1. What exactly is []?

The SQUARE BRACKETS [] represent list comprehension in Python. The square brackets are used for specifying Python to store the values supplied inside the brackets as a LIST. We can also use [] to create an empty list with no values inside it.

```
#Sample list creation using [] consisting of different elements with diff datatypes
list_= [1,2, 3 ,'Hello', ['H','I',11]]

#Emply List
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.)

Lists have an in-built method called insert() which can be used to insert a value at the desired index list.insert() takes the following inputs:

- INDEX Position at which the insertion should occur
- Value to be inserted

For the example above:

```
spam = [2, 4, 6, 8, 10]
#Lists are 0 indexed in Python, so the list 'spam' has 5 elements with indices from 0 to 4
#Currently third value in the list is 6 @ INDEX 2

#Insertion of 'hello' at the third position
spam.insert(3,'hello')
```

Note that, on performing the insert operation, the elements to the right of inserted index gets pushed by 1 index to the right. Overall length of the list gets increased by 1.

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

The question is ambiguous. It does not clearly say what the elements in the list 'spam' are. I am considering the original list [2, 4, 6, 8, 10] is appended with ['a', 'b', 'c','d']. The 'hello' insertion has been ignored. New list is [2, 4, 6, 8, 10, ['a', 'b', 'c','d']]

The questions 3, 4 and 5 are answered considering: spam = [2, 4, 6, 8, 10, ['a', 'b', 'c','d']]

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

The value is 8

Explanation: ('3'*2) results in the '33', int() converts '33' to numerical 33. With the division operator, 33/11 returns 3.0, outermost int() used to ensure the list indexing is passed an integer 3 rather than a float. Hence, output is spam[3] which is **8**.

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

spam[-1] is ‘d’. Numerical value of -1 in the index refers to the last index of the list.

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

The value is a sub-list [2,4]

Slicing of the list spam [num1:num2] returns a sub-list of the list ‘spam’ starting from index num1 to index (num1 -1). If num1 is empty, the sub-list starts from the first item of the list. If num2 is empty, the sub-list ends at the last item of the list.

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')?

#list.index(x) returns the index of the first occurrence of x in the list.
bacon.index('cat') = 1

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

#list.append(x) adds an element to the end of the list with the value x
#list.append(x) appends (x) to list at the tail

bacon.append(99)
#Updated ‘bacon’ list is now [3.14, 'cat,' 11, 'cat,' True, 99]

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

#list.remove(x) removes the first occurrence of x in the list.
bacon.remove('cat')
#Updated ‘bacon’ list is now [3.14, 11, ‘cat’, True, 99]

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

- **List Concatenation refers to adding a new list to the tail of an existing list and creating a new list.**
  - Arithmetic sum operator , + is the ’ List Concatenation Operator’
  - Example :
    list_1 = [1, 4]
    list_2=[‘Hello, ‘@’]
    list_conc=list_1 + list_2
    #list_conc has the new concatenated list : [1, 4, ‘Hello’, ‘@’]
- **List replication refers to create a new list which repeats an existing list**
  - Arithmetic multiplication operator , * is the ’ List Replication Operator’
  - Example :
    list_conc = [1, 4, ‘Hello’, ‘@’]
    list_replicated=list_conc*2
    #list_replicated has the new concatenated list : [1, 4, ‘Hello’, ‘@’, 1, 4, ‘Hello’, ‘@’]

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

list.append(x) creates a new element in the list by adding x to the tail (last position) of the list
list.insert(index,x) inserts the element at the index location specified.

For further details, please refer Question 6 and Question 7.

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

The two main methods of removing items from a list are:

- remove() : list.remove(x) removes the first occurrence of the x from the list
- del operator : del list(index) deletes the element @ the index specified
    - ‘del operator’ can be used to remove multiple items by proving an index range

In addition, we can also use pop(). It is similar to the ‘del operator’, but in addition to the deletion of the element at the specified index, it also returns the deleted the element. It is applicable only for popping out and returning one element at a time.

- list.pop(index)

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

Lists contain a list of elements of different data types, indexed from 0 to (length of the list – 1)
Strings contain a sequence of characters indexed from 0 to (length of the list – 1)
Thus the element calls using list[index] and a specific character from the string using str[index] are identical for strings and lists. All the sub-setting and slicing operations applicable for a list are applicable for strings also.

Note that, strings are IMMUTABLE; the contents of the existing string cannot be updated or modified. Any string method resulting in successful updation of the current string achieves it by regenerating the whole string from scratch.
On the other hand, LISTS are MUTABLE. Lists can be updated and modified on the fly without any regeneration.

The following sample codes provide additional information on string and list operations

String Operation:
https://github.com/arvindhhp/PyPro_ahhp/blob/main/Part_003_StringOperations.ipynb

List Operation:
https://github.com/arvindhhp/PyPro_ahhp/blob/main/Part_004_ListOperations.ipynb

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

Tuples are similar to lists, consisting of multiple elements with heterogeneous data types. But tuples are IMMUTABLE while lists are MUTABLE. The data within tuples cannot be modified.
Tuples are represented using parentheses () while lists are represented using square brackets []

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

Tuple definition for a single element tuple needs to be aided by a comma (,). This aids Python is interpreting the single integer value as a tuple and not as an integer expression within ()

```
tuple=(42, )
```

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

Tuples and lists can be inter-converted using the built-in methods list() and tuple()

- Type conversion method list(tuple) converts the tuple to a list
- Type conversion method tuple(list) converts the list to a tuple

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

Lists are a collection of items stored in an ordered and sequential manner. The list variable name refers to the list object created whenever called in the program. The list variable effectively contains the memory address of the list object.

In Python, everything is an object. Each and every string, number, list, dictionary, class, function, the boolean objects 'True' and 'False' including the 'None' object. Python will store each of them at a particular memory address. When a new list is created and it is assigned to some list variable, a new list object is created in the memory and the memory address of this list object is stored in the list variable name.

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

Deep copy is a process in which the copying process occurs recursively. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. In case of deep copy, a copy of object is copied in other object. It means that any changes made to a copy of object do not reflect in the original object. In python, this is implemented using "deepcopy()" function.

A shallow copy means constructing a new collection object and then populating it with references to the child objects found in the original. The copying process does not recurse and therefore won't create copies of the child objects themselves. In case of shallow copy, a reference of object is copied in other object. It means that any changes made to a copy of object do reflect in the original object. In python, this is implemented using "copy()" function.

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