1. What exactly is []?

[] is a list in python. A list is a collection of mutable values (i.e. values that can be changed) of same or different types.

Example: Alphabets = ['a', 'b', 'c', 'd']

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.)

spam.insert(2,'Hello')

spam = [2,4,’Hello’,6,8,10]

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)]?

spam[int(int(33)/11)] 🡪 spam[int(3)]🡪 spam[3]🡪d

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

Spam[-1] = ‘d’

5. What is the value of 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')?

bacon.index(‘cat’) = 1

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

bacon.append(99) will add the value 99 at the end of the list. The list bacon will now be [3.14, ‘cat’,11,’cat’,True,99]

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

The remove() method will remove the first occurance of ‘cat’ from the list. The list bacon will now be [3.14,11,’cat’,True,99]

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

‘+’ is used for list concatenation and ‘\*’ is used for list replication. In list concatenation, elements of one list are added to another.

Example of list concatenation:

spam = [2,4,’Hello’,6,8,10]

bacon = [3.14,11,’cat’,True,99]

newlist = spam + bacon

newlist is now [2, 4, 'Hello', 6, 8, 10, 3.14, 11, 'cat', True, 99]

Example of list replication:

spam = [2, 4, ’Hello’, 6, 8, 10]

newlist2 = spam\*2

newlist2 is now [2,4,’Hello’,6,8,10,2,4,’Hello’,6,8,10]

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

The method append() adds elements at the end of the list. insert(position, new\_value) adds the new\_value at the specified position in the list.

spam.append(‘world’) will result in the list spam = [2,4,’Hello’,8,10,’world’]

spam.insert(3,’world’) will result in the list spam = [2,4,’Hello’,’world’,8,10]

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

The two built-in methods for removing items from the list are pop() and remove().

**pop()** method removes items from the list using an index and returns the deleted value from the list. If no index is given it removes the last element from the list. Syntax: list\_name.pop(index)

Example:

bacon = [2,4,6,8,10,2,5,10,3,4]

bacon.pop(5) will return the element at 5th index which is 2

**remove()** method deletes items from the list using values. It removes the first matching element from the list. Syntax: list\_name.remove(value).

Example:

spam = [2,4,6,8,10,2,5,10,3,4]

spam.remove(10) will result in [2,4,6,8,2,5,10,3,4]

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

Lists and string are both **sequences**. A sequence is a group of items which follow deterministic ordering i.e. the order in which we input an item is the same in which we get an item out of them.

A major difference between list and string is while string is a sequence of characters, a list can have multiple items of different data types within it.

Secondly, **strings are immutable** while **lists are mutable objects**. Immutable means once created the value of the object cannot be changed and mutable means the value of the object can be changed after its creation.

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

|  |  |
| --- | --- |
| **Tuples** | **Lists** |
| Tuples are immutable objects i.e. tuples cannot be changed after creation | Lists are mutable objects i.e. lists can be changed after creation |
| Created by placing elements inside () bracket | Created by placing elements inside [] bracket |
| There can be any number of elements inside tuple and they can be of different data types (int, float, string etc.) | Similar to tuples there can be any number of elements inside lists and they can be of different types (int, float, string etc.) |
| Example of tuples: (‘a’, ’b’, 1, 2, 3) | Example of Lists: [‘a’, ‘b’, ‘c’, 1, 2, 3] |
| Tuples are faster than lists as they are static in nature | Lists are slower than tuples. |
| Tuples consume less memory | Lists consume more memory than tuples |
| Less number of built-in methods | Large number of built-in methods |

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

t = (42)

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

**Converting a list to tuple**

1. creating an integer list: IntList = [1,2,3,4]

2. converting the list to tuple by typecasting will give the result as follows:

IntTuple = tuple(IntList)

IntTuple = (1,2,3,4)

**Converting a tuple to list**

1. creating a tuple: IntTuple = (1,2,3,4)

2. converting the tuple to list by typecasting will give the result as follows:

IntList = List(IntTuple)

IntList = [1,2,3,4]

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

They contain references to the list object created in the memory

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

**Shallow Copy:** copy.copy() creates a shallow copy of the list. It creates a copy of the list object and copies the references of child objects of the old list (i.e. the elements of the old list) to the new list object

**Example:**

oldList = [“ABC”,”DEF”,”GHI” ,”JKL”]

newList = copy.copy(oldList)

newList will be [“ABC”,”DEF”,”GHI” ,”JKL”]

Now if we modify the 3rd element of the newList to contain a different string, the newList will be as follows:

newList[2] = “AAA”

newList = [“ABC”, “DEF”, “AAA”, “JKL”]

The oldList in this case remains unmodified and will be [“ABC”, “DEF”, “GHI”, “JKL”]

Figure below shows the following operations

1. creation of object oldList in memory

2. creation of object newList by copy() method

3. Modification of newList object

Diagram

Description automatically generated

A picture containing graphical user interface

Description automatically generated

**Deep Copy**

copy.deepcopy() creates a deep copy of the list. This method creates a copy of the list object and then recursively copies the child objects(elements) of the old list into new list.

Example:

oldFruits = [“Apple”, ”Grapes”, [“Strawberry”, ”Mango]]

newFruits = copy.deepcopy(oldFruits)

newFruits list will be [“Apple”, ”Grapes”, [“Strawberry1”,” Mango”]]

Now if we modify the newList as follows :

newFruits[2][0] = “Oranges”

newFruits list will now have the elements [“Apple”, ”Grapes”, [“Oranges”,” Mango”]]

Figure below shows the following operations

1. creation of object oldFruits in memory

2. creation of object newFruits by deepcopy() method

3. Modification of newFruits object

Diagram

Description automatically generated

Diagram

Description automatically generateds