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

**Ans:** In Python,’ []’ is used to create an empty list. A list is a fundamental data structure in Python that can hold an ordered collection of items, which can be of any data type. An empty list is simply a list with no elements.

Here's how you can create an empty list using []:

python code

empty\_list = []

we can also create a list with elements by placing values within the 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:** To assign the value "hello" as the third value in a list stored in a variable called ‘spam’, we can use indexing and assignment:

Python code:

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

spam[2]="hello"

print(spam)

[2, 4, 'hello', 8, 10]

**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:** Python Code

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

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

'd'

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

**Ans:** Python Code

spam[-1]

'd'

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

**Ans:** Python code

spam[:2]

['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:**

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

**Ans:** Python code

bacon=[3.14,"cat",11,"cat",True]

bacon.index("cat")

1

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

**Ans:** Python code

bacon.remove("cat")

print(bacon)

[3.14, 11, 'cat', True]

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

**Ans**: In Python, the list concatenation operator is +, and the list replication operator is \*.

1. **List Concatenation Operator (+):** The + operator is used to concatenate (combine) two or more lists into a single list. It creates a new list containing all the elements from the original lists in the order they appear.

Example:

List1=[1,2,3,4]

List2=[5,6,7,8]

Concatinated\_List=List1+List2

print(Concatinated\_List)

[1, 2, 3, 4, 5, 6, 7, 8]

1. **List Replication Operator (\*):** The \* operator is used to replicate a list by a specified number of times. It creates a new list containing repeated copies of the original list.

Example:

List=[1,0]

Replicated\_List=List\*3

print(Replicated\_List)

[1, 0, 1, 0, 1, 0]

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

**Ans:** Both the append() and insert() methods are used to add elements to a list in Python, but they differ in how they add elements and where they add them within the list:

1. append() Method: The append() method is used to add an element to the end of a list. It takes a single argument, which is the element you want to add.

Example:

list=[1,2,4]

list.append(3)

print(list)

[1, 2, 4, 3]

1. insert() Method: The insert() method is used to add an element at a specific index within the list. It takes two arguments: the index at which you want to insert the element and the element itself.

Example:

list=[1,2,4]

list.insert(2,3)

print(list)

[1, 2, 3, 4]

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

**Ans:** There are two methods for removing items from a list in Python:

1. remove() Method: The remove() method is used to remove the first occurrence of a specified value from the list. If the value is not found, it raises a ValueError.

Python code

my\_list = [1, 2, 3, 2, 4]

my\_list.remove(2)

print(my\_list)

[1, 3, 2, 4]

1. pop() Method: The pop() method is used to remove an element from a specific index in the list. It takes an optional index argument and returns the removed element. If no index is provided, it removes and returns the last element.

Python code

my\_list = [1, 2, 3, 4]

removed\_element = my\_list.pop(1)

print( my\_list)

[1, 3, 4]

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

**Ans:** **List values and string values share some similarities in terms of their characteristics and operations:**

1. **Ordered Sequences:** Both lists and strings are ordered sequences of elements. In a list, the elements can be of any data type (including other lists), while in a string, the elements are characters.
2. **Indexing and Slicing:** Both lists and strings can be accessed using indexing and slicing. You can retrieve individual elements by their index and extract sublists or substrings using slicing.
3. **Iteration:** You can iterate over both lists and strings using loops. This allows you to process each element or character in sequence.
4. **Concatenation:** Both lists and strings support concatenation using the + operator. You can combine multiple lists or strings to create a new one.
5. **Replication:** Both lists and strings support replication using the \* operator. You can create a new list or string by repeating the original multiple times.
6. **Length:** You can determine the length of both lists and strings using the len() function. It returns the number of elements (for lists) or characters (for strings).

However, there are also important differences between lists and strings. Lists are mutable (modifiable), meaning you can change their elements after creation, while strings are immutable (unchangeable), meaning you cannot modify individual characters within a string. Additionally, lists can contain elements of different types, while strings consist of characters only.

**Top of Form**

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

**Ans:** 1. Mutability:

* Lists: Lists are mutable, meaning you can modify their elements after creation. You can add, remove, or change elements within a list.
* Tuples: Tuples are immutable, meaning once you create a tuple, you cannot modify its elements. Tuples are essentially read-only lists.

2. Syntax:

* Lists: Lists are defined using square brackets [].
* Tuples: Tuples are defined using parentheses ().

3. Use Cases:

* Lists: Lists are commonly used when you need a collection that can be modified dynamically, such as storing a list of tasks, user inputs, or data that may change over time.
* Tuples: Tuples are often used when you want to ensure that the data remains unchanged. They are useful for situations where you want to group related data together, like representing coordinates or returning multiple values from a function.

4. Performance:

* Lists: Due to their mutability, lists may consume more memory and have slightly slower performance compared to tuples.
* Tuples: Tuples are more memory-efficient and generally have faster performance due to their immutability.

5. Syntax for Creating Single Items:

* Lists: To create a list with a single item, use square brackets: [item].
* Tuples: To create a tuple with a single item, use parentheses: (item,).

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

**Ans:** To create a tuple value that only contains the integer 42, you can use parentheses () with a single element inside.

Note the comma ’ , ’ after the integer value. It's important to include the comma even if the tuple contains a single element. This is because the parentheses are also used for grouping expressions, so the comma distinguishes a single-element tuple from a simple expression enclosed in parentheses.

Python code

tuple=(20)

print(tuple)

20

type(tuple)

<class 'int'>

tuple=(20,)

print(tuple)

(20,)

type(tuple)

<class 'tuple'>

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

**Ans:** To convert a list value to its tuple form, we can use the tuple() constructor. To convert a tuple value to its list form, you can use the list() constructor.

Example

mytuple=(1,2,3)

mylist=[4,5,6]

tuple\_to\_list=list(mytuple)

print(tuple\_to\_list)

[1, 2, 3]

list\_to\_tuple=tuple(mylist)

print(list\_to\_tuple)

(4, 5, 6)

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

**contain?**

**Ans:** Variables that "contain" list values in Python do not actually contain the list itself, but rather they contain a reference to the list. This is true for all objects in Python, not just lists.

When we assign a list to a variable, the variable doesn't store the entire list's data directly. Instead, it stores a reference (or memory address) that points to the location in memory where the list data is stored. This reference allows you to access and manipulate the list's contents through the variable.

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

**Ans:** In the copy module of Python, both copy() and deepcopy() functions are used to create copies of objects. However, they behave differently when it comes to nested objects and references. Here's how they differ:

1. **copy() Function**: The copy() function creates a shallow copy of an object. In a shallow copy, a new object is created, but the contents of the object (such as elements within a list) are still references to the same objects as the original. If the original object contains nested objects, the shallow copy only copies references to the nested objects, not the nested objects themselves.
2. **deepcopy() Function:** The deepcopy() function creates a deep copy of an object. In a deep copy, both the object and all the objects nested within it are copied recursively, creating completely new objects. This ensures that the copy is independent of the original object, and changes made to the copy do not affect the original or vice versa.

Example:

original\_list=[1,[2,3]]

shallowCopy=copy.copy(original\_list)

deepCopy=copy.deepcopy(original\_list)

print(original\_list)

[1, [2, 3]]

print(shallowCopy)

[1, [2, 3]]

print(deepCopy)

[1, [2, 3]]

shallowCopy[1][0]=9

deepCopy[1][0]=999

print(original\_list)

[1, [9, 3]]

print(shallowCopy)

[1, [9, 3]]

print(deepCopy)

[1, [999, 3]]