1. What exactly is []?

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.

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

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

1. [] is an empty list in Python.
2. You can assign the value 'hello' as the third value in the spam list by using the following code: spam[2] = 'hello'
3. The value of spam[int(int('3' \* 2) / 11)] is 'd'. Here, the expression int('3' \* 2) evaluates to 33, and dividing it by 11 gives 3. The integer index 3 corresponds to the fourth element in the spam list, which is 'd'.
4. The value of spam[-1] is 'd'. This returns the last element of the spam list.
5. The value of spam[:2] is ['a', 'b']. This is a list slice that includes the first two elements of the spam list.
6. The value of bacon.index('cat') is 1. This returns the index of the first occurrence of the string 'cat' in the bacon list.
7. bacon.append(99) adds the value 99 to the end of the bacon list, so the new value of bacon is [3.14, 'cat', 11, 'cat', True, 99].
8. bacon.remove('cat') removes the first occurrence of the string 'cat' from the bacon list, so the new value of bacon is [3.14, 11, 'cat', True].
9. The list concatenation operator is '+', which combines two lists into a single list. The list replication operator is '\*', which creates a new list by repeating an existing list a specified number of times.
10. The append() method adds a new element to the end of a list, while the insert() method adds a new element at a specific position in the list.
11. The two methods for removing items from a list are remove() and pop(). The remove() method removes the first occurrence of a specified element from the list, while the pop() method removes and returns the element at a specified position in the list (or the last element if no position is specified).
12. Both list values and string values are ordered sequences of elements that can be accessed using indexing and slicing. However, list values can contain any data type, including other lists, while string values can only contain characters.
13. Lists and tuples are both ordered sequences of elements. However, lists are mutable (i.e., their values can be changed), while tuples are immutable (i.e., their values cannot be changed).
14. You can type a tuple value that only contains the integer 42 by using the following code:

my\_tuple = (42,)

1. To get a list value's tuple form, you can use the tuple() function, like this: tuple(my\_list). To get a tuple value's list form, you can use the list() function, like this: list(my\_tuple).
2. Variables that "contain" list values actually contain references to the list objects in memory. When you assign a list to a variable, you are creating a reference to that list, not a copy of the list.

17.

In Python, the **copy** module provides two functions for copying objects: **copy()** and **deepcopy()**. The main difference between the two methods is that **copy.copy()** creates a shallow copy of an object, while **copy.deepcopy()** creates a deep copy of the object.

A shallow copy creates a new object, which is a copy of the original object. However, if the original object contains any mutable objects, such as lists or dictionaries, then the new object will still refer to the same mutable objects as the original object. Changes made to the mutable objects in the new object will also be reflected in the original object.

On the other hand, a deep copy creates a completely independent copy of an object and its contents, including all nested objects. In other words, a deep copy creates a new object with a new memory address, and any changes made to the copy will not affect the original object.

Here's an example to illustrate the difference:

import copy

# Original list

original\_list = [1, 2, [3, 4]]

# Shallow copy

shallow\_copy = copy.copy(original\_list)

shallow\_copy[2][0] = 5

print(original\_list) # Output: [1, 2, [5, 4]]

print(shallow\_copy) # Output: [1, 2, [5, 4]]

# Deep copy

deep\_copy = copy.deepcopy(original\_list)

deep\_copy[2][0] = 6

print(original\_list) # Output: [1, 2, [5, 4]]

print(deep\_copy) # Output: [1, 2, [6, 4]]

In the above example, we first create a list **original\_list** that contains an integer and another list. We then create a shallow copy of **original\_list** using **copy.copy()** and a deep copy using **copy.deepcopy()**.

Next, we modify the first element of the nested list in both the shallow and deep copies. As you can see from the output, the change made to the nested list in the shallow copy is also reflected in the original list, while the change made to the nested list in the deep copy does not affect the original list. This is because the shallow copy still refers to the same nested list object as the original list, while the deep copy creates a new object for the nested list.