**Python Assignment –4**

**By Abhishek Sachan**

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

In Python, **[]** is an empty list. It is a built-in data structure that is used to store a collection of items. Lists are ordered, mutable (i.e., their contents can be changed), and can contain elements of different data types such as integers, strings, and even other lists.

To create a list with items, you can enclose the items in square brackets, separated by commas. For example, **[1, 2, "three", True]** creates a list with four items of different data types.

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

Spam[2]=”hello”

Print(Sapm)

#output 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)]?**

It will print d

‘3’\*2 give 33 int will convert string to numeric

33/11 give 3

print(Spam [3])

#give d as output

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

spam [-1] means –ve indexing means the values is moving from right to left.

So it will be print ‘d’

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

It will be print string 2 character

Print(spam[:2]) #will give ab

Thus, **spam[:2]** creates a slice that starts at the beginning of the list (since **a** is not specified) and ends at the element with index **2-1 = 1**. This returns a new list containing the first two elements of **spam**: **['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’)?**

Index will give the index of the first occurrence which is 1.

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

append is always add new value at the end of list and file

Bacon new list [3.14,’cat’,11,’cat’,True,99].

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

remove() will be delete the first occurrence of the character

so bacon.remove(‘cat’) delete the first occurrence of character from the list then the new list will be look like.

Bacon[3.14,11,’cat’,true].

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

+ are the list concatenation operator which are used to concatenate two new list

For eg:

List1=[1,2,3,4]

List2=[5,7,8]

Newlist=List1+List2

print(Newlist)#1,2,3,4,5,7,8

\* are replicating operator it will repeat same list many time you want

For eg:

List1=[1,2,3,4]

Newlist=List1\*2

print(Newlist)#1,2,3,4,1,2,3,4

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

The **append()** and **insert()** methods are both used to add new elements to a list in Python, but they differ in where the new element is added:

* The **append()** method adds a new element to the end of the list.
* my\_list = [1, 2, 3]  
  my\_list.append(4)  
  print(my\_list)
* Output: **[1, 2, 3, 4]**
* The **insert()** method adds a new element to the list at a specified position, shifting the existing elements to the right. The method takes two arguments: the index at which to insert the new element, and the value of the new element.
* my\_list = [1, 2, 3]  
  my\_list.insert(1, 4)  
  print(my\_list)
* Output: **[1, 4, 2, 3]**

As shown in the examples above, **append()** is typically used to add elements to the end of a list, while **insert()** is used to add elements at a specific position within the list.

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

remove() and del[] are two method of deleting the list

Where remove() will remove the left to right list and always delete the first occurrence of character by typing value

While del is remove the any by their index position.

For eg

List=[1,2,3,4,5]

List.remove(5) # output [1,2,3,4]

del List[1] # output [1,3,4]

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

1. Both lists and strings are ordered sequences of elements.
2. Both can be accessed using indexing and slicing. For example, we can access the first element of a list or a string using **[0]** and we can get a substring or sublist using slicing like **my\_list[2:5]** or **my\_string[1:4]**.
3. Both can be concatenated using the **+** operator. For example, we can concatenate two strings like this: **'hello' + 'world'**.
4. Both can be repeated using the **\*** operator. For example, we can repeat a string three times like this: **'spam' \* 3**

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

In Python, lists and tuples are both sequence data types, but they have some important differences:

1.Mutability: Lists are mutable, meaning you can change, add, or remove elements after the list is created. Tuples are immutable, meaning once they are created, you cannot modify their contents.

2.Syntax: Lists are defined using square brackets, while tuples use parentheses. For example, a list would look like this: **my\_list = [1, 2, 3]**, while a tuple would look like this: **my\_tuple = (1, 2, 3)**.

3.Performance: Tuples are generally faster and more memory-efficient than lists, especially for large data sets or when iterating over data.

In summary, lists are mutable, defined with square brackets, and are used for collections of similar items, while tuples are immutable, defined with parentheses, and are often used to represent a collection of related values.

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

A=(42)

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

You can convert a list to a tuple using the **tuple()** function. Here's an exampl

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

Similarly, you can convert a tuple to a list using the **list()** function. Here's an example:

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

In both cases, the original sequence is converted to the other type of sequence, with the same elements in the same order.

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

Variables that "contain" list values are just references to the list objects. In other words, a variable that refers to a list doesn't contain the list itself, but rather a pointer or reference to the location in memory where the list is stored.

When you assign a list to a variable, Python creates a new list object in memory and assigns a reference to that object to the variable. You can think of the variable as pointing to the list object, rather than containing the list itself.

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

**copy.copy()**: This function creates a shallow copy of an object. In a shallow copy, the new object is a separate copy of the original object, but any mutable objects nested within the original object are still references to the same objects. This means that if you modify a mutable object nested within the copied object, the original object will also be modified. Here's an example:

import copy  
original\_list = [1, 2, [3, 4]]  
copied\_list = copy.copy(original\_list)  
copied\_list[2].append(5)  
print(original\_list) # Output: [1, 2, [3, 4, 5]]  
**copy.deepcopy()**: This function creates a deep copy of an object. In a deep copy, both the top-level object and any nested mutable objects are copied, so that the new object is completely independent of the original object. This means that if you modify a mutable object nested within the copied object, the original object will not be modified. Here's an example:

import copy  
original\_list = [1, 2, [3, 4]]  
deepcopied\_list = copy.deepcopy(original\_list)  
deepcopied\_list[2].append(5)  
print(original\_list) # Output: [1, 2, [3, 4]]

In summary, **copy.copy()** creates a shallow copy where nested mutable objects are still references to the original objects, while **copy.deepcopy()** creates a deep copy where both the top-level object and any nested mutable objects are copied.