1. What exactly is []?

Ans1

[] is used to represent an empty list, which is a collection of values that can be of any type. Lists are one of the built-in data structures in Python and are commonly used to store and manipulate sequences of data.

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.

ans2

To assign the value 'hello' as the third value in the list stored in the variable spam, you can use the indexing notation and assign the value directly to the desired index:

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

spam[2] = 'hello'

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

To assign 'hello' as the third value in the list [‘a’, ‘b’, ’c’, ‘d’], you can use the same approach:

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

spam[2] = 'hello'

print(spam) # Output: ['a', 'b', 'hello', 'd']

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

Ans3

The expression '3' \* 2 multiplies the string '3' by 2, resulting in the string '33'. Then, int('33') converts the string to an integer with the value 33.

The expression int('33') / 11 divides 33 by 11, resulting in the float value 3.

Finally, spam[int(3)] indexes the list spam at the integer value 3, which is the fourth element in the list since list indexing starts at 0.

Therefore, the value of spam[int(int('3' \* 2) / 11)] depends on the contents of the list spam.

Eg

If spam is equal to [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], then the expression evaluates to spam[3], which is the integer value 3. However, if spam is a list of strings or other types, the expression could produce an error if the indexing operation is not supported for that type.

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

Ans4

The expression spam[-1] accesses the last element in the list spam. In Python, negative indices are used to index elements from the end of the list, with -1 representing the last element, -2 representing the second last element, and so on.

Therefore, the value of spam[-1] depends on the contents of the list spam. For example, if spam is equal to [2, 4, 6, 8, 10], then spam[-1] evaluates to the integer value 10, which is the last element in the list.

If spam is an empty list, trying to access spam[-1] would result in an IndexError, since there are no elements in the list to index.

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.

Ans5

The expression spam[-1] accesses the last element in the list spam. In Python, negative indices are used to index elements from the end of the list, with -1 representing the last element, -2 representing the second last element, and so on.

if bacon is equal to [3.14, 'cat', 11, 'cat', True], then:

bacon[0] is 3.14, which is a float.

bacon[1] is 'cat', which is a string.

bacon[2] is 11, which is an integer.

bacon[3] is 'cat', which is a string.

bacon[4] is True, which is a boolean value.

Therefore, the list bacon contains elements of different types, including floats, strings, integers, and boolean values.

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

Ans6

If bacon is equal to [3.14, 'cat', 11, 'cat', True], then the expression bacon.index('cat') returns the index of the first occurrence of the string 'cat' in the list bacon.

In this case, the string 'cat' appears at indices 1 and 3 in the list bacon. However, the index method only returns the index of the first occurrence, so bacon.index('cat') evaluates to 1.

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

Ans7

If bacon is equal to [3.14, 'cat', 11, 'cat', True], the method append can be used to add an element to the end of the list. For example, the expression bacon.append(99) adds the integer value 99 to the end of the list, so bacon becomes [3.14, 'cat', 11, 'cat', True, 99].

Therefore, the append method modifies the original list by adding a new element to the end. In this case, the new element is the integer value 99.

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

Ans8

the original list bacon has been changed to [3.14, 11, 'cat', True].

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

Ans9

The list concatenation operator is the plus symbol +, which is used to join two or more lists into a single list. The list replication operator is the asterisk symbol \*, which is used to create a new list that contains multiple copies of the same list.

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

Ans10

append() and insert() are two methods used to add elements to a list, but they differ in how they add elements to the list.

The append() method adds an element to the end of the list, whereas the insert() method adds an element at a specified position in the list.

Eg

Using append()

my\_list = [1, 2, 3]

my\_list.append(4) # Adds the integer value 4 to the end of the list

print(my\_list) # Output: [1, 2, 3, 4]

my\_list.append('hello') # Adds the string value 'hello' to the end of the list

print(my\_list) # Output: [1, 2, 3, 4, 'hello']

Using insert()

my\_list = [1, 2, 3]

my\_list.insert(0, 0) # Adds the integer value 0 at position 0 in the list

print(my\_list) # Output: [0, 1, 2, 3]

my\_list.insert(2, 'hello') # Adds the string value 'hello' at position 2 in the list

print(my\_list) # Output: [0, 1, 'hello', 2, 3]

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

Ans11

The remove() method removes the first occurrence of a specified value from the list, while the pop() method removes an element at a specified index from the list and returns that element.

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

Ans12

Indexing: Both lists and strings support indexing, which means that you can access individual elements of the sequence by their position or index.

Slicing: Both lists and strings support slicing, which means that you can extract a portion of the sequence by specifying a start index, an end index, and a step size.

Iteration: Both lists and strings can be iterated over using a loop.

Concatenation: Both lists and strings support concatenation, which means that you can combine two or more sequences into a single sequence.

Repetition: Both lists and strings support repetition, which means that you can create a new sequence that repeats the original sequence a specified number of times.

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

Ans13

|  |  |
| --- | --- |
| tuples | lists |
| Tuples are immutable, which means that their contents cannot be changed after they are created. Once a tuple is created, its elements cannot be added, removed, or modified.  Tuples are defined using parentheses (())  Tuples are generally faster and more memory-efficient than lists, especially for large collections of data, because they are immutable and require less memory allocation.  Tuples are often used to store related items that cannot be changed, such as a pair of coordinates or a set of configuration settings. | Lists are mutable, which means that their contents can be changed after they are created.  Lists are defined using square brackets ([])  Lists are generally slower and less memory-efficient than tuples.  Lists are often used to store collections of related items that can be modified, such as a list of names or a list of numbers that can be sorted or filtered. |

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

Ans14

my\_tuple = (42,) without the comma it is not a tuple

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

Ans 15

To get a list value's tuple form

my\_list = [1, 2, 3]

my\_tuple = tuple(my\_list)

print(my\_tuple) # Output: (1, 2, 3)

To get a tuple value's list form

my\_tuple = (1, 2, 3)

my\_list = list(my\_tuple)

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

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

Ans 16

Variables that "contain" list values are not lists themselves, but rather references to lists. a variable that holds a list value actually holds a reference to the memory location where the list is stored. List object in memory and then assigns a reference to that object to the variable. When using the variable to access the list, accessing of the list object through its reference is made.

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

Ans17

|  |  |
| --- | --- |
| copy.copy() | copy.deepcopy() |
| creates a shallow copy of the object, meaning that it creates a new object with the same contents as the original, but the contents are references to the same memory locations as the original object. In other words, the new object is a separate object in memory, but its elements are not copied recursively. Therefore, if the original object contains other mutable objects, such as lists or dictionaries, changes made to those mutable objects will affect both the original object and the copied object. | creates a deep copy of the object, meaning that it creates a new object with the same contents as the original, but the contents are completely independent copies of the original object and all of its nested objects. In other words, the new object is a completely separate object in memory with its own set of references to nested objects, and changes made to the nested objects will not affect the original object or the copied object. |