1. What exactly is []?

**Ans.** The index [] is used to create an empty list.

1. 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.)

**Ans.** spam = [2, 4, 6, 8, 10]

spam[2]='hello'.

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

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

**Ans.** int(int(‘3’ \* 2) / 11) = 3

So, spam[3] = ‘d’ .

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

**Ans.** spam[-1] = ‘d’ .

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

**Ans.** Gives ['a', 'b']

Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions.

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

**Ans.** The value of bacon.index(‘cat’)

The .index() function displays the index number of the element mentioned.

Although the element ‘cat’ appears at index 1 and 3, so it displays the first index value.

So the answer here is 1.

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

**Ans.** By putting bacon.append(99) we get

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

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

**Ans.** bacon.remove(‘cat’) removes the first time ‘cat’ appears in the list

The final result is [3.14, 'cat', 11, 'cat', True, 99]

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

**Ans.** The operators for list concatenation is + and for list replication is \*.

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

**Ans**. With append, you just add a new entry at the end of the list. With insert() you can create a new entry exactly in the position you want.

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

**Ans.** The two methods of removing items from a list are remove() that removes the very first element that matches

The pop() method removes an element from the **list** based on the index given.

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

**Ans.** A string is made of only characters whereas lists can have both characters and integers. So list values and string values are identical when they both contain the same set of characters.

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

**Ans.**

|  |  |
| --- | --- |
| Lists | Tuples |
| It is mutable | It is immutable |
| The implications of iterations are time consuming in lists | The implications of iterations are much faster |
| Operations like insertion and deletion are better performed. | Elements can be accessed better. |
| Consumes more memory | Consumes less memory |
| Many built-in methods available | Doesn’t have many built-in methods |
| Unexpected errors and changes can easily occur in lists | Unexpected errors and changes rarely occur in tuples |
| List syntax is []  e.g. - [14, 28, 35] | Tuple syntax ()  e.g. – (14, 28, 35) |

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

**Ans.** tuple = (42)

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

**Ans.** Getting list value from tuple form:

tuple = ('Numpy', 'tensorflow', 'keras', 'seaborn')

print(list(tuple))

Getting list value from tuple form:

def convert\_tuple(list):

return tuple(i for i in list)

#Converting list values to tuple

list = [1, 2, 3, 4]

print(convert\_tuple(list))

RESULT: (1, 2, 3, 4)

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

**Ans.** The variables will contain reference to list values rather than the values itself.

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

**Ans.** The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

* A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.

So if you change an element in the original object the copied object too changes.

* A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

If you change an element in the original object the elements in the copied object changes too.