**Name: Abhas**

**Email:** [**abhas2495@gmail.com**](mailto:abhas2495@gmail.com)

1. What exactly is []?

**Answers:**

In Python, [] is used to denote an empty list. A list is a collection of items enclosed in square brackets [], where each item is separated by a comma.

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.)

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

**Answers:**

spam = [2, 4, 6, 8, 10] # Initial list

spam[2] = 'hello' # Assign 'hello' to the third value

print(spam) # Output: [2, 4, 'hello', 8, 10]

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

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

spam[2] = 'hello'

print(spam)

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

**Answers:**

-> The expression '3' \* 2 evaluates to the string '33'.

-> The expression int('33') converts the string '33' to an integer with a value of 33.

-> The expression int('33') / 11 performs integer division, resulting in 3.

Finally, spam[3] accesses the element at index 3 in the list spam

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

Answers:

The value of spam[-1] depends on the content of the list spam.

In general, indexing with -1 in Python refers to the last element of a list. It provides a way to access elements from the end of the list without knowing the exact length of the list.

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

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

**Answers:**  
If the list bacon is [3.14, 'cat', 11, 'cat', True], then the value of bacon[:2] can be determined as follows:

The slicing bacon[:2] retrieves elements from the beginning of the list up to index 2 (exclusive). Therefore, it returns a new list containing the elements 3.14 and 'cat'.

So, in this case, the value of bacon[:2] is [3.14, 'cat']

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

**Answers:**

If the list bacon is [3.14, 'cat', 11, 'cat', True], the value of bacon.index('cat') can be determined as follows:

The index() method is used to find the first occurrence of a specified element in a list. It returns the index of the element if found.

In this case, 'cat' appears at index 1 and also at index 3 in the bacon list. However, the index() method returns the index of the first occurrence, which is 1.

So, the value of **bacon.index('cat') is 1.**

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

**Answers:**

If the list bacon is [3.14, 'cat', 11, 'cat', True], the bacon.append(99) statement will add the value 99 to the end of the list bacon. Here's how the list will look after executing bacon.append(99):

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

The append() method in Python is used to add an element to the end of a list. In this case, it adds the value 99 as the last element in the list bacon. The list is modified in place, and the new value is appended at the end.

So, after the bacon.append(99) statement, the updated list bacon will have the following values: [3.14, 'cat', 11, 'cat', True, 99].

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

**Answers:**

If the list bacon is [3.14, 'cat', 11, 'cat', True], the bacon.remove('cat') statement will remove the first occurrence of the value 'cat' from the list bacon. Here's how the list will look after executing bacon.remove('cat'):

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

The remove() method in Python is used to remove the first occurrence of a specified element from a list. In this case, it removes the first occurrence of the value 'cat' from the list bacon.

After the bacon.remove('cat') statement, the updated list bacon will have the following values: [3.14, 11, 'cat', True]. The first 'cat' is removed, while the second occurrence remains in the list.

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

**Answers:**

The list concatenation operator in Python is the + symbol, and the list replication operator is the \* symbol. Here's a brief explanation of each:

List Concatenation (+): The + operator is used to concatenate or combine two lists into a single list. When you use the + operator between two lists, it creates a new list that contains all the elements from both lists in the order they appear.

Example:

list1 = [1, 2, 3]

list2 = [4, 5, 6]

concatenated\_list = list1 + list2

print(concatenated\_list)

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

List Replication (\*): The \* operator, when used with a list and an integer, replicates the elements of the list a specified number of times to create a new list. The original list is not modified.

Example:

list1 = [1, 2, 3]

replicated\_list = list1 \* 3

print(replicated\_list)

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

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

**append():** The append() method is used to add an element to the end of a list. It takes a single argument, which is the element to be added, and modifies the list in-place.

**insert():** The insert() method is used to add an element at a specific index position within a list. It takes two arguments: the index position where the element should be inserted and the element itself.

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

**Answers:**

**remove():** The remove() method is used to remove the first occurrence of a specific value from a list. It takes a single argument, which is the value to be removed. If the value is found in the list, it is removed; otherwise, it raises a ValueError.

**pop():** The pop() method is used to remove an element from a specific index position in a list. It takes an optional argument, which is the index of the element to be removed. If no index is specified, it removes and returns the last element in the list.

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

**Answers:**

**Similarity**

**Sequential Access:** Both lists and strings allow for sequential access to their elements. Each element in a list or a string can be accessed individually by its index or through iteration using loops.

**Indexing and Slicing:** Both lists and strings support indexing and slicing operations. You can access individual elements of a list or a string by their index, and you can extract a portion of a list or a string using slicing.

**Length:** Both lists and strings have a length, which can be obtained using the len() function. The length represents the number of elements or characters in the list or string.

**Differences**

**However, there are also significant differences between lists and strings:**

**Mutability:** Lists are mutable, which means their elements can be modified. Elements can be added, removed, or modified within a list. In contrast, strings are immutable, meaning their individual characters cannot be changed. Instead, you need to create a new string with the desired modifications.

**Type of Elements:** Lists can store elements of different data types, including numbers, strings, booleans, and even other lists. Strings, on the other hand, can only store a sequence of characters.

**Methods and Operations:** Lists and strings have different methods and operations specific to their respective data types. For example, lists have methods like append(), insert(), and remove() for modifying the list, while strings have methods like split(), join(), and replace() for string manipulation.

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

**Mutability:** Lists are mutable, meaning their elements can be modified, added, or removed after the list is created. Tuples, on the other hand, are immutable, meaning once a tuple is created, its elements cannot be changed.

**Syntax:** Lists are defined using square brackets [], while tuples are defined using parentheses ()

**Modification:** Since tuples are immutable, you cannot directly modify their elements. If you want to change a tuple, you need to create a new tuple with the desired modifications. Lists, being mutable, allow direct modification of their elements.

**Usage:** Lists are commonly used for storing and manipulating collections of data where the order and changeability of elements are important. Tuples, on the other hand, are often used to represent a collection of related values where immutability is desired, such as coordinates, database records, or function arguments and return values.

**Performance:** Tuples are generally more memory-efficient and faster to access than lists. Since tuples are immutable, they require less memory and can be optimized by the Python interpreter. Lists, being mutable, require additional memory allocation and handling.

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

**Answers:**

To create a tuple with a single value of 42, you can use the following

syntax: my\_tuple = (42,)

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

**Answers:**

To convert a list to a tuple, you can use the tuple() function. It takes an iterable as an argument, such as a list, and returns a tuple containing the elements of the iterable.

**Example:**

**my\_list = [1, 2, 3, 4, 5]**

**my\_tuple = tuple(my\_list)**

**print(my\_tuple) # (1, 2, 3, 4, 5)**

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

**Answers:**

Variables that "contain" list values in Python do not actually store the list itself. Instead, they store a reference to the list. In other words, the variable contains the memory address where the list is stored in the computer's memory.

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

The **copy.copy()** and **copy.deepcopy()** functions are both part of the copy module in Python, but they differ in how they create copies of objects:

1. **copy.copy()** performs a shallow copy of an object. It creates a new object and copies the references of the original object's elements to the new object.
2. **copy.deepcopy()** performs a deep copy of an object. It creates a completely independent copy of the object and recursively copies all the elements within the object, including nested objects. This means that changes made to the elements of the copied object will not affect the original object or other copied objects.