1.What exactly is []?

[] represents an empty list in Python. It is a data structure that can hold an ordered sequence of elements. An empty 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.

spam = [2, 4, 6, 8, 10]

spam[2] = 'hello'

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

int('3' \* 2) is equivalent to int('33'), which is 33. Then, 33 / 11 is 3. So, spam[3] would access the fourth element in the list 'a', 'b', 'c', 'd', which is 'd'.

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

spam[-1] accesses the last element of the list, which is 'd'.

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

Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.

spam[:2] returns a new list containing the elements from the beginning (index 0) up to, but not including, index 2. So, it will be ['a', 'b'].

6. What is the value of bacon.index('cat')?

bacon.index('cat') returns the index of the first occurrence of 'cat' in the list bacon. In this case, it would be 1 because 'cat' is at index 1 in the list.

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

bacon.append(99) adds the value 99 to the end of the list bacon, so the list will now be [3.14, 'cat', 11, 'cat', True, 99].

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

bacon.remove('cat') removes the first occurrence of 'cat' from the list bacon. After this operation, the list will become [3.14, 11, 'cat', True, 99].

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

* List Concatenation: The + operator is used for concatenating lists. For example, list1 + list2 combines the elements of list1 and list2 into a new list.
* List Replication: The \* operator is used for replicating a list. For example, list1 \* 3 creates a new list with the elements of list1 repeated three times.

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

* append(): Appends an element to the end of the list.
* insert(): Inserts an element at a specific index in the list, allowing you to specify the position of the new element.

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

* remove(): Removes the first occurrence of a specified value.
* pop(): Removes an element at a specified index and returns the removed element.

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

List values and string values are similar in that both can be indexed and sliced. They are ordered sequences of elements, and you can use indices to access specific elements within them. However, lists can hold a variety of data types, while strings are collections of characters.

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

* Lists are mutable, meaning you can change their elements after creation.
* Tuples are immutable, meaning their elements cannot be changed after creation.
* Lists are defined with square brackets [...], while tuples use parentheses (...).
* Lists are typically used for collections of items, and tuples are often used for fixed collections or for representing coordinate points.

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

my\_tuple = (42,)

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, use the tuple() constructor. For example: tuple(my\_list).
* To get a tuple value's list form, use the list() constructor. For example: list(my\_tuple).

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

Variables that "contain" list values store references or pointers to the list in memory, not the list data itself. This means that changes made to the list through one variable will be reflected in all variables that reference the same list.

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

* copy.copy(): Creates a shallow copy of an object. For lists, this means that the new list will be a copy of the original list, but it will still reference the same objects within (not a deep copy of elements).
* copy.deepcopy(): Creates a deep copy of an object. For lists, it recursively copies all elements and their nested elements, resulting in a completely independent copy.