**Assignment-4**

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

**Ans:-** These are the square brackets used to cerate a list. In list values enclosed in square brackets.

1. **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:-** From above question we can create spam variable as follows

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

To assign 3rd value as “hello” we can use below statement.

Spam.insert(2,”hello”)

A computer screen with text

Description automatically generated

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

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

**Ans:- “d”**

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

**Ans:- “d”**

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

**Ans:- ['a', 'b']**

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

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

**Ans:- 1**

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

**Ans:-** Final output will be **[3.14, 'cat', 11, 'cat', True, 99]**

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

**Ans:-** Final output will be **[3.14, 11, 'cat', True]**

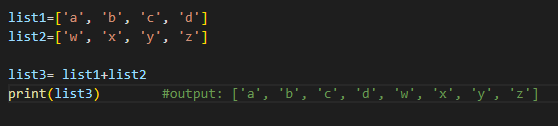
1. **What are the list concatenation and list replication operators?**

**Ans:-**  List concatenation operator is used to combine two or more different lists, while list replication operator used to replicate the lists.

* List Concatenation operator:- The list concatenation operator is represented by the plus sign (+). It allows to concatenate or join two or more lists into a single list.

When we use the “+” operator between two lists, a new list is created that contains all the elements from both lists in the order they appear.

e.g.;-



* List Replication Operator:

The list replication operator is represented by the asterisk (\*). It allows you to create a new list by replicating the elements of an existing list multiple times. When you use the \* operator with a list and an integer, a new list is created with the elements of the original list repeated the specified number of times.

e.g.=

A screenshot of a computer

Description automatically generated

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

**Ans:- append()** and **insert()** are the methods of list. Both are used to add new value into the list. But major difference between them is append() add the value at the end of list values while by using insert() we can add new value at any position of list.

append() having only one attribute in it while insert() requires two attributes to execute.

e.g.:-

A screenshot of a computer

Description automatically generated

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

**Ans:-** There are three methods to remove the items from Python list.

* **pop() Method:-** by using this method we canremove the last value of list. This written as list1.pop().

**Example:-**

**A computer screen with numbers and symbols

Description automatically generated**

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

**Example:-**

**A screenshot of a computer

Description automatically generated**

* **‘del’ statement:**

The ‘del’ statement is a general-purpose statement in Python used to delete objects, including elements from a list. It can be used to remove an item from a list based on its index or to remove a slice of elements from a list.

Example:-

A screenshot of a computer program

Description automatically generated

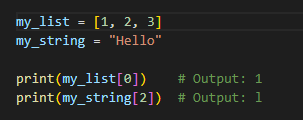
1. **Describe how list values and string values are identical.**

**Ans**:- In Python, list values and string values share some similarities and behaviours, which make them appear identical in certain aspects. Some of them given below.

* **Sequence and Index based access:-**

Both lists and strings allow sequential and index-based access to their elements. Elements in both lists and strings are ordered and can be accessed using indices. The first element is at index 0, the second element at index 1, and so on.

Example:



* **Iteration with for Loop:**

Both lists and strings can be iterated over using a **for** loop. This allows us to process each element or character in a sequential manner.

Example:

A screen shot of a computer program

Description automatically generated

* **Slicing:**

Both lists and strings support slicing, which allows us to extract a portion of the sequence based on start and end indices.

Example:

A screenshot of a computer program

Description automatically generated

1. **What's the difference between tuples and lists?**

**Ans:** List and tuples are the type of data structure in Python. However they have some key differences listed below:

* Lists are mutable , means we can modify, add, remove values from the list. While tuples are immutable data type we cannot change any values from tuple once we define it.
* The syntax of both data type is different. We write a list in closed square “[ ]“ brackets while for tuple we use closed round “( )” brackets.
* Lists are generally slightly slower in terms of performance as compared to tuples because of its mutability.

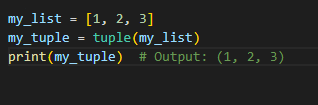
1. **How do you type a tuple value that only contains the integer 42?**

**Ans:** tuple1= (42)

1. **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 a tuple form in Python, you can use the **tuple()** function.

**e.**g.:



On the other hand, to convert a tuple value to a list form, you can use the **list()** function.

e.g.:

A computer screen shot of a program

Description automatically generated

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

**Ans:** This means that the variable holds a pointer or memory address that points to the actual list data structure stored elsewhere in memory.

In Python, objects such as lists are stored separately in memory, and variables act as labels or references to those objects. When you assign a list to a variable, the variable does not store the entire list's contents, but rather holds a reference to the list's memory location.

e.g.: my\_list= [1, 2, 3]

In this example, the variable “my\_list” does not contain the list itself but instead contains a reference to the memory location where the list object “[1, 2, 3]” is stored.

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

**Ans:-**

In Python's copy module, there are two functions available to create copies of objects: **copy.copy()** and **copy.deepcopy().**

* **copy.copy():**

The copy.copy() function creates a shallow copy of an object. It constructs a new object and populates it with references to the same child objects as the original. In other words, it creates a new object but still refers to the original nested objects.

Shallow copies are useful when you want to create a new object that shares the structure of the original but allows modifications without affecting the original object.

Example:

A screen shot of a computer program

Description automatically generated

In this example**, copy.copy()** creates a shallow copy of **list1** called **list2**. When **list2[1].append(4)** is called, the modification is reflected in both **list1** and **list2.**

* **copy.deepcopy():**

The **copy.deepcopy()** function creates a deep copy of an object. It constructs a new object and recursively copies all child objects found in the original, creating independent copies. In other words, it creates a completely independent new object with no shared references to nested objects.

Deep copies are useful when you want to create a new object that is fully independent of the original, allowing modifications without affecting the original or its nested objects.

Example:

A screenshot of a computer program

Description automatically generated

In this example, **copy.deepcopy()** creates a deep copy of **list1** called **list2**. When **list2[1].append(4)** is called, only list2 is modified, and list1 remains unchanged.