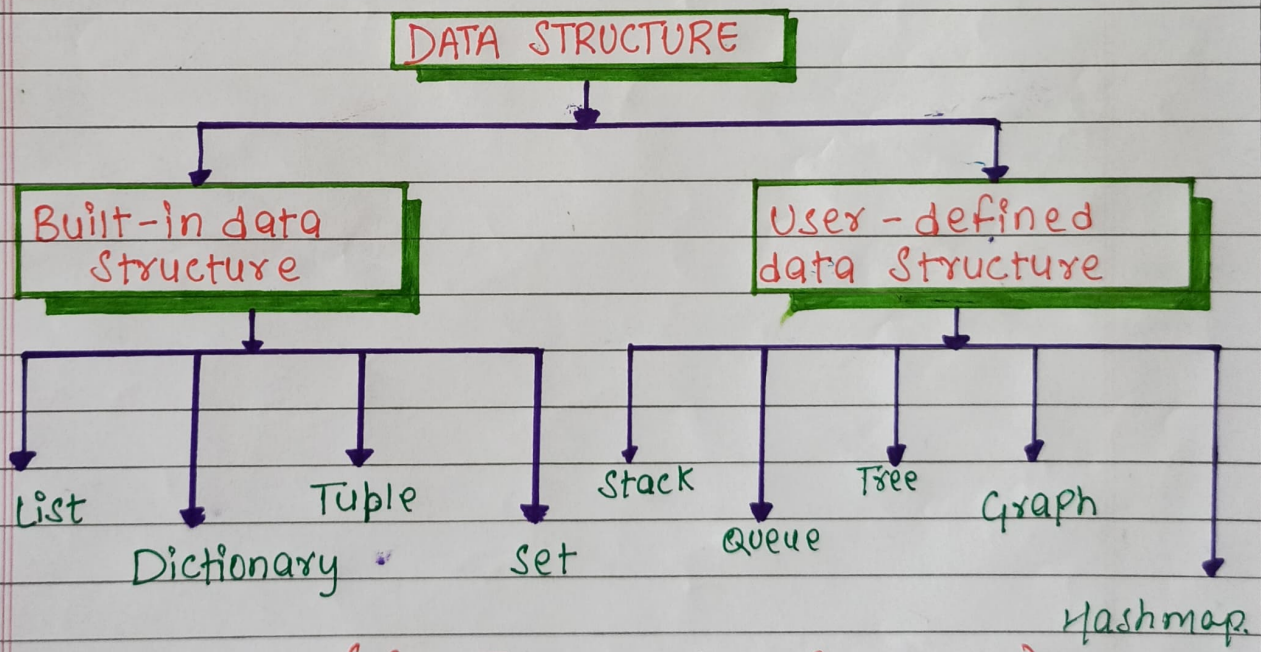
DATA STRUCTURE IN PYTHON

ATUL KUMAR (LINKEDIN).
NOTES GALLERY (TELEGRAM).

What is Data Structure?

Organizing, Managing and storing data is important as it enables easier access and efficient modifications.
Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

Types of data structures in python :-



(fig. Types of data Structures)

Python has implicit support for data structures which enables you to store & access data.

LISTS

Lists are used to store data of different data types in a sequential manner and there are addresses assigned to every element, called index.

Creating a list :- To create a list, you use square brackets and add elements into it accordingly.

```
my_list = [ ]   # Create empty list
print (my_list)
my_list=[1,2,3,'example',3.132] # data
print(my_list)
```


Output :- [ ]
[1,2,3,'example', 3.132].

Deleting elements from list :- To delete elements, use the del keyword which does not return anything back.

```
my_list=[1,2,3,'example', 3.132, 10, 30]
del my_list [5] # delete element at index 5.
print(my_list)
a=my_list.pop(1) # pop element from list
print('popped Element:', a)
my_list.clear( ) # empty the list
print(my_list)
```

Continue ⟹

Output :- [1, 2, 3, 'example', 3.132, 30].
[1, 2, 3, 3.132, 30].
Popped Element : 2
[ ]

Accessing Elements :- Accessing element, you should pass the index values and hence can obtain value.

```
my_list=[1,2,3,'example', 3.132, 10, 30]
for element in my_list: # access one by one.
 print (element)
print(my_list) # access all elements
print(my_list[3]) # access index 3 element.
print(my_list[0:2])#access from 0 to 1, exclude 2.
print(my_list[::-1])# access elements in reverse.
```

Output :-

```
1
2
3
example
3.132
10
30
[1,2,3,'example', 3.132, 10, 30]
example
[1, 2]
[30, 10, 3.132, 'example', 3, 2, 1].
```

TUPLE

Tuples are same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what.

Creating a Tuple :- You can create a tuple using parenthesis or using the tuple ( ) function.

```
#my_tuple = (1,2,3)
print(my_tuple)
```

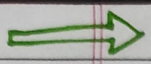
o/p :- (1, 2, 3).

Accessing elements :- Accessing elements is the same as it is for accessing values in lists.

```
my_tuple 2 = (1,2,3,'python') # accessing elements
for x in my_tuple 2:
print(x)
print(my_tuple 2)
print(my_tuple 2 [0]
print(my_tuple 2 [:])
print(my_tuple 2 [3][4])
```


o/p :- 1
2
3

Continue ⟹



python
(1, 2, 3, 'python')
1
(1, 2, 3, 'python')
0

Appending elements :-

To append the values, you use '+' operator which will take another tuple to be appended to it.

```
my_tuple = (1, 2, 3)
my_tuple = my_tuple + (4, 5, 6) # add elements
print (my_tuple)
```

Output :-

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

SETS

Sets are the collection of unordered elements that are unique. If the data is repeated more than one time, it would be entered into the set only once.

Creating a Set :- Sets are created using curly braces but instead of adding key-value pairs you just pass values to it.

```
my_set = {1,2,3,4,5,3,2,5} # Create set
print(my_set)
```


o/p :- {1,2,3,4,5}.

Adding elements :- To add elements, you use the add( ) function and pass the value to it.

```
my_set = {1,2,3}
my_set.add(4) # add element to set.
print(my_set)
```

o/p :- {1,2,3,4}.

Operations in sets :- The different operations on set such as union, intersection and so on are shown below:

```
my_set = {1, 2, 3, 4}
my_set_2 = {3, 4, 5, 6}
print(my_set.union(my_set_2)
print(my_set.intersection(my_set_2)
print(my_set.difference(my_set_2)
print(my_set.Symmetric_diff(my_set_2)
my_set.clear()
print(my_set)
```

- Union() function combines data present in both sets.
- intersection() function finds data present in both sets.
- difference() function deletes data present in both and outputs data present only in set.
- Symmetric difference() function does same as difference() function but outputs data which is remaining in both sets.

Output :-


```
{1, 2, 3, 4, 5, 6}
{3, 4}
{1, 2}
{1, 2, 5, 6}
set()
```

DICTIONARY

Dictionaries are used to store key-value pairs. Key value pair is like phone numbers with contact name simply, let's understand more:

Creating a Dictionary :- To create dictionary, we use curly braces or use dist() function.

```
my_dict = {} # empty dictionary
print(my_dict)
my_dict = {1: 'python', 2: 'Java'} # elements
print(my_dict)
```


O/P :- { }
{1: 'python', 2: 'Java'}.

Deleting key, value pairs :- To delete values, use POP( ) function which returns value that has been deleted.

```
my_dict = {'1: 'python', 2: 'Java', 3: 'Ruby'}
a = my_dict.pop('Third') # pop element.
print('value', a)
print('Dictionary', my_dict)
b = my_dict.popitem() # pop key value - pair.
print('key, value pair :', b)
print('Dictionary', my_dict)
```

Continue ⟹

O/P :-

Value : Ruby
Dictionary : {1 : 'Python', 2 : 'Java'}
Key, value pair : (2, 'Java')
Dictionary {1 : 'Python'}.

Accessing Elements :- You can access elements using keys only. You can either use get ( ) function or just pass the key values and you will be retrieving the values.

```
my_dict = {'first': 'Python', 'second': 'code'}
print(my_dict['first']) # access using keys
print(my_dict.get('second'))
  # access elements using get.
```

O/P :- Python
code