1.What exactly is []?

[] is an empty list. A list is a data structure that can store multiple items of data. The items in a list are separated by commas and enclosed in brackets. An empty list is a list that does not contain any items.

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] = 'hello'

- To add the value 'hello' as the third value in the list spam, we can use the assignment operator (=) to assign the value 'hello' to the third element of the list. The third element of the list is spam[2].

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

- The value of spam[int(int('3' * 2) / 11)] is 'c'.
- To calculate the value of spam[int(int('3' * 2) / 11)], we first need to calculate the value of int('3' * 2). This is equal to 3 * 2 = 6. We then need to divide this value by 11, which is equal to 6 / 11 = 0.5454545454545454. We then need to round this value to the nearest integer, which is 0.
- We then need to use this value to index into the list spam. The index of the first element in the list is 0, so the index of the second element is 1, and so on. The index of the element we are looking for is 0, so the value of spam[int(int('3' * 2) / 11)] is 'c'.

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

- The value of spam[-1] is 'd'.
- The list spam contains the elements 'a', 'b', 'c', and 'd'. The index of the first element in the list is 0, the index of the second element is 1, and so on. The index of the last element in the list is -1. Therefore, the value of spam[-1] 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.

- The value of bacon[:2] is [3.14, 'cat'].
- The list bacon contains the elements 3.14, 'cat', 11, 'cat', and True. The slice bacon[:2] returns the first two elements of the list, which are 3.14 and 'cat'. What is

the value of bacon[1:]? The value of bacon[1:] is ['cat', 11, 'cat', True]. The list bacon contains the elements 3.14, 'cat', 11, 'cat', and True. The slice bacon[1:] returns the elements from the second element to the end of the list, which are 'cat', 11, 'cat', and True.

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

- The value of bacon.index('cat') is 1.
- The list bacon contains the elements 3.14, 'cat', 11, 'cat', and True. The index() method returns the index of the first occurrence of the specified element in the list. In this case, the first occurrence of the element 'cat' is at index 1.

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

- The append() method adds the specified element to the end of the list. In this case, the element 99 is added to the end of the list, resulting in the list [3.14, 'cat', 11, 'cat', True, 'dog', 99].

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

- The remove() method removes the first occurrence of the specified element from the list. In this case, the first occurrence of the element 'cat' is at index 1, so the list is changed to [3.14, 11, 'cat', True, 'dog', 99].

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

- The list concatenation operator is the plus sign (+). It is used to combine two or more lists into one list. For example, the following code creates a list called fruits that contains the elements 'apple', 'banana', and 'cherry':
- fruits = ['apple', 'banana', 'cherry']
- The list replication operator is the asterisk (*). It is used to repeat a list a specified number of times. For example, the following code creates a list called numbers that contains the elements 1, 2, 3, 4, 5, 6, 7, 8, and 9:
- numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9] * 3
- The list concatenation and list replication operators are very useful for creating lists of data.

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

- The append() method adds an element to the end of a list, while the insert() method adds an element to a specific position in a list. For example, the following code adds the element 'dog' to the end of the list fruits:

- fruits.append('dog')

- The following code adds the element 'dog' to the beginning of the list fruits

- : fruits.insert(0, 'dog')
- The append() method is more commonly used, as it is simpler to use. However, the insert() method can be useful if you need to add an element to a specific position in a list

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

There are two methods for removing items from a list:

- the remove() method and the pop() method. The remove() method removes the first occurrence of the specified element from the list.
- For example, the following code removes the element 'cat' from the list fruits: fruits.remove('cat')

- The pop() method removes the element at the specified index from the list.
- For example, the following code removes the element at index 0 from the list fruits: fruits.pop(0)
- The remove() method is more commonly used, as it is simpler to use. However, the pop() method can be useful if you need to remove an element from a specific position in the list.

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

- List values and string values are not identical. A list is a collection of items, while a string is a sequence of characters. Lists can contain different types of items, while strings can only contain characters

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

- A tuple is a collection of items, just like a list. However, tuples are immutable, which means that they cannot be changed after they are created. Lists, on the other hand, are mutable, which means that they can be changed after they are created.
- Tuples are often used to store data that will not change, such as the coordinates of a point on a map. Lists are often used to store data that will change, such as the items in a shopping cart.

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

- To type a tuple value that only contains the integer 42, you can use the following syntax:
- (42,)
- The parentheses indicate that the value is a tuple, and the comma after the integer 42 indicates that it is the only element in the tuple.

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, you can use the tuple() function. For example, the following code converts the list [1, 2, 3] to a tuple:
- tuple([1, 2, 3])
- To get a tuple value's list form, you can use the list() function. For example, the following code converts the tuple (1, 2, 3) to a list:
- list((1, 2, 3))

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

- Variables that "contain" list values are not necessarily lists themselves. Instead, they can contain references to lists. A reference is a way of pointing to a value in memory. When you assign a list to a variable, you are creating a reference to that list. The variable does not actually contain the list itself. It only contains a reference to the list.
- For example, the following code creates a list and assigns it to the variable my_list:
- my_list = [1, 2, 3]
- The variable my_list now contains a reference to the list [1, 2, 3]. If you change the list [1, 2, 3], the variable my_list will also change. This is because the variable my_list is still pointing to the same list.
- References are very useful because they allow you to store values in variables without having to copy them. This can save memory and improve performance.

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

- The copy.copy() function creates a shallow copy of an object. This means that the new object will have the same values as the original object, but it will not have its own copy of the object's contents. If the object contains other objects, those objects will be shared between the original and the copy.

- The copy.deepcopy() function creates a deep copy of an object. This means that the new object will have the same values as the original object, and it will also have its own copy of the object's contents. If the object contains other objects, those objects will also be copied. Here is an example of the difference between copy.copy() and copy.deepcopy():

```
>>> a = [1, 2, 3]
>>> b = copy.copy(a)
>>> c = copy.deepcopy(a)
>>> a[0] = 4
>>> print(b)
    [1, 2, 3]
>>> print(c)
```

[4, 2, 3]

- In this example, the copy.copy() function creates a shallow copy of the list a. This means that the new list b has the same values as the original list a, but it does not have its own copy of the list's contents. When the value of a[0] is changed, the value of b[0] is also changed.
- The copy.deepcopy() function creates a deep copy of the list a. This means that the new list c has the same values as the original list a, and it also has its own copy of the list's contents. When the value of a[0] is changed, the value of c[0] is not changed.
- In general, you should use copy.copy() when you need to create a new object that has the same values as an existing object, but you do not need to make any changes to the original object. You should use copy.deepcopy() when you need to create a new object that has the same values as an existing object, and you need to make changes to the new object without affecting the original object.