3] Dictionary data-type:

Dictionary is collection or sequence of element in the form of 'key : value' pair, key is attached by it's value through indentation.

Element of dictionary are enclosed within curly braces "{ }" and separated by commas.

```
dictionary_name = { key1 : value1, key2 : value2,
                    key3 : value3, ...., keyn : valuen }
```

Example:

```
# dictionary of square/cube of resp numbers
dict_sq = { 1:1, 2:4, 3:9, 4:16, 5:25 }
dict_cb = { 1:1, 2:8, 3:27, 4:64, 5:125 }
dict = { 1:'odd', 2:'even', 3:'odd', 4:'even' }
dictionary = { 1:'one', 2:'two', 3:'three' }
dictionary2 = { 1:'gold', 2:'silver', 3:'bronze' }
```

It is unordered data-type and also it is partially mutable i.e. key value is unique and cannot be change where as value can be change.

There are various operations thant can be perfor-med on Dictionary.

1) Creating or Declaring Dictionary:

In Python we can Create or Declare a Dictionary by using following Syntax. i.e. by assing sequence of elem. in form of key : value pair enclosed in curly braces. to valid variable.

```
dictionary_name = { key1 : val1, key2 : val2, ..., keyn : valn }
```

where, dictionary-name = valid variable(a-z, A-Z, 0-9, _)
key = all data-type (except list, set, dict), variable, input, expression.
value = all data-type, variable, expression, input.

Example;
#1. dict. of alph. with corresponding word.
dict_1 = {'a': 'apple', 'b': 'ball', 'c': 'cat'}
#2. dict. of product & their price
dict_2 = {'potato': 30, 'onion': 35, 'tomato': 30}
#3. dict. of bilionaire and their billions
dict_3 = {'Lucky': '99 B', 'Jeff': '187 B', 'Mark': '153 B'}

2) Accessing the element of Dictionary:
As dictionary is collection of key: value pair we can access the element of dictionary. As other data-types uses index to access the element, dictionary uses 'key' values to access the element.

a) dict[key]: by using this method we can access the value present at this key.

dict_name[key]['key']
Example:
```
dict = {1: 'one', 'two': 2, '#': 'char', 3.1: 1.3}
print(dict[1]) ; print(dict['#'])
print(dict['two'])

dict_2 = {1: 1, 2: 'ninja', 'code': 3}
print(dict_2[1]) ; print(dict_2[2])
print(dict_2['code'])
```

b) get(): by using this method we can access the value at particular key by passing that key. It return 'None' value when key is not present.

dict_name('key'), default)

Example:

```
dict={'13':'iphone', 22:'samsung',
      9:'oneplus', 0:'infinix'}
print(get('13')) ; print(get(0))
print(get(9))    ; print(get(22))
```

3) Traversing dictionary:

Traversing dictionary is way of printing all items one by one.
As other collection datatype use indexing for traversing, dictionary use different method for traversing.

a) Keys(): by using key method we can only iterate the keys of dictionary

dict_name.keys()

Example.

```
dict1={'A':'a', 'B':'b', 'C':'c', 'D':'d'}
dict2={1:0b101, 2:0b010, 3:0b011}
print(dict1.keys())
print(dict2.keys())
```

b) values(): it return the iteration of values of dictionary.

:: dict.values() ::

Example: dict3 = {bool(1): True, bool('h'): True}
print(dict1.values())
print(dict2.values())
print(dict3.values())

c) items(): the most elegent method to iterate over dictionaries is item method. it return the list of tuple of individual elem. of dictionary.

```
for var in dict.items():
    print(var)
```

Example:

```
for var in dict1.items():
    print(var)
for var1 in dict2.items():
    print(var1)
print(dict3.items())
```

4) Updating elements of Dictionary:
Since dictionaries are partially mutable we can update or modify the elem. of dict. As key are unique and immutable we can only update the values associated with key or can add new key:value pair.

a) dict[key]: by using this method we can update a value of particular key or add a new key:value pair.
If key is not present then it will consideres as new key:value pair with associated value.

dict[key] = value.

e.g. dict = {1:'one', 2:'two', 3:'three', 4:'four'}
dict[1] = 'one-1' ; dict[2] = 'two-2'
dict[5] = 'five' ; dict['absent'] = 'new-ele

b) update(): update method is used to add new key:value pair or update the value of existing key.

dict.update(parameter)

Example:

```
dict={'A':'a','B':'b','C':'c','D':'d','E':'f'}
dict.update('E':e), dict.update('F':'f')
dict.update('G'='g')
dict_ = {'Upper case':'Lower case'}
print(dict.update(dict_))
```

- Updating Keys:

As keys of dictionary unique and immutable, it cannot be change/modified directly. But indirectly, using different methods or function it can be changed.

Steps:

1) Assign the value of key to be deleted to new variable (to store the value of key)

2) Delete the key using del() or pop() function.

3) Assing the value of previous key to the new key.

4) Display the dictionary.

- Definition
- Syntax
- Parameter
- Return value
- Example

```
# create dictionary
dict = {'1st':'gold', '2nd':'silver', '3rd':'bronze'}
# assinging value to a variable
a = dict['2nd'] / dict.get('2nd')
# delete the key
del dict['2nd'] / dict.pop('2nd')
# assigning value to new key.
dict['2nd'] = a
# display the dictionary
print(dict)
```

5] Deleting Element's of Dictionary:

Delction i.e. removing an element from dictionary is carried out but different methods.

1) .pop() : this method is used to remove key:value pair of specified key. It return the value of 'key' specified.

Syntax: dict-name.pop('key', default)

it accept two parameters, 'key' which is to be removed and default value i.e. if key is not found it return default value.

Example:

```
dict = {10:'SSC', 12:'HSC', '12+':'Degree',
        '20+':'Phd-Dr.'}
dict.pop('20+')
dict.pop('12+')
dict.pop('9+')
dict.pop('8+', 'Not Found')
print(dict).
```

2) .del(): this method is used to delete key: value pair of specified key.

Syntax: del dict-name[key]

it accept only one argument 'key', it does not return any value. If key is not present then it raises key-error.

Example: dict={2:4, 3:9, 4:16, 5:25, 6:36}
del dict[2] ; del dict[3]
del dict[4] ; del dict[5]
print(dict) → {6:36}

3) popitem(): this method remove the last element of dictionary. and return it in pair form.

Syntax: dict_name.popitem()

Example: dict={'name':'Lucky', 'Class':'SE', 'Div':'B', 'Roll no.': 2263}
dict.popitem ; dict.popitem()
print(dict.popitem())
print(dict)
→ {'name':'Lucky'}

popitem() method does not accept any parameter.
It return the ~~to~~ key: value pair of last element of dictionary.

6] Different Method's and Built-in Functions:

There are various Built in methods or function available, that can be used to perform different operation on dictionary.

1) len(): this method return the length i.e. total no. of element of dictionary

Syntax: len(dict_name){

Example:
```
dict = {1:1.1, 2:2.2, 3:3.3}
print (len (dict))
dictL = {'a':'a', 'b':'b'}
print (len (dictL))
```

2) max(): this method return the max key of dictionary, key should be campairable.

Syntax: max (dict_name)

Example:
```
dict = {1:1.1, 2:2.2, 3:3.3}
print (max (dict))
print (max (dictL)
```

3) min(): this method return the minimum i.e. least element of dictionary.

Syntax: min (dict_name)

Example:
```
dict = {-1:1, -2:2, -3:33
print (min (dict))
dict2 = {0.0:1, 0.001:1, 0.0001:22}
print (min (dict2))
print (min (dictL))
```

4) sum(): it return the sum of keys of dictionary, keys should be of same data-type.

Syntax: sum(dict_name)

Example:
```
dict = {1:'one', 2:'two', 3:'three'}
print(sum(dict))
print(sum(dict2))
print(sum({1:1, 1:1:1, 1.111:1, 1.1111:1}
```

5) Sorted(): it sort the keys of dictionary i.e. arrange the key in increasing order and return the list of keys.

Syntax: sorted(dict_name)

Example:
```
dict = {1:1.1, 1.1:1.2, 1.11:1.3, 1.01:2}
print(sorted(dict))
print(sorted(dict1))
```

6) all(): it return true value when all the keys of dictionary are true.

Syntax: all(dict_name)

Example:
```
dict = {1:1, 2:2, 3:3, 4:5}
print(all(dict)) ; print(all(dict2))
Print(all(dict1)
```

7) any(): it return true value when any of the dictionary key is true.

Syntax: any(dict_name)

```
dict = {False:True, False:None, bool(0.0):1}
print(any(dict)) ; print(any(dict1))
print(any(dict2))
```

8) clear(): this method is used to clear or empty the entire dictionary, it doesn't accept any parameter or return None value.

Syntax: clear() → dict_name.clear()

Example:
```
dict = { 1:'gold', 2:'silver', 3:'bronze'}
dict.clear() ; dict2.clear()
print(dict) → print(dict2)
```

«#» Following code for clear dictionary

```
dictionary = {'key':'value'}
dictionary.clear()
print(dictionary)
```

-:<<#>>:- Following are the important methods used in dictionary:

1) copy(): The copy() method creates a copy of dictionary. It does not take any parameter it returns a shallow copy of dictionary

Syntax: dict_name.copy()

```
dict = {'key':'value'}
dict1 = {'key1':'value1'}
print(dict.copy); a= dict1.copy()
print(a)
```

2) fromkeys(): The fromkeys() method creates a dictionary from given sequence of keys and values.

Syntax: dict.fromkeys(key_sequence, value_sequence)

Example:

```
keys = [1,2,3,4]
value = (value)
print(dict.fromkeys(keys,value))
print(dict.fromkeys((2,4,6),'even')
print(dict.fromkeys((1,5,3),'odd')
```

3) get(): it return value of specified key
for reference see. #2) Accessing element of dict.
b) get(key) method.

Syntax: get(key, default)

Example:

```
dict = {'key','value'}
dict.get('Key')
dict.get('Key2')
dict.get('key3','not found')
```

4) keys(): This method return the sequence (list) of keys of dictionary.

Syntax: dict_name.keys()
Example:
```
dict = {'key': 'value'}
dict1 = {2:4, 3:6, 4:8}
print(dict.keys())
print(dict1.keys())
```

5) values(): This method return the value object which return view object which display the list of values of dictionary.

Syntax: dict_name.values()
Example:
```
dict = {'keys': 'values'}
print(dict.values())
print(dict1.values())
```

6) items(): This method return the value object which return view object which display the list of element of dictionary

Syntax: dict_name.items()
Example:
```
dict = {'a': 'apple', 'b': 'ball'}
print(dict.items())
print(dict1.items())
```

7) pop(): this method remove and return the key: values of specified key.

Syntax: dict_name.pop(key, default)
Example:
```
dict.pop('key')
dict1.pop('key', 'not found)
print(dict1.pop(3))
```

8) popitems(): it remove the last element of the dictionary, it return the last element

Syntax: dict-name.popitem()
Example:
```
dict = {2:'two', 3:'three', 4:'four'}
dict.popitem(); dict.popitem()
print(dict.popitem())
```

9) update(): this method update the keys of dictionary or add new key:value pair.
It also add new diction to previous dictionary.

Syntax: dict-name.update([parameter])
Example:
```
dict1 = {1:'one', 2:'two', 3:'three'}
dict2 = {4:'four', 5:'five'}
dict1.update(dict2)
dict1.update({'key':'value'})
```

10) setdefault(): it return the value of key, if key is not present it add new key with specified value.

Syntax: dict-name.setdefault(key, value)
Example:
```
dict = {1:'one', 2:'two'}
dict.setdefault(2)
dict.setdefault(1)
dict.setdefault(3, 'three')
dict.setdefault(4) (=> None)
dict.setdefault(3)
print(dict)
```

11) dict(): dict() method is used to create dictionaries

Syntax: dict(iterable)

Example:
```
dict1={1:2, 2:4, 3:9}
print(dict({1:11, 2:22}))   mapping
print(dict(x=5, y=6, z=7))   key argument
print(dict([(x,6),(y,6),(z,6)]))   iterables.
```

▸ Dictionary Comprehension:

Dictionary Comprehension is an elegent and consise way of creating Dictionary.
It is based on methodology, that writting a code inside curly bracket, it will return in dictionary form.

Syntax: { expression for var in iterable }
expression should be in the form of key : value.

Example: dictionary Comprehension.

```
print({a:a for a in range(1,11)})
print({a:a*2 for a in range(1,11)})
print({a: a+1 if a%2==0 for a in
        range(1,11)})
dict={1:'one', 2:'two'}
print({key:value+'Y' for (key, value) in
        dict.item()})

print({key: value & 'even' if key%2==0
        else 'odd' for key in range(1,11)})
```

4] Set data-type:

Set is one another collection data-type. It is collection of unique (only one) element enclosed in curly broces '{ }', elements are separated by ','

Set = { val1, val2, val3, ..., valn }

Examples: set_num = {1, 2, 3, 4, 5}
set_str = {'one', 'two', 'three'}
set_flo = {3.14, 9.8, 2.4, 3.77}

Item / element of set are not ordered.
Set have unique value i.e. they eleminate duplicate value, although the set contain an element / item more than once it considers first occurance only.

1) Creating and Declaring Set:

In python we can create and declare a Set by using following Syntax.

Syntax: set_name = {val1, val2, val3, ..., valn}

set_name can be a valid variable or identifier.
value = all datatype except(list, dict, set)

Example: set = {1, 2, 3, 4, 5, 6}
set_str = {'Sunday', 'Monday', 'Tuesday'}
set_rc = {'violet', 'indigo', 'blue', 'green', 'orange', 'red', 'Yellow'}
st_rep = {3.14, 9.8, 7+4j, 3.14, 8, 8, 8, 3.14, 9.8}

2) Accessing Element of Set:

Sets are mutable, since they are unordered indexing has no meaning. Hence we cannot access the element of Set using indexing.

In set we can't access any random element, but by traversing a set all element can be accessed.

3) Traversing a Set:

Traversing is a technique of accessing the elements one by one.

Set can be traversed by using two technique.

a) Using For loop:

Using loop a set can be traversed usually for loop.

Syntax:

```
for var in set-name:
    print(var)
```

Example:

```
set1 = {1, 2, 3, 4, 5}
for ele in set1:
    print(ele)
```

b) Using Enumerate:

Using enumerate function we can traverse the loop set elements along with their index

Syntax: for elem in enumerate(set-name):

```
set = {'a', 'e', 'i', 'o', 'u'}
for vowels in enumerate(set):
    print(vowels)
```

3] Updating element of Set:
Since, sets are unordered, set cannot be updated using indexing. There are another method by which set can be updated.

a) add(): this method add single element to set at random place.

Syntax: set_name.add(element)
Example:
```
set = {2,4,6,8,10}
set.add(10) ; set.add(12)
set.add(13) ; set.add(None)
print(set)
```

b) update(): this method is used to add a sequence of element to set.

Syntax: set_name.update(sequence)
Example:
```
set = {1,2,3}
set.update([4,5,5,6])
set.update([7,8],{7,8,9})
set.update('Hii'); set.update(77,82)
print(set)
```

4) Deleting element's of Set: As set are mutable deleting / removing of element is possible in set. There are diff. methods -

a) discard(): this method remove / delete the specified element of set. It does not change the set if element is absent.

Syntax: set_name.discard(element)

Example: set1={22, 5, 55, 1.23, 4}
set1.discard(22) ; set1.discard(4)
set1.discard(None); set1.discard(0)
print(set1)

b) remove(): this method is also used to remove specified element it will raise key error if elem. not present.

Syntax: set_name.remove(element)
Example set2={5, 55, 0.5, 5.5, 555}
set2.remove(55) ; set2.remove(0.5)
set2.remove(5.5) → # error
print(set2)

c) pop(): this method used to remove and return last element, but set is unordered any element will be removed arbitarily.

Syntax: set_name.pop()
Example: set={1, 1.1, 0.1, 1.11, 11, 11.1}
set.pop() ; set.pop(), set.pop()
print(set)

- Set does not support '+' and '*' operator hence, operations like concatenation and repeatation are not applicable to set.

6). Different Method's and Built-in function on Set:
There are various In-built methods available to perform operation on Set.

1). Set union(): union() method return the set containing all element of both set.
it can also be used one by '|' operator.

Syntax: set1.union(set_name 2)
Example: set_1 = {2, 4, 6, 8}
set_2 = {2.2, 4.4, 6.6, 8.8}
print(set_1.union(set_2))
print(set_1 | set_2)

2) intersection(): intersection() method return the set of containing common elements from bothe the set.
it can also be acieved by '&' operator.

Syntax: set_name1.intersection(set_name2)
set_name1 & set_name2
Example: set1 = {1, 2, 3} set2 = {4, 5, 6}
print(set1.insersection(set2))
print(set1 & set2)

3) difference(): difference() method return the difference of set two set, i.e. collection of elem. of set that are not in other set.
it can also be done using '-' operator

Syntax: set_name1.difference(set_name2)
set_name1 - set_nam2

Example: set1 = {2, 3, 4} and set2 = {4, 5, 6}
print(set1.difference(set2))
print(set1 - set2)

4) Symmetric-Difference: this method return the set of elements of 1st and 2nd set except the elements which are common to both the sets.

Syntax: set1.symmetric-difference(set2)
set1 ^ set2
Example: set1 = {2, 4, 6, 8}
set2 = {6, 8, 10, 12}
print(set1.symmetric-difference(set2))
print(set1 ^ set2)

5) difference_update(): this method performs difference operation between two set and modifies the first set as a result of difference and then the resultant is updated to first set.

Syntax: set_name1.difference_update(set_name2)
Example: set1 = {1, 2, 3, 4}; set2 = {3, 4, 5, 6}
print(set1.difference_update(set2))

6) intersection_update(): this method performs intersection of two set and assign the result to first set and then update the result to first set.

Syntax: set1.intersection_update(set2, ...)

Example: set1 = {1,2,3,4} , set2 = {2,3,4,5}
set1.intersection_update(set2)
print(set1)

7) symmetric_difference_update():
this method perform the symmetric_difference operation and assign result to first a set and then update above result to the 1st set.

Syntax: set1.symmetric_difference_update(set2)
Example: set1 = {1,2,3,4} ; set2 = {3,4,5,6}
set1.symmetric_difference_update(set2)
print(set1)

8) isdisjoint(): this method return true when both set are different else return False

Syntax: set1.disjoint(set2)
Example: set1 = {1,2,3} , set2 = {4,5,6}
set3 = {3,4,5}
print(set1.isdisjoint(set2))
print(set1.isdisjoint(set3))

9) issubset(): issubset() method return true value if second set is subset of first. else it return false.

Syntax: set1.issubset(set2)
Example: set1 = {1,2,3,4} ; set2 = {3,4}
print(set2.issubset(set1))
print(set1.issubset(set2))

10) issuperset(): this method return True value if first set is superset of second else it return False value.

Syntax: set1.issuperset(set2)

Example:
```
set1 = {1,2,3,4,5} ; set2 = {2,3,4}
print (set1.issuperset(set2))
print (set2.issuperset(set1))
```

- There are other methods also available, that performs different operation on set, the are discussed in previous topics:

- clear(): it clear the set and return the empty set.
- copy(): it generates shallow copy of set.
- set(): it convert other data/iterable to set.
- all(): it return True if all element are true
- any(): it return True if any element of set is True.
- max(): it return largest element of set.
- min(): it return least element of set.
- len(): it return the length of set.
- sum(): it return the sum of element of set.
- sorted(): it return sorted set.

- add(): add single element at random place in set.
- update(): add multiple element to set
- discard(): it remove specified element from set.
- remove(): it remove specified element from set.
- pop(): it remove any arbitary element.

- Set Comprehensions:

Set Comprehension is an elegent and consize way of creating set from existing set of to create new set.

It is based on methodology of writing the code inside curly bracket and the output is stored inside the set, and it will return in the form of set.

Set Comprehension is similar to List comprehension, but set will discard the duplicate elements.

Syntax: {Expression for item in range()}

Example:

#1). a={ item for item in range(10)}
print(item)

#2). print {n for n in [1,2,3]})

#3). print({a if a%2==0 for a in range(1,11)})

#4). b = {a if a%2 != 0 for a in range(1,11)}

print(b)

#5). c = {a for a in range(1,51) if a%2==0
if a%5==0})

print c.

#6). d = {a if a%2 != 0 for a in (1,2,3,4,5)}
print(a)

#7). print {ele 'even' if ele%2==0 else 'odd'
for ele in range(1,11)}