Data Structures in Python

In python, we have **data types** like *str*, *float*, *int*, *bool* etc. which help us know the type of the variables we store our information/data in. Further, these variables having the same or different data types into a structure known as a **data structure.**
The basic data structures in Python which are inbuilt in Python and do not require any external libraries/packages to be installed first in order to use them. These are ***Tuple, List, Dictionary*** **and** ***Set***. They have few properties in common.

1. All four are 1D (**one-dimensional**) i.e. data can be stored either in the form of a single row or a single column.

2. All four are **heterogeneous** in nature so we can store any type of data in these i.e. the entries can have the same or different data types.

Python Data Structures – Lists

A list is defined as an ordered collection of items, and it is one of the essential data structures when using Python to create a project. The term "ordered collections" means that each item in a list comes with an order that uniquely identifies them. The order of elements is an inherent characteristic that remains constant throughout the life of the list.

Since everything in Python is considered an object, creating a list is essentially creating a Python object of a specific type. When creating a list, all the items in the list should be put in square brackets and separated by commas to let Python know that a list has been created. A sample list can be written as follows:

List_A = [item 1, item 2, item 3….., item n]

Lists can be nested

A list can be nested, which means that it can contain any type of object. It can include another list or a sublist – which can subsequently contain other sublists itself. There is no limit to the depth with which lists can be nested. An example of a nested list is as follows:

List_A = [item 1, list_B, item 3….., item n]

Lists are mutable

Lists created in Python qualify to be mutable because they can be altered even after being created. A user can search, add, shift, move, and delete elements from a list at their own will. When replacing elements in a list, the number of elements added does not need to be equal to the number of elements, and Python will adjust itself as needed.

It also allows you to replace a single element in a list with multiple elements. Mutability also enables the user to input additional elements into the list without making any replacements.

Python Data Structures – Tuples

A tuple is a built-in data structure in Python that is an ordered collection of objects. Unlike lists, tuples come with limited functionality.

The primary differing characteristic between lists and tuples is mutability. Lists are mutable, whereas tuples are immutable. Tuples cannot be modified, added, or deleted once they've been created. Lists are defined by using parentheses to enclose the elements, which are separated by commas.

The use of parentheses in creating tuples is optional, but they are recommended to distinguish between the start and end of the tuple. A sample tuple is written as follows:

tuple_A = (item 1, item 2, item 3,…, item n)

Empty and One Single Item Tuple

When writing a tuple with only a single element, the coder must use a comma after the item. This is done to enable Python to differentiate between the tuple and the parentheses surrounding the object in the equation. A tuple with a single item can be expressed as follows:

some_tuple = (item 1,)

If the tuple is empty, the user should include an empty pair of parentheses as follows:

Empty_tuple= ( )

Why Tuples are Preferred over Lists

Tuples are preferred when the user does not want the data to be modified. Sometimes, the user can create an object that is intended to remain intact during its lifetime. Tuples are immutable, so they can be used to prevent accidental addition, modification, or removal of data.

Also, tuples use less memory, and they make program execution faster than using lists. Lists are slower than tuples because every time a new execution is done with lists, new objects are created, and the objects are not interpreted just once. Tuples are identified by Python as one immutable object. Hence, they are built as one single entity.

Python Data Structures – Sets

A set is defined as a unique collection of unique elements that do not follow a specific order. Sets are used when the existence of an object in a collection of objects is more important than the number of times it appears or the order of the objects. Unlike tuples, sets are mutable – they can be modified, added, replaced, or removed. A sample set can be represented as follows:

set_a = {“item 1”, “item 2”, “item 3”,….., “item n”}

One of the ways that sets are used is when checking whether or not some elements are contained in a set or not. For example, sets are highly optimized for membership tests. They can be used to check whether a set is a subset of another set and to identify the relationship between two sets.

Python Data Structures – Dictionary

Another useful data type built into Python is the dictionary (see Mapping Types — dict). Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

It is best to think of a dictionary as a set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

The main operations on a dictionary are storing a value with some key and extracting the value given the key. It is also possible to delete a key:value pair with del. If you store using a key that is already in use, the old value associated with that key is forgotten. It is an error to extract a value using a non-existent key.