1. What exactly is []?  
  
**Answer:** [] represents an empty list. A list is a collection of items that can be of any data type, such as integers, strings, or other lists. An empty list is simply a list with no items in it.

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.  
  
**Answer:** To assign the value 'hello' as the third value in a list stored in the variable spam:

spam[2] = 'hello'

After executing this code, the spam list would be [2, 4, 'hello', 8, 10].

If spam includes the list ['a', 'b', 'c', 'd'], the same code would work to assign 'hello' as the third value:

spam[2] = 'hello'

After executing this code, the spam list would be ['a', 'b', 'hello', 'd']

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

4. What is the value of spam[-1]?  
  
**Answer: ‘**d’

5. What is the value of spam[:2]?  
  
**Answer:** [ ‘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')?

**Answer:** 1

7. How does bacon.append(99) change the look of the list value in bacon?  
**Answer:** [3.14, 'cat', 11, 'cat', True, 99]

8. How does bacon.remove('cat') change the look of the list in bacon?  
**Answer:** [3.14, 11, 'cat', True, 99]

9. What are the list concatenation and list replication operators?  
  
**Answer:** The list concatenation operator is +, and the list replication operator is \*.

The + operator can be used to concatenate two or more lists together.

The \* operator can be used to create a new list that is a replication of an existing list.

10. What is difference between the list methods append() and insert()?  
**Answer:** The append() method adds an element to the end of the list.The insert() method, on the other hand, adds an element at a specific position in the list, shifting the existing elements to make room for the new one.

11. What are the two methods for removing items from a list?  
**Answer:** In Python, there are two main methods for removing items from a list:

The remove() method: This method removes the first occurrence of a specified item from the list. The pop() method: This method removes an element from the list at a specified index and returns its value.

12. Describe how list values and string values are identical.  
**Answer:** List values and string values in Python are both sequences of individual elements, where each element is identified by its position or index within the sequence. Both types of values can be accessed and manipulated using similar syntax and methods.

Specifically, both lists and strings:

Can be indexed and sliced: You can access individual elements or subsets of elements within both lists and strings using square brackets and index numbers. For example, my\_list[0] and my\_string[0] both retrieve the first element of the list and string, respectively. Additionally, both types support slicing using the syntax [start:stop:step].

Can be iterated over with a for loop: You can use a for loop to iterate over each element of both lists and strings. For example:

my\_list = [1, 2, 3]

for element in my\_list:

print(element)

# Output: 1

# 2

# 3

my\_string = "hello"

for char in my\_string:

print(char)

# Output: h

# e

# l

# l

# o

Support concatenation and replication: Both types support concatenation using the + operator, which allows you to combine two or more lists or strings into a new list or string. Additionally, both types support replication using the \* operator, which creates a new list or string by repeating the original list or string a specified number of times.

Have a length that can be obtained using the len() function: You can use the len() function to get the number of elements in both lists and strings. For example, len(my\_list) and len(my\_string) both return the length of the list and string, respectively.

Because of these similarities, many operations that work on lists also work on strings, and vice versa. However, it's important to note that there are also important differences between lists and strings, such as the fact that lists are mutable (can be changed), while strings are immutable (cannot be changed).

13. What's the difference between tuples and lists?  
**Answer:** Tuples and lists are both used to store collections of data, but there are several key differences between them.

1.Mutability: The main difference between tuples and lists is that tuples are immutable, meaning that their values cannot be changed once they are created, while lists are mutable, meaning that their values can be changed.

2.Syntax: Tuples are defined using parentheses, while lists are defined using square brackets.

3.Size: Tuples tend to be used for fixed-sized collections of data, while lists tend to be used for variable-sized collections of data. This is because tuples are immutable and therefore cannot be resized after they are created. Lists, on the other hand, can be resized using methods such as append() and remove().

4.Performance: Tuples are generally more performant than lists for small collections of data, since they require less memory and are faster to create. However, for larger collections of data, lists may be faster and more efficient since they can be resized as needed.

14. How do you type a tuple value that only contains the integer 42?  
**Answer:** my\_tuple = (42,)

15. How do you get a list value's tuple form? How do you get a tuple value's list form?  
**Answer:** To get a list value's tuple form, you can simply pass the list to the tuple() function. For example: my\_list = [1, 2, 3]

my\_tuple = tuple(my\_list)

print(my\_tuple) # Output: (1, 2, 3)

16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?  
**Answer:** Variables that "contain" list values in Python are actually just references to the list objects in memory. In other words, the variable itself does not contain the list values directly; instead, it contains a reference to the location in memory where the list is stored.

17. How do you distinguish between copy.copy() and copy.deepcopy()?  
**Answer:** copy.copy() and copy.deepcopy() are both methods in the Python copy module that can be used to create copies of objects, including lists and tuples. The main difference between these two methods is the level of copying that is performed.

copy.copy() creates a shallow copy of an object, which means that it creates a new object with the same contents as the original object, but any mutable objects within the original object are still shared between the original and copied objects.

On the other hand, copy.deepcopy() creates a deep copy of an object, which means that it creates a completely new object with the same contents as the original object, and any mutable objects within the original object are also copied recursively.