Q 1. What exactly is []?

Answer –

Empty list - [] represents an empty list. Following we have a list named test which is empty and can be appended anywhere in the program.

```
test = []
```

Accessing list element - [] is also used to access a particular element of the list. Here we have a list named test. The statement test[1] we access the element at index 1, “b” in this case

```
test = ["a","b","c"]
test[1]
```

Q 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.)

Answer –

```
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.

Q 3. What is the value of spam[int(int('39' * 2) / 11)]?

Answer – The statement will output the 358th element of the spam list. But, since the spam has only 5 elements, it will result in an index out of range exception.

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

Answer – Output = 'd'

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

Answer – Output = ['a', 'b']

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

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

Answer – Output = 1

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

Answer – Output = [3.14, 'cat', 11, 'cat', True, 99]

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

Answer – Output - [3.14, 11, 'cat', True, 99]

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

Answer –

Concatenation Operator : The + operator is used for concatenating two or more lists together, resulting in a new list that contains all the elements from the two lists.

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
concatenated = list1 + list2
print(concatenated)
```

Output: [1, 2, 3, 4, 5, 6]

Replication Operator : The * operator is used for replicating a list by a specified number of times, creating a new list with repeated elements.

```
list1 = [1, 2, 3]
replicated = list1 * 3
print(replicated)
```

Output: [1, 2, 3, 1, 2, 3, 1, 2, 3]

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

Answer –

Append - The append method is used to add an element to the end of a list.

We have a list *testlist = [1,2,3]*. *testlist.append(4)* will add value 4 to the end of the list.

testlist=[1,2,3,4]

Insert - The insert method is used to insert a value inside the list at a particular index. We have a list *testlist = [1,2,3]*. The line *testlist.insert(1,4)* will insert the value 4 at index 1 of the list. The *testlist will be [1,4,2,3]*

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

Answer –

Remove - The remove method removes the first occurrence of the element from the list.

```
testlist = [1, 2, 3, 2, 4]
testlist.remove(2)
print(testlist)

Output: [1, 3, 2, 4]
```

Del - The del keyword is used to remove an element at a specific index of the list.

```
testlist = [1, 2, 3, 2, 4]
del testlist[1]
print(testlist)

Output: [1, 3, 2, 4]
```

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

Answer –

Loops usage - We can iterate the values of list as well as string using a for loop.

Length of variable - We can use len() method to find out the length of the string and the list.

Accessing the elements - Lists and strings can be accessed using index

Concatenation - Lists and strings both can be joined using **'+'** operator.

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

Answer –

1. Tuples are immutable. This means that once we define a tuple it can't be changed later in the program. On the other hand, the list can be modified, which means we can add, modify, remove list elements.
2. The tuples are defined using parentheses **"()".** Lists are defined using square brackets **"[]"**

   ```
   testtuple = (1,2,3,4)
   testlist = [1,2,3,4]
   ```

3. Tuples are faster to process and more memory efficient as they are non changeable. Lists require more memory due to their ability to change.
4. List provides a wide range of methods to manipulate the list. Tuples on the other hand provide fewer methods as it is immutable.

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

Answer –

```
Test_tuple =(42)
```

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

Answer –

To convert a list to tuple we can use the tuple() function. Following we convert list named *test_list* to a tuple named *tuple_test*

```
test_list = ["a","b","c","d"]
tuple_test = tuple(test_list)
```

To convert a tuple to a list, we can use the list() function. Following we convert list named *test_tuple* to a list named *list_test*

```
test_tuple = ("a","b","c","d")
list_test = list(test_tuple)
```

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

Answer –

Variables that "contain" list values actually contain references to the list objects rather than the lists themselves. Objects are stored in memory, and variables serve as references or pointers to the memory locations where the objects reside. This means that the variable "contains" the memory address of the list rather than the list data directly.

Q 16. How do you distinguish between copy.copy() and copy.deepcopy()?

Answer –

For this we will take an example of a list named *test_list* which has a nested list. *test_list = [1, 2, [3, 4]]* and we make its copy and name it as *test_list_copy*

Copy.copy() - This function performs a shallow copy of the list. The list *test_list* contains a nested list, so the reference to this nested list will be copied to *test_list_copy* instead of the list. This means that any changes made to the nested list will be reflected in both *test_list and test_list_copy.*

copy.deepcopy() - This function creates a new object and recursively duplicates the entire object hierarchy. Any changes made to the nested list of *test_list_copy* will not reflect in *test_list.*