important to include the comma even if the tuple contains only one element. This distinguishes the tuple from a simple integer value enclosed in parentheses, ensuring it is recognized as a tuple data type.

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

ANS :- To convert a list value to its tuple form, we can use the `tuple()` function. Here's an example:

```python

my\_list = [1, 2, 3, 4, 5]

my\_tuple = tuple(my\_list)

print(my\_tuple)

```

Output:

```

(1, 2, 3, 4, 5)

```

The `tuple()` function takes an iterable, such as a list, and converts it into a tuple.

To convert a tuple value to its list form, we can use the `list()` function. Here's an example:

```python

my\_tuple = (1, 2, 3, 4, 5)

my\_list = list(my\_tuple)

print(my\_list)

```

Output:

```

[1, 2, 3, 4, 5]

```

The `list()` function takes an iterable, such as a tuple, and converts it into a list.

By using these functions, we can easily convert between list and tuple forms in Python.

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

ANS :- Variables that "contain" list values in Python actually contain references or pointers to the list objects rather than the lists themselves. In other words, the variable holds the memory address where the list is stored in the computer's memory.

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

ANS :- The copy.copy() and copy.deepcopy() functions are part of the copy module in Python, and they serve different purposes when it comes to creating copies of objects.

1. copy.copy():

2. copy.deepcopy():

The copy.copy() function creates a shallow copy of an object. It creates a new object with a new memory address but references the same elements as the original object. If the object being copied contains other objects (e.g., a list with nested lists), the copy will refer to the same nested objects as the original. Modifying the nested objects in the copy or the original will affect both.

The copy.deepcopy() function creates a deep copy of an object. It creates a completely independent copy with new memory addresses for all elements, including nested objects. All nested objects are recursively copied, creating new instances that are independent of the original object. Modifying the nested objects in the deep copy or the original will not affect each other