1. What exactly is []?

Ans: It represents 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.)

Ans: spam.insert(2,”hello”)

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)]?

Ans: d

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

Ans: d

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

Ans: ['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')?

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 operator for list concatenation is +, while the operator for replication is *. (This is the same as for strings.)

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

Ans: append() will put the values at the end of list

Insert() can put the value anywhere in the list given the index of list

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

Ans: remove(element name) & pop(element index)

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

Ans: Both can be indexed, concatenated and replicated

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

Ans: List are mutable while tuples are immutable.

Lists represented with [] and tuples with ()

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

Ans: t = (42,) trailing comma is compulsory, if we don't give comma then it is integer

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

Ans: t = tuple(list) and l = list(tuple)

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

Ans:

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
```

```
>>> cheese
42
```

You assign 42 to the spam variable, and then you copy the value in spam and assign it to the variable cheese. When you later change the value in spam to 100, this doesn't affect the value in cheese. This is because spam and cheese are different variables that store different values.

But lists don't work this way. When you assign a list to a variable, **you are actually assigning a list *reference* to the variable.** A reference is a value that points to some bit of data, and **a list reference is a value that points to a list.** Here is some code that will make this distinction easier to understand. Enter this into the interactive shell:

```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam
❸ >>> cheese[1] = 'Hello!'
  >>> spam
  [0, 'Hello!', 2, 3, 4, 5]
  >>> cheese
  [0, 'Hello!', 2, 3, 4, 5]
```

This might look odd to you. The code changed only the cheese list, but it seems that both the cheese and spam lists have changed.

When you create the list ❶, you assign a reference to it in the spam variable. But the next line ❷ **copies only the list reference in spam to cheese, not the list value itself**. This means the values stored in spam and cheese now both refer to the same list. **There is only one underlying list because the list itself was never actually copied.** So, when you modify the first element of cheese ❸, you are modifying the same list that spam refers to.

Variables will contain references to list values rather than list values themselves. But for strings and integer values, variables simply contain the string or integer value. Python uses references whenever variables must store values of mutable data types, such as lists or dictionaries. For values of immutable data types such as strings, integers, or tuples, Python variables will store the value itself

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

Ans: Eg of **deep copy**

```
import copy

l1 = [1,2, [3,4],5]

l2 = copy.deepcopy(l1)

l2[2][1] = 5

print(l1)

print(l2)
```

Output: l1 = [1, 2, [3, 4], 5] l2 = [1, 2, [3, 5], 5]

In case of deep copy the element of **the original remains unchanged**

Shallow copy

```
import copy

l1 = [1,2, [3,4],5]

l2 = copy. copy(l1)

l2[2][1] = 5

print(l1)

print(l2)
```

Output: l1 = [1, 2, [3, 5], 5] l2 = [1, 2, [3, 5], 5]

In case of shallow copy the element of **the original changes**