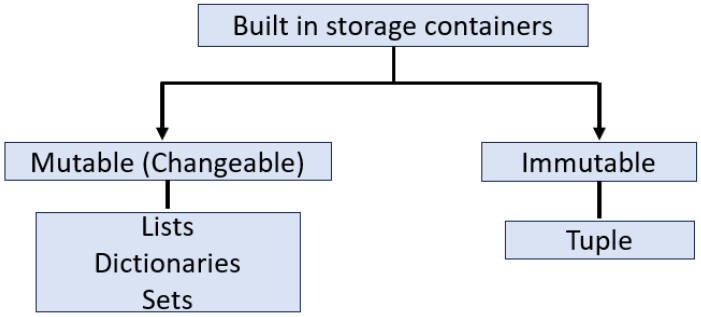
DAILY DSA | DAY-4 | Built-in Data structures in Python|
-GOPALKRISHNA A

As we know a data structure is a way of organizing data in computer memory.

This needs efficient storage, retrieval, and modification of data. Knowing what data structures exist and how to use them efficiently in different situations leads to avoiding inefficient memory usage and reducing execution time

In this post, we will go through data storage containers in Python
Categorization of built-in data structures in Python:



In everyday programming, a significant portion, **approximately 70%, of data storage and manipulation revolves around fundamental data structures such as lists, tuples, dictionaries, and sets**.
In our upcoming Daily DSA posts, we will delve into more advanced data structures like stacks and queues

Lists:

- Lists in Python are implemented as dynamic mutable arrays that hold an ordered collection of items
- In Python, a list can contain heterogeneous data types and objects. For instance, integers, strings within the same list
- When creating a list, we do not have to specify in advance the number of elements it will contain; therefore, it can be expanded as we wish, making it dynamic
- Pros:
    - They represent the easiest way to store a collection of related objects
    - They are easy to modify by removing, adding, and changing elements
    - They are useful for creating nested data structures, such as list of lists/dictionaries
- Cons:
    - Lists use more disk space for large data because of their under-the-hood implementation

Dictionaries:

- Dictionaries in Python are very similar to real-world dictionaries (Oxford English Dictionary). These are mutable data structures that contain a collection of **keys** and, associated with them, **values**
- We use dictionaries when we are able to associate (in technical terms, **to map**) a unique key to certain data, and we want to access that data very quickly.
- Pros:
    - Dictionaries make code much easier to read if we need to generate key: pairs
    - We can look up a certain value in a dictionary very quickly

- Cons:
    - They occupy a lot of space. If we need to handle large amounts of data, this is not the most suitable data structure

Sets:

- Sets in Python can be defined as mutable dynamic collections of **immutable unique elements**. Sets may seem very similar to lists, but in reality, they are very different.
- Sets can be used to remove duplicates from a list, they also have unique operations which can be applied to them, such as **set union, intersection**, etc.
- Pros:
    - We can perform unique (but similar) operations
    - They are significantly faster than lists if we want to check whether a certain element is contained in a set
- Cons:

    - Sets are intrinsically unordered. if we care about keeping the order of insertion, they are not the best choice
    - We cannot change set elements by indexing as we can with lists

Tuples:

- Tuples are almost identical to lists, so they contain an ordered collection of elements, except for one property: They are **immutable**. We would use tuples if we needed a data structure that, once created, cannot be modified anymore

- Pros:
    - They are immutable, so once created, we can be sure that they won't change their contents by mistake
    - They can be used as dictionary keys if all their elements are immutable

- Cons:
    - Tuples cannot be copied
    - They occupy more memory than lists

Parameters	List	Tuple	Set	Dictionary
Definition	A list is an ordered, mutable, collection of elements	A tuple is an ordered, immutable collection of elements	A set is an unordered collection of unique elements	A dictionary is an unordered collection of key-value pairs
Syntax	Syntax includes square brackets [,] with ‘,’ separated data	Syntax includes square brackets (,) with ‘,’ separated data	Syntax includes square brackets {,} with ‘,’ separated data	Syntax includes square brackets {,} with ‘,’ separated data
Creation	A list can be created using the list() function or a simple assignment to []	A tuple can be created using the tuple() function	A set dictionary can be created using the set() function	A dictionary can be created using the dict() function
Empty data structure	An empty list can be created by l=[]	An empty tuple can be created by t=()	An empty set can be created by s=set()	An empty dictionary can be created by d={}
Order	It is an ordered collection of data	It is also an ordered collection of data	It is an unordered collection of data	Ordered collection
Duplicate data	Duplicate data entry is allowed in the list	Duplicate data entry is allowed in a Tuple	All elements are unique in a set	Keys are unique, but two different keys can have the same value
Indexing	Has integer-based indexing that starts from ‘0’	Also has an integer-based indexing that starts from ‘0’	Does not have an index-based mechanism	Has a key-based indexing. I.e keys identify the value
Addition	New items can be added using the append() method	Being immutable, new data cannot be added to it	The add() method adds an element to a set	update() method updates specific key-value pair
Deletion	The pop () method allows the deletion of an element	Being immutable no data can be popped/deleted	Elements can be randomly deleted using pop()	Pop(key) removes the specified key along with its value
Sorting	The sort method sorts the elements	Immutable, so the sorting method is not applicable	Unordered, so sorting is not advised	Keys are sorted using the sorted() method
Search	index() returns the index of the first occurrence	index() returns the index of the first occurrence	Unordered, so searching is not applicable	get(key) returns value against specified key
Reversing	reverse() method reverses the list	Immutable, so the reverse method is not applicable	Unordered, so reverse is not advised	No integer-based indexing, so no reversal
Count	count() method returns the occurrence count	count() method returns the occurrence count	count() is not defined for sets	count() not defined for dictionaries