**1. What exactly is []?**

[] represents an **empty list** in Python. It is a container that can hold a collection of items, but in this case, it is currently empty, with no values inside 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.)**

In Python, list indices start at 0. So, the third value corresponds to index 2. To assign 'hello' to the third value, you would do:

python

Copy

spam[2] = 'hello'

After this operation, spam would be ['2', '4', 'hello', 8, 10].

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

Let's break this down:

* '3' \* 2 gives '33' (string multiplication).
* int('33') converts it to the integer 33.
* 33 / 11 equals 3.0 (float division).
* int(3.0) converts it to 3 (integer).
* So, spam[int(int('3' \* 2) / 11)] is spam[3].

Since the list spam is ['a', 'b', 'c', 'd'], spam[3] is 'd'.

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

The index -1 refers to the **last element** in the list. So, for spam = ['a', 'b', 'c', 'd'], the value of spam[-1] is 'd'.

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

The slice spam[:2] refers to the first two elements in the list (index 0 and 1). For spam = ['a', 'b', 'c', 'd'], the value of spam[:2] is ['a', 'b'].

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

The index() method returns the index of the **first occurrence** of the specified value. For the list bacon = [3.14, 'cat', 11, 'cat', True], bacon.index('cat') will return 1 because 'cat' first appears at index 1.

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

The append() method adds the specified value to the **end** of the list. After executing bacon.append(99), the list will be:

python

Copy

bacon = [3.14, 'cat', 11, 'cat', True, 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 value from the list. After executing bacon.remove('cat'), the list will be:

python

Copy

bacon = [3.14, 11, 'cat', True]

The first 'cat' is removed.

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

* **List concatenation operator**: +  
  Concatenates two or more lists together. Example: [1, 2] + [3, 4] results in [1, 2, 3, 4].
* **List replication operator**: \*  
  Replicates a list a specified number of times. Example: [1, 2] \* 3 results in [1, 2, 1, 2, 1, 2].

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

* **append()**: Adds an element to the **end** of the list. Example: my\_list.append(5) adds 5 at the end.
* **insert()**: Adds an element at a **specific index** in the list. Example: my\_list.insert(1, 5) adds 5 at index 1.

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

1. **remove()**: Removes the **first occurrence** of a value from the list. Example: my\_list.remove(5) removes the first 5 from the list.
2. **pop()**: Removes an item at a **specific index** (or the last item if no index is provided). Example: my\_list.pop(1) removes the item at index 1.

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

* Both **lists** and **strings** are **ordered collections** of elements (values or characters).
* Both have **indexes**: You can access elements using indices (positive or negative).
* Both support **slicing**: You can slice them to create sublists or substrings.
* Both have similar **methods** (e.g., len(), in, etc.), but lists can contain different types of values, while strings are sequences of characters.

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

* **Lists** are **mutable**, meaning you can change their contents (add, remove, or modify items).
* **Tuples** are **immutable**, meaning once created, their contents cannot be changed.

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

A tuple with only one element requires a **trailing comma**:

python

Copy

(42,)

Without the comma, Python would interpret it as just an integer in parentheses.

**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 as a tuple**: You can use the tuple() function. Example:

python

Copy

my\_list = [1, 2, 3]

my\_tuple = tuple(my\_list) # (1, 2, 3)

* **To get a tuple as a list**: You can use the list() function. Example:

python

Copy

my\_tuple = (1, 2, 3)

my\_list = list(my\_tuple) # [1, 2, 3]

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

Variables that "contain" list values actually contain a **reference** to the list, not the list itself. This means they point to the location in memory where the list is stored.

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

* **copy.copy()**: Performs a **shallow copy**. It creates a new list, but the elements themselves are references to the original objects. Changes to mutable objects inside the copied list will affect the original list.
* **copy.deepcopy()**: Performs a **deep copy**. It creates a new list and also recursively copies all the objects inside it, meaning changes to the copied list's elements won't affect the original list.

Example:

python

Copy

import copy

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

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

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

shallow\_copy[0][0] = 99

print(original\_list) # Original list will be affected

print(shallow\_copy) # Modified list

deep\_copy[0][0] = 99

print(original\_list) # Original list will not be affected

print(deep\_copy) # Modified deep copy