1. What exactly is []?

[] is an empty list literal. It is used to create a list with no elements. For example, the following code creates an empty list:

Code snippet

my\_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.)

Let's pretend the spam includes the list ['a', 'b', 'c', 'd'] for the next three queries.

Ans:

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

# Assign the value 'hello' as the third value in the list

spam[2] = 'hello'

print(spam)

spam = ['a', 'b', 'c', 'd']

# Calculate the index of the third value in the list

index = int(int('3' \* 2) // 11)

# Print the value at the specified index

print(spam[index])

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

spam = ['a', 'b', 'c', 'd']

int('3' \* 2) / 11 = 3

spam[int(int('3' \* 2) / 11)] = spam[3] = 'd'

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

spam = ['a', 'b', 'c', 'd']

Then spam[-1] will be equal to 'd'.

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

spam = ['a', 'b', 'c', 'd']

Then spam[:2] will be equal to ['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')?

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

print(bacon.index('cat')) => 1

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

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

bacon.append(99)

print(bacon)

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

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

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

bacon.remove('cat')

print(bacon)

O/P : [3.14, 11, True]

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

List Concatenation (+):

list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined\_list = list1 + list2

print(combined\_list) # Output: [1, 2, 3, 4, 5, 6]

List Replication (\*):

original\_list = [1, 2, 3]

replicated\_list = original\_list \* 3

print(replicated\_list) # Output: [1, 2, 3, 1, 2, 3, 1, 2, 3]

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

The append() and insert() methods are used to add items to a list in Python. The main difference between the two methods is that append() adds an item to the end of the list, while insert() inserts an item at a specified index in the list.

Ex:

list = ["apple", "banana"]

list.append("cherry")

print(list)

list = ["apple", "banana"]

list.insert(1, "orange")

print(list)

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

In Python, there are two common methods for removing items from a list:

The remove() method:

my\_list = [1, 2, 3, 4, 5]

my\_list.remove(3)

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

The pop() method:

my\_list = [1, 2, 3, 4, 5]

removed\_item = my\_list.pop(2)

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

print(removed\_item) # Output: 3

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

List values and string values are identical in the following ways:

They are both sequences.

They both have a length.

They can both be compared for equality.

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

Tuples are immutable, while lists are mutable

Tuples are typically used to store data that does not need to be changed, such as the coordinates of a point or the names of people in a group

Tuples are slightly faster than lists, because they do not need to track changes to their contents

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

tup = (42,)

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

list\_value = [1, 2, 3]

tuple\_value = tuple(list\_value)

tuple\_value = (1, 2, 3)

list\_value = list(tuple\_value)

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

Variables that "contain" list values do not actually contain lists directly. Instead, they contain references to lists. A reference is a pointer to the actual list object in memory.

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

copy.copy() creates a shallow copy of an object. This means that the copy will contain the same values as the original object, but it will not contain any new references to other objects.

copy.deepcopy() creates a deep copy of an object. This means that the copy will contain the same values as the original object, and it will also contain new references to any other objects that are referenced by the original object.