1. What exactly is []?

[] is an empty list in Python. So, it doesn’t hold any elements. Lists are mutable, which means you can add, remove, or modify elements in the list. You can add elements to the empty list using various methods like append (), insert (), or concatenation. An empty list can thus be later populated.

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.)

spam = [2, 4, 6, 8, 10]

spam.insert(2, 'hello')

print(spam)

output:

[2, 4, 'hello', 6, 'hello', 8, 10]

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)]?

The expression int('3' \* 2) evaluates to the string '33', which is then converted to an integer value 33. The expression int('33') / 11 gives the value 3. spam[3] is executed, which accesses the element at index 3 in the spam list.

Output: ‘d’

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

Output: ‘d’

Accesses the first element from last.

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

Accesses all elements upto index ‘2’. Ie, index ‘0’ and index ‘1’.

Output:

['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')?

The index() method in Python returns the index of the first occurrence of a specified value in a list. Though ‘cat’ occurs twice, only first occurrence is considered.

Output: 1

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

Adds the element 99 at the end of the list to get the following output:

[3.14, 'cat', 11, 'cat', True, 99]

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

The remove () method in Python removes the first occurrence of a specified value from a list. It searches for the first occurrence of 'cat' in the list and removes it.

Output:

[3.14, 11, 'cat', True, 99]

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

The operators + and \* to concatenate and replicate lists.

Concatenation:

list1 = [1, 2, 3]

list2 = [4, 5, 6]

new\_list = list1 + list2

print(new\_list)

Output:

[1, 2, 3, 4, 5, 6]

Replication:

list1 = [1, 2, 3]

replist = [1, 2, 3] \* 2

print(replist)

Output:

[1, 2, 3, 1, 2, 3]

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

The append () and insert () methods in Python are used to add elements to a list, but they differ in how they add elements and where the elements are inserted within the list.

1. append () method:

list1 = [1, 2, 3]

list1.append(4)

print(list1)

output:

[1, 2, 3, 4]

1. insert () method:

my\_list = [1, 2, 3]

my\_list.insert(1, 4)

print(my\_list)

output:

[1, 4, 2, 3]

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

1. remove() method:

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

my\_list.remove(2)

print(my\_list)

output:

[1, 3, 2, 4]

1. del statement:

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

del my\_list[2]

print(my\_list)

output:

[1, 2, 4]

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

Similarities:

* The similarity between Lists and Strings in Python is that both are **sequences**. A string is a character's sequence between single or double quotations. A list is an item sequence in which each item could be anything such as a float, an integer, a string, etc.
* The second similarity between Strings and Lists is that both strings and lists have **lengths**. A length of a string is the number of characters in the string and a length of a list is the number of items in the list.
* The third similarity between Strings and Lists is that both strings and lists have a **position**. Every character in a string as well as every item in a list has a position which is known as an index.

Differences:

* The differences between them are that firstly, Lists are mutable but Strings are immutable.
* Secondly, elements of a list can be of different types whereas a String only contains characters that are all of String type.

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

|  |  |
| --- | --- |
| **Lists** | **Tuples** |
| It is mutable | It is immutable |
| The implication of iterations is time-consuming in the list. | Implications of iterations are much faster in tuples. |
| Operations like insertion and deletion are better performed. | Elements can be accessed better. |
| Consumes more memory. | Consumes less memory. |
| Many built-in methods are available. | Does not have many built-in methods. |
| Unexpected errors and changes can easily occur in lists. | Unexpected errors and changes rarely occur in tuples. |

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

Tuple1 = (42,)

The comma at the end is essential to differentiate it from the integer itself, which would be interpreted as just a value without being a part of a tuple. By including the trailing comma, you indicate that you want to create a tuple with a single element.

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

An iterable can be passed as an input to the tuple () function, which will convert it to a tuple object. If you want to convert a Python list to a tuple, you can use the tuple() function to pass the full list as an argument, and it will return the tuple data type as an output.

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

my\_tuple = tuple(my\_list)

print(my\_tuple)

output:

(1, 2, 3, 4)

Similarly, to convert a tuple to a list, you can use the list () function. It takes an iterable (such as a tuple) as an argument and returns a list containing the same elements.

my\_tuple = (1, 2, 3, 4)

my\_list = list(my\_tuple)

print(my\_list)

output:

[1, 2, 3, 4]

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

Variables that "contain" list values in Python actually store references to list objects rather than the lists themselves. In Python, objects, including lists, are stored in memory, and variables hold references to the memory locations where the objects are stored.

When you assign a list to a variable, the variable stores a reference to the memory location where the list object is stored. This means that the variable doesn't directly contain the list's elements but rather a reference to the list object.

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

1. copy.copy() (Shallow Copy):

The copy() function performs a shallow copy of an object. When applied to a list, it creates a new list object but only copies the references to the elements of the original list. In other words, it creates a new list with references to the same underlying objects as the original list. Changes to the original list's elements will be reflected in the copied list, as they refer to the same objects.

1. copy.deepcopy() (Deep Copy):

The deepcopy() function, on the other hand, performs a deep copy of an object. When applied to a list, it creates a new list object and recursively copies all the nested objects within it, creating separate copies. In this way, it ensures that the copied list is fully independent of the original list, and changes made to one do not affect the other.