1. What exactly is []?

* Empty 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.)

🡪 spam = [2, 4, 6, 8, 10]

spam[2] = 'hello'

print(spam)

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)]?

🡪 spam[int(int('3' \* 2) / 11)] = spam[int(33/11)] = spam[3] = 'd'

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

🡪 spam[-1] = 'd'

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

🡪 spam[:2] = ['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')?

🡪 1

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

🡪 bacon = [3.14, 'cat', 11, 'cat', True, 99]

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

🡪 bacon = [3.14, 11, 'cat', True, 99] : only the first occurrence of the string 'cat' was removed.

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

* The list concatenation operator is **+**, which concatenates two or more lists to create a new list. For example:

🡪 list1 = [1, 2, 3]

list2 = [4, 5, 6]

concatenated\_list = list1 + list2

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

* The list replication operator is **\***, which replicates a list a given number of times to create a new list. For example:

🡪 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()?

* **append()** adds an element to the end of the list.
* **insert()** adds an element to the list at a specific position

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

* **remove()**
* **pop()**

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

* Both can be indexed and sliced.
* Both are ordered sequences of elements.
* Both support the use of the **len()** function to determine their length.
* Both can be concatenated using the **+** operator.
* Both can be replicated using the **\*** operator.
* Both can be used in **for** loops to iterate over their elements.

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

* Lists are mutable but tuples are immutable.
* Syntax of list is [] but for tuples is ()
* Lists are used for storing items that can be changed but tuples are used to store items that cannot be changed.

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

* Tuple1 = (42,)

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, we can use the **tuple()** function and pass the list as an argument. For example:

🡪 my\_list = [1, 2, 3, 4]

my\_tuple = tuple(my\_list)

print(my\_tuple) # (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 are actually references to list objects in memory.
* When we assign a list to a variable, the variable does not actually "contain" the list. Instead, it contains a reference to the list object in memory.

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

* **copy.copy()** creates a shallow copy of the original list, which means that a new list is created with the same elements as the original list. Any changes to original list would reflect the change to the new copied list.
* **copy.deepcopy()** creates a deep copy of the original list, which means that a new list is created with the same elements as the original list, but the new list contains copies of the original elements. Any changes to original list would not reflect the change to the new copied list.