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

**ANSWER:**

Index brackets [] have many uses in Python. First, they are used to define “LIST” literals, allowing you to declare a list and its contents in your program.

Index brackets are also used to write expressions that evaluate to a single item within a list or a single character in a string.

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.)**

**ANSWER:**

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

spam[2] = 'hello'

print(spam)

**Q. 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)]?**

**ANSWER:**

Let's start from the middle of this expression. '3' \* 2 evaluates to '33'. This is because multiplication of a str object by a int object results in concatenating string with itself n times (providing you are multiplying by n).

int('33') is just 33 as an integer. When you feed int class a string it will try to convert it to a number, and will raise ValueError if it cannot convert (like int('xfew')). int has an optional second argument base, that specifies the numeric base in with string is written. For instance int('ff', 16) returns 255. base is by default equal to 10.

Operator // is an operator of integer division. Therefore 33 // 11 is equal to 3. Note that 33 / 11 is equal to 3.0, a float. Of course after converting it to int this will also be just 3.

int(3) does really nothing. Just returns 3. Concluding spam[int(int('3' \* 2) // 11)] evals to spam[3]:

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

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

# returns 'd'

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

**ANSWER:**

The value of spam[-1] is

d

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

**ANSWER:**

The value of spam[:2] is

['a', 'b']

**Q. 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')?**

**ANSWER:**

The value of bacon.index('cat') is 1.

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

**ANSWER:**

bacon.append(99) will add a new item in the list

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

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

**ANSWER:**

bacon.remove('cat') will remove the value of cat from start so index 2 will be deleted.

[3.14, 11, 'cat', True]

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

**Answer:**

The concatenation operator connects one string at the end of the other. The operator for the List concatenation is +

The multiplication operator acts as a replication when we have one string and one integer as operands. The operator for replication is \*.

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

**ANSWER:**

The difference between the two methods is that

Append() adds an item to the end of list whereas

Insert() inserts an item in a specified position in the list.

For **Example-**

list2.insert(0,"divya") ######### will insert divya as 0th /1st element in list 2 (need to passs index)

list2.append("mauu") ########### append at last position

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

**ANSWER:**

The methods are remove(), pop() and clear().

The remove() helps to remove the very first given element matching from the list.

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

The clear() method will remove all the elements present in the list.

**For EXAMPLE-**

list1.remove("mango") ############## will remove particular name from list

list1.pop(1) ############## will delete 1st index element

list1.clear() ############# will clear the elements from list1

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

**Answer:**

 The similarity between Lists and Strings in python is that both are sequence.

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

**ANSWER:**

1. **List**

Lists are mutable.

The Implication of Iteration is time consuming in list.

Operations are like insertion and deletion are better performed.

Consumes more memory.

Many built in methods are available.

Unexpected errors and changes can easily occur in lists.

1. **Tuple**

Tuples are immutable.

The implication of iterations are much faster in tuples.

Elements can be accessed better.

Consumes less memory.

Does not have any built-in methods.

Unexpected errors and changes rarely occur in tuples.

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

**ANSWER:**

Tuples can be written using parentheses, ( \_\_\_,) where the trailing comma is mandatory.

As for integer 42, it will be written as (42,).

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

**ANSWER:**

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.

For **EXAMPLE-**

list\_names=['Meredith', 'Kristen', 'Wright', 'Franklin']

tuple\_names= tuple(list\_names) ###### created new tuple as list\_names

print(tuple\_names)

print(type(tuple\_names)) ######## print class of the variable tuple\_names

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

**ANSWER:**

A list is used to store multiple values/items in single variable. OR collection of variables.

Variables will contain references to list values rather than list values themselves.

But for strings and integer values, variables simply contain the string or integer value.

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

**ANSWER:**

Copy.copy() [copy(shallow copy)] does not create a copy of nested objects, instead it just copies the references to the nested objects,

while deepcopy(deep copy) copies all the nested objects recursively.

A a,b,c,d,a1,b1,c1,d1 are references to objects in memory, which are uniquely identified by their id’s.

For **EXAMPLE-**

copy.copy(x)

Return a shallow copy of x.

copy.deepcopy(x[, memo])

Return a deep copy of x.