**Module 14) Python – Collections, functions and Modules**

**Accessing List:**

1. **Understanding how to create and access elements in a list**

We can create a Python list by placing elements within square brackets [ ]. Unlike sets, there is no need for built-in functions to create a list. Also, it can contain mutable items. Make sure that the items are separated by a comma (,) in square brackets.

**Example:**

x = [2, 3, -7, 6.7, ‘a’]

Each element stored in a list is related to unique integer number called an index. In a list, the first element is indexed as 0, the second as 1, and so on. This means that if a list contains seven items, it will have an index ranging from 0 to 6. We can access these elements in the list by mentioning the index with index operator [ ] preceded by the name of the list. The index must be an integer. Also, we can access nested lists using nested indexing.

**Example:**

x = [2, 3, -7, 6.7, ‘a’]

print(x[0])

print(x[1])

**Output:**

2

3

1. **Indexing in lists (positive and negative indexing).**

In Python, lists are zero-indexed, meaning that the first element is at index 0, the second element at index 1, and so on. Negative indexes, shows the beginning from the end. So, -1 means the last time, -2 means the second last item, and so on. It represents positions from the end of an array. We don’t have to compute the offset as in List, but we can simply write List.

**Example:**

x = [2, 3, -7, 6.7, ‘a’]

print(x[-1])

print(x[-2])

**Output:**

a

6.7

1. **Slicing a list: accessing a range of elements.**

To access a range of elements in a list, you must slice it. One method is to utilize the simple slicing operator, i.e. colon (:) With this operator, one can define where to begin slicing, and where to terminate slicing, and the step. List slicing creates a new list from an old one.

**Syntax:**

List[start: stop: stop]

where,

* **start –** Index position from where the slicing will start in a list
* **stop** – index position till which the slicing will end in a list
* **step –** number of steps, i.e. the start index is changed after every n step, and list slicing is performed on that index

**Note -**  In Python, indexing starts from 0, and not 1.

**Get all the Items**

my\_list = [1,3,5,7,9]

print(“Item in List are:”,my\_list[:])

**Output:**

Items in List are: [1,3,5,7,9]

In above program, to get all the elements of the list we use ‘:’. This is similar to the statement print(my\_list)

**Get all the Items After a Specific Position**

my\_list = [1,3,5,7,9,11,13,15]

print(“Items in List after 3rd position are:”,my\_list[3:])

**Output:**

Items in List after 3rd position are: [7,9,11,13,15]

Int this program, we have used the start parameter by specifying an integer value that indicates the starting position of the slicing of the list. The elements at index 3 and all the elements after index 3 will be displayed.

**Get all the Items Before a Specific Position**

my\_list = [1,3,5,7,9,11,13,15]

print(“Items in List till 4th index are:”,my\_list[:4])

**Output:**

Items in List till 4th index are: [1,3,5,7]

In this program, we have used the stop parameter by specifying an integer value that indicates the ending position of the slicing of a list. The items before index 4 are sliced in the example. Element on index position 4 is not included.

**Get all the Items from One Position to Another Position**

my\_list = [1,3,5,7,9,11,13,15]

print(“Items in List from 1st to 6th index are:”,my\_list[1:6])

**Output:**

Items in List from 1st to 6th index are: [3,5,7,9,11]

If you wish to display all the elements between two specific indices, put them before and after the ‘:’ symbol. In the preceding example, my list[1:6] returns the elements between the first and sixth positions. The beginning position (i.e. 1) is included, but the finishing position (i.e. 6) is not.

**Get the Items at Specified Intervals**

my\_list = [1,3,5,7,9,11,15,17,19]

print(“Items in List at step of 2 are:”,my\_list[::2])

**Output:**

Items in List at step of 2 are: [1,5,9,13,17]

In this type, we have made use of the step parameter for slicing. The step parameter is an integer value that prints the list elements after specific intervals. For example, in the above program, we have declared a step of 2. So the elements in the position of 0,2,4,6, and so on will be printed.

If you want the indexing to begin with the final item, we can use the negative sign ‘-‘. In the below example, Items at interval 2 beginning with the last index are sliced.

my\_list = [1,3,5,7,9,11,13,15,17,19]

print(“Items in List are:”,my\_list[::-2])

**Output:**

Items in List are: [19,15,11,7,3]

**List Operations:**

1. **Common list operations: concatenation, repetition, membership**

It is the computation or actions applied to the variable containing the list of data types in and expression.

List manipulation in Python can be done using various operators like concatenation (+), repetition (\*), slicing of the list, and membership operators (in/not in). So, Let’s understand each operation in brief.

* Concatenation operator (+)
* Repetition operator (\*)
* Membership Operator (in, not in)
* **Concatenation operator (+)**

The (+) operator is used to add to two lists.

The syntax of the given operation is*: List1 + List2*

>>> lst1 = [12,43, 53]

>>> lst2 = [78,90]

>>> print(lst1 + lst2)

**Output:**

[12,43,53,78,90]

* **Repetition operator (\*)**

Like string, (\*) operator replicates the string number of specified times.

The syntax of the given operation: *List \* n*

>>> lst1 = [12,43,56]

>>> print( lst1 \* 3 )

**Output:**

[12,43,56,12,43,56,12,43,56]

* **Membership Operators (in, not in)**

The membership operator checks whether an element exists in the given list.

* **In:** return true if an element exists in the given list; False otherwise
* **Not in:** return true if an element does not exist in the given list; False otherwise.

>>> lst1 = [12,43,56,78,90]

>>> 56 in lst1

>>> 12 not in lst1

**Output:**

True

False

1. **Understanding list methods like append (), insert (), remove (), pop ().**

* **append():** Add its argument as a single item to the end of a list.
* **insert():** Insert an element at a specified position.
* **pop():** Removes and returns the element at the specified position. (or the last element if no index is specified).
* **remove():** Remove the first occurrence of a specified element.
* **append():**

List have several useful built-in methods, one of which is the append methods. When calling append on a list, we append an object to the end of the list:

>>> my\_list = [1,2]

>>> my\_list.append(‘a’)

>>> print(my\_list)

**Output:**

[a,2,’a’]

* **pop():**

The pop() method removes and returns the last item by default unless you give it an index argument.

Here are a couple of examples that demonstrate both the default behavior and the behavior when given an index:

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

>>> my\_list.pop()

>>>print(my\_list)

**Output:**

[1,2,3,4]

* **remove():**

If you want to remove a specific value from the list, use the remove() method. This method will remove the first occurrence of the given object in a list. Let’s demonstrate this by remove the number two from my\_list.

>>> my\_list = [1,2,3,4]

>>> my\_list.remove(2)

>>> print(my\_list)

**Output:**

[1,3,4]

* **insert();**

The insert() method inserts an element at a specified position within a list.

>>> fruits = [‘apple’, ‘banana’, ‘cherry’]

>>> fruits.insert(1, ‘orange’)

>>> print(fruits)

**Output:**

[‘apple’, ‘orange’, ‘banana’, ‘cherry’]

**Working with Lists:**

1. **Iterating over a list using loops**

The simplest and the most common way to iterate over a list is to use a for loop. This method allows us to access each element in the list directly.

**Example:** Print all elements in the list one by one using for loop.

a = [1,3,5,7,9]

for val in a:

print(val)

**Output:**

1

3

5

7

9

1. **Sorting and reversing a list using sort(), sorted(), and reverse().**

* **sort() method:**

The sort() method in Python is a built-in function that allows us to sort the elements of a list in ascending or descending order and it modifies the list in place which means there is no new list created. This method is useful when working with lists where we need to arranged the elements in a specific order, whether numerically of alphabetically.

Below is a simple example that use sort() method to arrange a list of integer values in ascending order.

**Example:**

a = [5,2,9,1,5,6]

a.sort()

print(a)

**Output:**

[1,2,5,5,6,9]

* **sorted() function:**

sorted() function returns a new sorted list from the elements of any iterable like (e.g., list, tuples, strings). It creates and returns a new sorted list and leaves the original iterable unchanged.

Let’s start with a basic example of sorting a list of numbers using the sorted() function.

**Example:**

a = [4,1,3,2]

b = sorted(a)

print(b)

**Output:**

[1,2,3,4]

1. **Basic list manipulations: addition, deletion, updating, and slicing.**

Python list methods are built-in function that allow us to perform various operations on lists, such as adding, removing, or modifying elements. In this article, we’ll explore all Python list methods with a simple example.

* **Addition (Appending and Inserting elements)**

You can add elements to a list using two common methods: append() and insert().

* + **append():** Adds an element to the end of the list.
  + **insert():** Adds an element at a specified position.

**Example:**

fruits = [‘apple’, ‘banana’, ‘orange’]

fruits.append(‘grapes’)

print(“After append:”,fruits)

fruits.insert(1, ‘mango’)

print(“After insert:”,fruits)

**Output:**

After append: [‘apple’, ‘banana’, ‘orange’, ‘grapes’]

After insert: [‘apple’, ‘mango’, ‘banana’, ‘orange’, ‘orange’]

* **Deletion (Removing elements)**

You can remove elements from a list using remove() or pop()

* + **remove():** Removes the first occurrence of a specific value.
  + **pop():** Removes and returns an element at a specified index.

**Example:**

fruits = [‘apple’, ‘mango’, ‘banana’, ‘orange’, ‘grapes’]

fruits.remove(‘banana’)

print(“After remove:”,fruits)

last\_fruit = fruits.pop()

print(“After pop:”,fruits)

print(“Popped element:”,last\_fruit)

**Output:**

After remove: [‘apple’, ‘mango’, ‘orange’, ‘grapes’]

After pop: [‘apple’, ‘mango’, ‘orange’]

Popped element: grapes

* **Updating (Modifying elements)**

You can update an element in the list by accessing its index and assigning a new value.

**Example:**

fruits = [‘apple’, ‘mango’, ‘orange’]

fruits[1] = ‘pineapple’

print(“After update:”, fruits)

**Output:**

After update: [‘apple’, ‘pineapple’, ‘orange’]

* **Slicing (Extracting parts of a list)**

You can extract a portion of a list using slicing by specifying a range of indices.

**Example:**

fruits = [‘apple’, ‘banana’, ‘orange’, ‘grapes’, ‘pineapple’]

sliced\_fruits = fruits[1:3]

print(“Sliced list (1:3):”, sliced\_fruits)

sliced\_fruits2 = fruits[:2]

print(“Sliced list (:2):”, sliced\_fruits2)

sliced\_fruits3 = fruits[2:]

print(“Sliced list (2:):”, sliced\_fruits3)

**Output:**

Sliced list (1:3): [‘banana’, ‘orange’]

Sliced list (:2): [‘apple’, ‘banana’]

Sliced list(2:): [‘orange’, ‘grapes’, ‘pineapple’]

**Tuple:**

1. **Introduction to tuples, immutability**

Python Tuple is a collection of objects separated by commas. A tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

**Example:**

t = (10,20,30)

print(t)

print(type(t))

**Output:**

(10,20,30)

<class ‘tuple’>

Unlike Python lists, tuples are immutable. Some characteristics of Tuples in Python.

* Like Lists, tuples are ordered and we can access their elements using their index values
* We cannot update items to a tuple once it is created.
* Tuples cannot be appended or extended.
* We cannot remove items from a tuple once it is created.

**Example:**

t = (1,2,3,4,5)

print(t[1])

print(t[4])

t = (1,2,3,4,2,3)

t[1] = 100

print(t)

**Output:**

2

5

(1,2,3,4,2,3)

Traceback (most recent call last):

File “Solution.py”, line 12, in <module>

T[1] = 100

TypeError: ‘tuple’ object does not support item assignment

1. **Creating and accessing elements in a tuple**

We can access elements in a tuple in the same way as we access elements in a list. Python follows 0-based indexing, so a tuple with n elements has indices from 0 through n-1. To access an index in a tuple, we use the index operator [ ].

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| ‘Welcome’ | ‘to’ | ‘Python’ | ‘Have’ | ‘a’ | ‘great’ | ‘day!’ |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

**Example:**

x = (‘welcome’, ‘to’, ‘Python’)

print(x[0])

print(x[1])

print(x[2])

print(x[-1])

print(x[-2])

print(x[-3])

**Output:**

Welcome

to

Python

Python

to

Welcome

1. **Basic operations with tuples: concatenation, repetition, membership**

* **Concatenation:**

Concatenating tuples means joining two or more tuples together. You can concatenate tuples using the + operator.

**Example:**

tuple1 = (1,3,2)

tuple2 = (4,6,5)

result = tuple1 + tuple2

print(result)

**Output:**

**(1,3,2,4,6,5)**

* **Repetition**

Repetition involves repeating the elements of a tuple multiple times. You can use the \* operator to repeat a tuple a specific number of times.

**Example:**

tuple1 = (1,2,3)

result = tuple \* 3

print(result)

**Output:**

(1,2,3,1,2,3,1,2,3)

* **Membership**

Membership checks whether a particular element exists in a tuple. You can use the in or not in operators for this.

**Example:**

tuple1 = (1,2,3,4,5)

print(3 in tuple1)

print(6 not in tuple1)

**Output:**

True

True

**Accessing Tuples:**

1. **Accessing tuple elements using positive and negative indexing**

* **Positive Indexing**

As we have seen that tuple items have index, as such we can access items using these indexes.

**Example:**

country = (‘Spain’, ‘Italy’, ‘India’, ‘England’, ‘Germany’)

print(country[1])

print(country[3])

print(country[0])

**Output:**

Italy

England

Spain

* **Negative Indexing**

Similar to positive indexing, negative indexing is also used to access items, but from the end of the tuple. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.

**Example:**

country = (‘Spain’, ‘Italy’, ‘India’, ‘England’, ‘Germany’)

print(country[-1])

print(country[-2])

print(country[-3])

**Output:**

Germany

India

Italy

1. **Slicing a tuple to access ranges of elements.**

You can also use slicing to access a range of items in a tuple. The syntax for slicing is tuple [start:stop:step].

* Start is the index at which the slice starts (inclusive).
* Stop is the index at which the slice ends (exclusive).
* Step defines the increment between elements in the slice(optional).

**Example:**

my\_tuple = (‘apple’, ‘banana’, ‘cherry’, ‘date’, ‘elderberry’)

print(“Sliced tuple (1:4):”, my\_tuple[1:4])

print(“Slicedtuple (:3):”, my\_tuple[:3])

print(“Sliced tuple (2:):”, my\_tuple[2:])

print(“Sliced tuple (::2):”, my\_tuple[::2])

**Output:**

Sliced tuple (1:4): (‘banana’, ‘cherry’, ‘date’

Sliced tuple (:3): (‘apple’, ‘banana’, ‘cherry’)

Sliced tuple (2:): (‘cherry’, ‘date’, ‘elderberry’)

Sliced tuple (::2): (‘apple’, ‘cherry’, ‘elderberry’)