1. What exactly is []?

- An Empty list

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.insert(2, 'hello)   # Since, we have to **assign** 'hello' in third place
- spam[2] = 'hello'   # We can also do this, if we have to **reassign** the value 'hello' as the third place. Since, strings are mutable.

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

- It is equal to spam[3] that equals 'd'

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

- '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.

- ['a', 'b']

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

- It returns the index value of the string "cat", if string 'cat' is present inside bacon object.
- If string cat is present more than once in bacon, then it returns the index value of first instance of the string cat in the list.
- Since, bacon is not defined. No one can answer this question with a proper defined value.

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

- Since, it is mentioned that bacon is a list. Integer object of 99 will be added as the last element in the list bacon.
- We call 99 using bacon[-1]

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

- It will remove the string cat from the list bacon. If there are more than 2 instances of string cat in that list, then it will remove the first instance.

Note: Looks like all the above three questions are based on some invisible bacon list. Without specifying that list, it is not possible to answer these questions with proper values, that's why I answered with clear explanations and that should work for these questions. If you want answers with definite values, you should mention the values in the list bacon, without that, the way I answered is the only best way possible to answer.

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

- + sign between two list objects is the concatenation operator.
- * sign between a list and a +ve integer is a replication operator.

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

- Append is used to add an element in the last place of a list
- Insert used to add an element at a specific index value, it won't replace the existing value. It will just replace the index value of that existing value.

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

- Remove and pop
- Remove takes the value that we want to remove
- Pop takes the index value of the element we want to remove. By default, it removes the last element.

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

- There are a lot of same operations and methods can be used on both the string and list objects.
- We can do indexing and slicing on both of them
- We can use if/else statements on both the strings and lists
- We can use same concatenation and replications operators in the same way for both strings and lists.
- We can use methods like len() on both strings and lists.
- We can use for loops and while loops on both the string and lists because they are ordered with index positions.

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

- Tuples are immutable
- Lists are mutable
- In terms of representation, lists use [] and tuples use ()

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

- (42,)

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

- By passing the tuple in list()
- By passing the list in tuple()

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

- They are variables stored in main memory, with reference to that list that is present in heap memory.

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

- Copy.copy() is shallow copy and copy.deepcopy() is deep copy of list copy.
- Shallow copy stores the object references in the original memory. While deep copy stores copies of the object’s value.
- Shallow copy reflects the changes made to the new/copied object in the original object. And, deep copy doesn’t reflect the changes made to the new/copied in the original object.
- Shallow copy only stores the copy of the original object and points the references to the objects, while deep copy stores the copy of the original object and recursively copies the objects as well. Because of this, shallow copy is comparatively faster than deep copy.