Course name: **Python programming and analytics by Rahul Sir**

Topic name: **data structure**

Video name: **data structures-Manav batch (17-08-2019)**

Video length: **32 minutes 44 seconds**

In python there are 4 types of data structure:

1. **Tuples**
2. **Lists**
3. **Dictionaries**
4. **Sets**

Tuples

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. A tuple can have any number of items and they may be of different types (integer, float, list, string, [in quotes] and Boolean).

```
In [1]: my_tuple =("Python",1, True, False, 2.2 )
        my_tuple

Out[1]: ('Python', 1, True, False, 2.2)

In [2]: type(my_tuple)

Out[2]: tuple

In [3]: len(my_tuple)

Out[3]: 5
```

To know the length of the tuple len() function is used, it tells how many elements are there in the tuple

To know the data type and data structure, type() function is used

How to Access Tuple Items?

You can access tuple items by referring to the index number, inside square brackets

```
In [5]: print( my_tuple[0])
        print( my_tuple[1])
        print( my_tuple[4])

        Python
        1
        2.2
```

```
In [9]: print( type(my_tuple[0]))
        print( type(my_tuple[1]))
        print( type(my_tuple[2]))
        print( type(my_tuple[3]))
        print( type(my_tuple[4]))

        <class 'str'>
        <class 'int'>
        <class 'bool'>
        <class 'bool'>
        <class 'float'>
```

If you want to know the type of the tuple items

```
In [6]: print( my_tuple[-1])
        print( my_tuple[-2])
        print( my_tuple[-3])

        2.2
        False
        True
```

Indexing from the left side starts with 0 and from the right side it is from -1. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

We can use the index ope he index starts from 0.

So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an element outside of tuple (for example, 6, 7,...) will raise an Index Error.

The index must be an integer; so we cannot use float or other types. This will result in Type Error.

```
my_tuple[2] = 4   #Cant alter the tuple
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-97491497cabc> in <module>()
----> 1 my_tuple[2] = 4   #Cant alter the tuple

TypeError: 'tuple' object does not support item assignment
```

Tuples are immutable, so the tuples can't be altered

```
my_tuple.append(25) #Cant append the tuple
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-ef45662b3333> in <module>()
----> 1 my_tuple.append(25) #Cant append the tuple

AttributeError: 'tuple' object has no attribute 'append'
```

```
my_tuple2 = my_tuple + ("Anaconda", 120)
my_tuple2
```

```
('Python', 1, True, False, 2.2, 'Anaconda', 120)
```

There is no append function, if you want to add more items the already existing tuple or join two or more tuples, You can use '+' function

You can specify a range of indexes by specifying where to start and where to end the range. Remember that the first item has index 0.

```
In [11]: my_tuple2[0:2]
Out[11]: ('Python', 1)

In [12]: my_tuple2[0:]
Out[12]: ('Python', 1, True, False, 2.2, 'Anaconda', 120)

In [13]: my_tuple2[:3]
Out[13]: ('Python', 1, True)
```

You cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple. Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely

```
In [14]: del(my_tuple2[0])

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-8dda1fc08c1b> in <module>()
----> 1 del(my_tuple2[0])

TypeError: 'tuple' object doesn't support item deletion
```

You can't delete the items in the tuple but you can delete the entire tuple

```
In [15]: Scores = (10, 8 , 3, 2 , 4, 0 , 0, 3)
         Scores

Out[15]: (10, 8, 3, 2, 4, 0, 0, 3)

In [16]: type(Scores)

Out[16]: tuple

In [17]: sorted_Scores = sorted(Scores, reverse=True)
         sorted_Scores

Out[17]: [10, 8, 4, 3, 3, 2, 0, 0]

In [18]: sorted(Scores)

Out[18]: [0, 0, 2, 3, 3, 4, 8, 10]
```

If you want to sort the items in a tuple, you can do it by using sorted() function.

You can sort in increasing order or descending order

To sort in decreasing order, you need to add (reverse=**true**) in the code as well

Lists:

A list is a collection which is ordered and changeable.

In Python lists are written with square brackets. Lists need not be homogeneous always which makes it a most powerful tool in Python. A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

```
In [22]: my_list = ["Virat", 9.34, 1982, True]
         my_list

Out[22]: ['Virat', 9.34, 1982, True]
```

Elements can be added to the List by using built-in **append()** function. Only one element at a time can be added to the list by using append() method

```
In [23]: my_list.append("Ram")

In [24]: my_list #Append can use in lists

Out[24]: ['Virat', 9.34, 1982, True, 'Ram']
```

```
In [34]: my_list = ["Virat", 9.34, 1982, True]

         my_list.append(['new_item',100])
         my_list

         my_list.append((1,2,3))
         my_list

Out[34]: ['Virat', 9.34, 1982, True, ['new_item', 100], (1, 2, 3)]
```

Also, a list can even have another list as an item. This is called nested list.

```
In [29]: num1 = 23
         tuple1 = (1, 3, 9)
         list1 = ["Sachin", 982, True]

         mixed_list = ["Virat", 9.34, 1982, True, num1, tuple1, list1, [ 34, 23], ('a', 'v')]
         mixed_list #List of lists

Out[29]: ['Virat',
          9.34,
          1982,
          True,
          23,
          (1, 3, 9),
          ['Sachin', 982, True],
          [34, 23],
          ('a', 'v')]
```

How to access elements from a list?

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Trying to access an element other that this will raise an Index Error. The index must be an integer. You can't use float or other data Types this will result into Type Error. Python allows negative indexing for its

sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
['Virat', 9.34, 1982, True, 'Ram']
```

```
print(my_list[0] , "    " , my_list[-5])
print(my_list[1] , "    " , my_list[-4])
print(my_list[2] , "    " , my_list[-3])
print(my_list[3] , "    " , my_list[-2])
print(my_list[4] , "    " , my_list[-1])
```

```
Virat      Virat
9.34      9.34
1982      1982
True      True
Ram      Ram
```

You can specify a range of indexes by specifying where to start and where to end the range. Remember that the first item has index 0.

```
In [30]: my_list = ["Virat", 9.34, 1982, True]
         my_list
Out[30]: ['Virat', 9.34, 1982, True]

In [27]: my_list[:]
Out[27]: ['Virat', 9.34, 1982, True, 'Ram']

In [28]: my_list[1:]
Out[28]: [9.34, 1982, True, 'Ram']

In [29]: my_list[:2]
Out[29]: ['Virat', 9.34]

In [30]: my_list[:-1]
Out[30]: ['Virat', 9.34, 1982, True]
```

method for Addition of elements, **extend()**, this method is used to add multiple elements at the same time at the end of the list.

```
In [31]: my_list = ["Virat", 9.34, 1982, True]

         my_list.extend(['new_item',100])
         my_list

Out[31]: ['Virat', 9.34, 1982, True, 'new_item', 100]
```

To remove any item from the list, you can use "remove" function.

```
In [37]: my_list = ["Virat", 9.34, 1982, True]

         my_list.remove("Virat")
         my_list

Out[37]: [9.34, 1982, True]
```

Lists can be altered with the help of index

```
my_list = ["Virat", 9.34, 1982, True]
print("Before : " , my_list)

my_list[1] = 34.3
print("After : " , my_list)  #lists can be altered
```

```
Before :  ['Virat', 9.34, 1982, True]
After :  ['Virat', 34.3, 1982, True]
```

List items can be deleted using "del" function. The del keyword removes the specified index. The del keyword can also delete the list completely

```
my_list = ["Virat", 9.34, 1982, True]
print("Before : " , my_list)

del(my_list[2])
print("After : " , my_list)
```

```
Before :  ['Virat', 9.34, 1982, True]
After :  ['Virat', 9.34, True]
```

If you want to sort the items in a tuple, you can do it by using sorted() function. You can sort in increasing order or descending order.to sort in decreasing order, you need to add (reverse=**true**) in the code as well

```
list5 = [1, 99, 40]
print(list5)

list5.sort(reverse=True)
print(list5)

list5 = [1, 99, 40]
list5.sort(reverse=False)
print(list5)

list5 = [1, 99, 40]
list5.sort()
print(list5)
#Ascending by default
```

```
[1, 99, 40]
[99, 40, 1]
[1, 40, 99]
[1, 40, 99]
```

Split () function is used to split the items in a list. By giving index, you can get the Splitted item.

```
'This string needs to be splitted'.split()
```

```
['This', 'string', 'needs', 'to', 'be', 'splitted']
```

```
splitted_str = 'This string needs to be splitted'.split()
print(splitted_str)

print(splitted_str[0])
print(splitted_str[1])
print(splitted_str[2])
print(splitted_str[3])
print(splitted_str[4])
print(splitted_str[5])

print(type(splitted_str))

print(len(splitted_str))
```

```
['This', 'string', 'needs', 'to', 'be', 'splitted']
This
string
needs
to
be
splitted
<class 'list'>
6
```

If you want to reverse the arrangement of items in a list, use **.reverse()** function.

```
list7 = [1, 3, 5, 7 , -1 , 2]
print(list7)

list7.reverse()
print(list7)
```

```
[1, 3, 5, 7, -1, 2]
[2, -1, 7, 5, 3, 1]
```

You can convert a list into tuple and vice versa

```
my_tuple = (1, 3, 5, 7 , -1 , 2)
print(my_tuple)

few_elements = my_tuple[:2]
print(few_elements)

list_from_tupple = list(few_elements)
print(list_from_tupple) #converting tuple to list

tuple_from_list = tuple(list_from_tupple)
print(tuple_from_list) #converting list to tuple
```

```
(1, 3, 5, 7, -1, 2)
(1, 3)
[1, 3]
(1, 3)
```

Dictionaries:

Dictionary in Python is an unordered collection of data values, dictionaries are written with curly brackets, and they have keys and values. Each key-value pair in a Dictionary is separated by a colon (**:**), whereas each key is separated by a 'comma'.

How to create a dictionary?

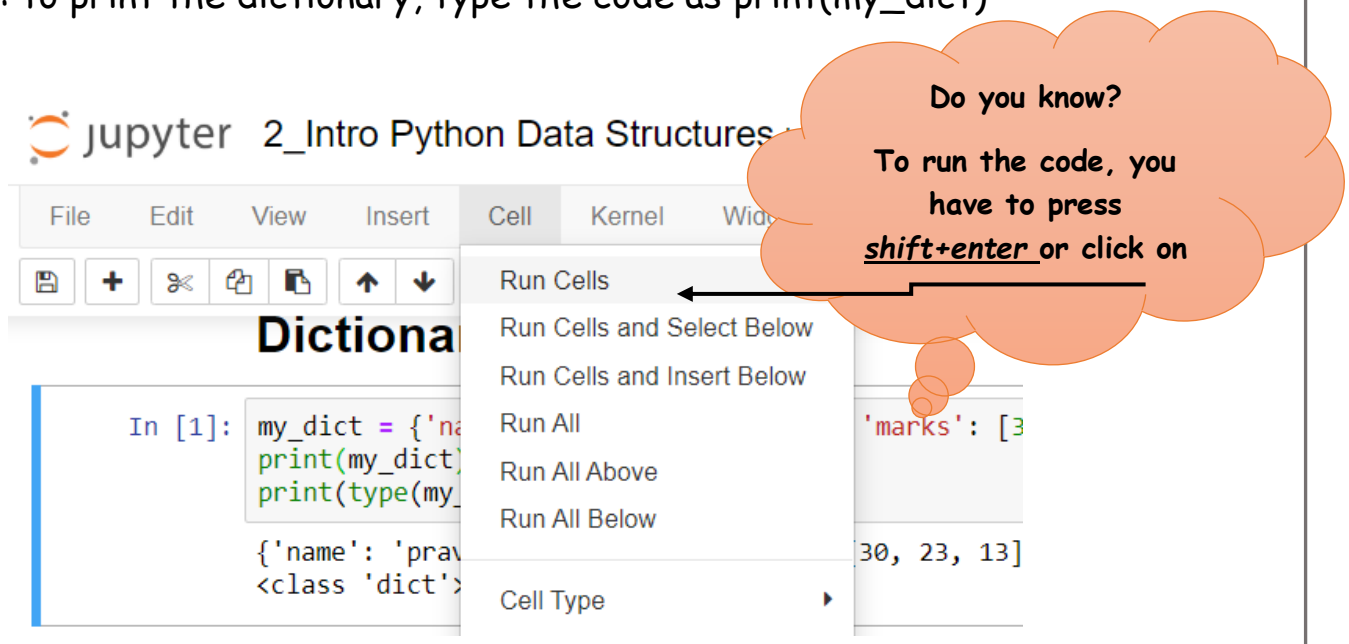
In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by 'comma'. The Key : value format has to be used .

```
In [1]: my_dict = {'name': 'pravin', 'age': 22, 'marks': [30, 23, 13], 'assign_grades': ('A', 'B', 'A')}
        print(my_dict)
        print(type(my_dict))#Key value format

        {'name': 'pravin', 'age': 22, 'marks': [30, 23, 13], 'assign_grades': ('A', 'B', 'A')}
        <class 'dict'>
```

In the above picture, you can see that how you can create a dictionary.
e.g., my_dict = {'name': 'pravin', 'age': 22, 'marks': [30, 23, 13], 'assign_grades': ('A', 'B', 'A')}
so here in this example, name, age marks , assign_grades are the keys and pravin , 22, [30,23,13] , ('A', 'B', 'C') are the values.

Note: to print the dictionary, type the code as print(my_dict)



Now, if you want to access the values .how you can do that?
You can access the items of a dictionary by referring to its key name, inside square brackets

```
In [52]: print(my_dict["name"])
         print(my_dict["age"])
         print(my_dict["marks"])                    #key is used to get the value
         print(my_dict["assign_grades"])

         pravin
         22
         [30, 23, 13]
         ('A', 'B', 'A')
```

The key names are called to get the values. You can also acces the values by using the **"get function"**

```
In [3]: my_age = my_dict.get("age")
        my_age #Using get function to get the age

Out[3]: 22
```

How to alter an element in a dictionary?

In the above example the name is pravin , and if we want to change the name to pawar , then

```
In [55]: my_dict["name"] = "pawar"
         my_dict #Altering dictionary element

Out[55]: {'name': 'pawar',
          'age': 22,
          'marks': [30, 23, 13],
          'assign_grades': ('A', 'B', 'A')}
```

The same way you can change the age, marks and assign_grades Like, my_dict["age"] = 24 then call the name of your dictionary to get the altered dictionary.
You get to know that you can alter the dictionary, now you will be wondering that can we add a new element in an existing dictionary? So, the answer to this question is **yes**. Let's see how

Adding a new element:

```
In [4]: my_dict["new_key"] = "new_data"
        my_dict #Adding new eliments to dictionary

Out[4]: {'name': 'pravin',
         'age': 22,
         'marks': [30, 23, 13],
         'assign_grades': ('A', 'B', 'A'),
         'new_key': 'new_data'}
```

```
In [5]: key_list = my_dict.keys()
        print(key_list)

        dict_keys(['name', 'age', 'marks', 'assign_grades', 'new_key'])

In [6]: dict_values = my_dict.values()
        dict_values

Out[6]: dict_values(['pravin', 22, [30, 23, 13], ('A', 'B', 'A'), 'new_data'])

In [7]: dict_items = my_dict.items()
        dict_items

Out[7]: dict_items([('name', 'pravin'), ('age', 22), ('marks', [30, 23, 13]), ('assign_grades', ('A', 'B', 'A')), ('new_key', 'new_dat
        a')])
```

In the above picture, you can see that how you can get **only the keys**, **only the values**, and **both (together called items)**

For deleting any element from a dictionary, simply use the code :

```
In [8]: print(my_dict)
        del(my_dict['name'])
        print(my_dict)

{'name': 'pravin', 'age': 22, 'marks': [30, 23, 13], 'assign_grades': ('A', 'B', 'A'), 'new_key': 'new_data'}
{'age': 22, 'marks': [30, 23, 13], 'assign_grades': ('A', 'B', 'A'), 'new_key': 'new_data'}
```

'name' key is deleted here by using the ***del function.***

Note: as you can see that in the code only the key 'name' is written not the value 'pawar', so, by this you can understand that if we delete the key, the value of that respective key will be automatically deleted.

```
In [9]: 'Name' in my_dict
Out[9]: False

In [10]: 'age' in my_dict
Out[10]: True
```

So, by the above written codes, you can understand that **'keys are case sensitive'**. To determine if a specified key is present in a dictionary use the 'in 'keyword .As in the 'name' key the n is in lower case, but in the above written code it is in the upper case that is why when you run this code , it is showing false means the 'Name' key is not in my_dict,but 'age' key is there in my_dict that is why when we run the code it is showing the output as true

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the len() method.

Code: print(len(my_dict))

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys

pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Sets

A Set is an unordered collection data type. In Python sets are written with curly brackets. Unlike dictionaries, sets do not have key:value format.

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

How to create a set?

Sets

In [19]:
```
city_set = {"Pune", "Mumbai", "Pune", "Mumbai", "Bangalore", "Delhi", "pune"}
print(city_set)
print(type(city_set)) #no key value #Sets are unordered collection of objects
```

```
{'Mumbai', 'Delhi', 'Pune', 'Bangalore', 'pune'}
<class 'set'>
```

as you can see that there are two Mumbai in the code but in the output, there is only one Mumbai. So, the set only take similar elements one time. But the pune is written two times, because in one pune p is lower case and in the other it is lower case. So, the sets are case sensitive.

If you are wondering about that Hashtag (#)

Hashtag sign is used when you want to add a comment,

I do it all the time, you can write some important points so that you don't forget

De

Python's built-in set type has the following characteris

- Sets are unordered.
- Set elements are unique. Duplicate elements are no allowed.
- A set itself may be modified, but the elements contained in the set must be of an immutable type

Can a list be converted into a set? Yes, it can.

By using the set function, a list can be converted into a set and the vice versa is also possible by using the list function

In [20]:
```
city_list1 = ["Pune", "Mumbai", "Pune", "Mumbai", "Bangalore", "Delhi"]
city_set = set(city_list1)
city_set #list to set
```

Out[20]:
```
{'Bangalore', 'Delhi', 'Mumbai', 'Pune'}
```

Like, LIST= [2,4,6,8]

SET = set(LIST)

Like, SET= {1,3,5,7}

LIST= list(SET)

Note: To determine if a specified element is present in a set, use the 'in 'keyword.

```
          country_set
Out[8]:  {'India', 'US'}

In [39]: "India"  in country_set
Out[39]: True

In [40]: "UK"  in country_set
Out[40]: False
```

If a specified element is present, it will give the output as true otherwise false

How to add the element to a set?

```
In [7]: country_set = {"India", "US", "US", "India"}
        country_set.add("uk")
        country_set

Out[7]: {'India', 'US', 'uk'}
```

- By using the **add function**

How to remove the element from the set?

```
In [8]: country_set
        country_set.remove("uk")
        country_set

Out[8]: {'India', 'US'}
```

- By using the **remove function**

Get the Length of a Set

To determine how many items a set has, use the len() method.

Intersection of sets:

To get the common elements between two sets, either by using **&** or **intersection function.**

Set1= {1,2,3,4} set2= {3,4,5,6}

Code:

Set=Set1&set2 or set= set1.intersection(set2)

Output will be {3,4}

Union of sets:

To get all the elements between two sets, either by using **|** or **union function.**

Set1= {1,2,3,4} set2= {3,4,5,6}

Code:

Set=Set1|set2 or set= set1.union(set2)

Output will be {1,2,3,4,5,6} – no repetitive elements

Difference between two sets:

To get the uncommon values only by using the difference function

Set1= {1,2,3,4} set2= {3,4,5,6}

Code:

```
set= set1.difference(set2)
```

Output will be {1,2,5,6}

Symmetric Difference between two sets:

symmetric difference is used to get the elements which are not included in the intersection.

Set1= {1,2,3,4} set2= {3,4,5,6}

Code: set= set1.symmetricdifference(set2)

Output will be {1,2,5,6}

Frozen Sets:

Python provides another built-in type called a **frozen set**, which is in all respects exactly like a set, except that a frozen set is immutable

Note: all the data types (float, integer ,string)can be used to create a set.

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not

issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others