1. What exactly is []?

Ans:   
In Python, [] is an empty list. A list is a data structure that can store a collection of objects. The objects in a list can be of any type, such as numbers, strings, or objects. The elements of a list are separated by commas and enclosed in square brackets.

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.)

Ans: here is the program

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)]?

Ans: output is : ‘d’

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

Ans: output is : ‘d’

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

Ans: output is : ['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')?

Ans: output is : 1

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

Ans: output is : [3.14, 'cat', 11, 'cat', True, 99]

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

Ans: output is : [3.14, 11, True, 99]

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

Ans: In Python, the list concatenation operator is +, and it is used to combine two or more lists into a single list. When the + operator is used between two lists, it creates a new list containing all the elements from the first list followed by all the elements from the second list. Here's an example:

list1 = [1, 2, 3]

list2 = [4, 5, 6]

concatenated\_list = list1 + list2

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

> On the other hand, the list replication operator is \*, and it is used to create a new list by repeating the elements of an existing list a certain number of times. When the \* operator is used between a list and an integer, it creates a new list with the elements of the original list repeated the specified number of times. Here's an example:

list1 = [1, 2, 3]

repeated\_list = list1 \* 3

print(repeated\_list) # Output: [1, 2, 3, 1, 2, 3, 1, 2, 3]

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

Ans: The main difference between the append() and insert() methods in Python is that append() adds an item to the end of a list, while insert() inserts an item at a specified position in the list.

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

ANS: There are two main methods for removing items from a list in Python:

* remove() method: This method takes a single element as an argument and removes it from the list. If the element doesn't exist, it throws a ValueError exception.
* pop() method: This method takes an index as an argument and removes the element at that index from the list. If the index is out of range, it throws a IndexError exception.

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

Ans: List values and string values are identical in the following ways:

* They are both sequences. This means that they can be indexed and iterated over.
* They can both be compared for equality. The comparison operator == can be used to compare two lists or two strings.
* They can both be converted to other data types. For example, a list can be converted to a string using the str() function, and a string can be converted to a list using the list() function.

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

ANS:   
Tuples and lists are both data structures in Python that can be used to store collections of data. However, there are some key differences between the two.

* Mutability: Lists are mutable, which means that the elements in a list can be changed after the list is created. Tuples are immutable, which means that the elements in a tuple cannot be changed after the tuple is created.
* Speed: Tuples are typically faster than lists because they are immutable. This is because the Python interpreter does not need to check to see if the elements in a tuple have been changed when it is executing code.

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

ANS: here is the example : tuple\_value = (42)

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

ANS: o get a list value's tuple form, you can use the built-in function tuple(). For example, the following code converts a list value to a tuple value:

list\_value = [1, 2, 3]

tuple\_value = tuple(list\_value)

> The tuple() function takes a list as an argument and returns a tuple with the same elements as the list.

To get a tuple value's list form, you can use the built-in function list(). For example, the following code converts a tuple value to a list value:

tuple\_value = (1, 2, 3)

list\_value = list(tuple\_value)

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

Ans:

Variables that "contain" list values do not actually contain lists themselves. Instead, they contain **references** to lists. A reference is a pointer to the location in memory where the list is stored. This is important to understand because it means that multiple variables can refer to the same list

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

Ans:

The copy module in Python provides two functions for copying objects: copy.copy() and copy.deepcopy().

* copy.copy() performs a shallow copy of an object. This means that it creates a new object that contains the same data as the original object, but the new object does not contain any copies of the original object's subobjects.
* copy.deepcopy() performs a deep copy of an object. This means that it creates a new object that contains the same data as the original object, and any subobjects of the original object are also copied recursively.