DICTIONARY-DATA Type :

- A Python dictionary is a data structure that allows us to easily write very efficient code
- Python dictionaries allow us to associate a value to a unique key, and then to quickly access this value.
- Dictionaries contains set of key/value pairs which are enclosed between curly braces separated by comma. Here keys are unique.
- A pair of curly braces creates an empty dictionary: ------->>> { }
- The main operations on a dictionary are storing a value with some key and extracting the value with given key.
- Dictionary keys are not allowed duplicates but dictionary values are allowed duplicates.
- We can use homogeneous and heterogeneous elements for both keys and values.
- Insertion order is not preserved.
- Dictionary keys are only immutable and values are **mutable** | **immutable**.
- Dictionary will not allow indexing and slicing.
- Dictionaries are indexed by keys, which can be any immutable types like strings and numbers can always be keys But Tuples can be used as keys if they contain only strings, numbers, or tuples.
- NOTE1: If a tuple contains any mutable objects like list, set, dict either directly or indirectly, it cannot be used as a key.
- NOTE2: You can't use lists as keys, since lists can be modified using index assignments, slice assignments, or methods like append() and extend().
- NOTE3: Dictionary Keys are numbers , strings and tuples only but not list, set and dictionaryes.

We can create dictionary in different ways, ----->> using {} , dict()

1. Creating empty dictionary and adding key : value pairs.

Example:

```
>>> dic1 = {}                  # Creating empty dict
>>> print(dic1)                {}
>>> type(dic1)                 <class 'dict'>
>>> dic1['a']=10               # adding Key:Value pair to dictionary
>>> dic1['b']=20               # adding Key:Value pair to dictionary
```

```
>>> dic1['c']=30              # adding Key:Value pair to dictionary
>>> dic1['a']=10              # trying to add same Key:value pair
>>> dic1['a']=50              #adding new value for same key 'a'
>>> print(dic1)                    {'a': 50, 'b': 20, 'c': 30}
```

2. Creating a dictionary with dict() function:

Example:

```
>>> dic1=dict( [ ( 'a ', 10 ) , ( 'b' , 20 ) , ( 'c' , 30 ) , ( 'd' , 40 ) ] )
>>> print(dic1)        # {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> type(dic1)         # <class 'dict'>
```

3. Creating a dictionary with "curly braces" including "key:value" pairs.

Example:

```
>>> dic1={'a':10,'b':20,'c':30,'d':40}
>>> print(dic1)                    # {'a': 10, 'b': 20, 'c': 30, 'd': 40}
>>> type(dic1)                    # <class 'dict'>
```

Accessing data from dictionary:

- We can assign values to the keys and later we can fetch values by using Keys.

Example:

```
>>> stuDetails={'Id':100,'Name':'Sai','Age':20,'Marks':90}
>>> print(stuDetails)          # {'Id': 100, 'Name': 'Sai', 'Age': 20, 'Marks': 90}
>>> type(stuDetails)           # <class 'dict'>
>>> print(stuDetails['Id'])                100
>>> print(stuDetails['Age'])          20
```

Example:

```
>>> stuDetails={'Id':100,  'Name':'Sai','subjects':['SQL Server', 'Oracle', 'Python']}
>>> stuDetails
{'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python']}
>>> stuDetails['Id']                    100
>>> stuDetails['Name']                  'Sai'
>>> stuDetails['subjects']               ['SQL Server', 'Oracle', 'Python']
>>> stuDetails['subjects'][0]           'SQL Server'
>>> stuDetails['subjects'][1]           'Oracle'
>>> stuDetails['age']                  # KeyError: 'age'     # this key not available
```

has_key():

- To prevent this type of error we can check whether the specified key existed or not by using "has_key()".
- But this "has_key()" is available in python2 only not in python3 version.

```
>>> stuDetails.has_key('age')          # False          #in python2 version
```

- In python3 we have to check by using membership operator like 'in'.

```
>>> 'age'  in  stuDetails              # False      #in python3 version
```

Adding new key : value pairs:

- If we need to add another element then use like below,

```
>>> stuDetails['Age']=25      #adding new element to the stuDetails Dictionary
>>>print(stuDetails)
{'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python'], 'Age': 25}
```

Updating dictionary values:

- If we need to change the age from 25 to 27 then

```
>>> stuDetails['Age']=27          #changing the age from 25 to 27
>>>print(stuDetails)
{'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python'], 'Age': 27}
```

Dictionary Methods:

keys(): It returns all list of keys from dictionary object.

Example:

```
>>> dic1 = {1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>> dict1.keys()
     Dict_keys([1,2,3,4, 's']
```

values(): it returns list of all values from the dict.

Example:

```
>>> dic1={1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>> dic1.values()
dict_values(['Python', (3+5j), (10, 20, 30), [100, 'a', False], 100])
```

items():

- This function is used to get all items. All key and value pairs will be displayed in tuple format.

Example:

```
>>> stuDetails= {'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle',
'Python'], 'id': 1000,'name': 'nani'}
>>> stuDetails.items()
dict_items([('Id', 100), ('Name', 'Sai'), ('subjects', ['SQL Server', 'Oracle',
'Python']), ('id', 1000), ('name', 'nani')])
```

get(key):

- This function is used to get the value of specified key.

Example:

```
>>> stuDetails=
{'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python'], 'id': 1000,
'name': 'nani'}
>>> stuDetails.get('Id')                100
>>> stuDetails.get('subjects')              ['SQL Server', 'Oracle', 'Python']
```

copy(): it copies the one dictionary object values into new dictionary object.

Example:

```
>>> dic1  =  {1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>> dic2  =  dic1.copy()
>>> dic1
{1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>> dic2
{1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>> id(dict1)     #  70894768
>>> id(dic2)      #   61428432
```

Deleting key : value pairs from dictionary :

- If we need to delete the value 27 from the above dictionary then (by using pop).

```
>>> stuDetails.pop('Age')            27
>>> print(stuDetails)
{'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python']}
```

popitem():

- We can also delete by using popitem()

```
#  This function removes last key : value  pair in the dictionary,
>>> stuDetails.popitem()     ('subjects', ['SQL Server', 'Oracle', 'Python'])
```

- If we need to delete all key:value pairs then we can use clear()

```
>>> stuDetails.clear()          # deleting all pairs, then we can have an empty dict.
>>> print(stuDetails)           {}
```

pop(): it removes specific key : value pair and returns value as result.

Example:

```
>>> dic1={1: 'Python', 2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
>>>dic1.pop(1)          # 'Python'
>>> print(dic1)         # {2: (3+5j), 3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
```

clear():

- It removes all key : value pairs from dictionary object and maintains empty dictionary object.

Example:

```
>>> dic1.clear()
>>> dic1          #  {}
```

del command:

- We can also remove the key: value pair by using 'del' command.

Example:

```
>>> del dic1[2]
>>> print(dic1)              {3: (10, 20, 30), 4: [100, 'a', False], 's': 100}
```

fromkeys():

- We can use tuple elements as keys in the dict and the elements must be unique

  Example:

```
>>> tup = (1,2,3,4,5)          # creating tuple
>>> dic1.fromkeys(tup)         # taking tuple elements as keys in the dict dic1
{1: None, 2: None, 3: None, 4: None, 5: None}
```

- By default values are None, if we need to get 0 as default then,

```
>>> tup = (1,2,3,4,5)
>>> dic  =  {}.fromkeys(tup,0)
```

>>> dic {1: 0, 2: 0, 3: 0, 4: 0, 5: 0}

- **We can also use list elements as keys in the dict and the elements must be unique.**

  >>> lst = [1,2,3,4,5] #creating list

  >>> dic2.fromkeys(lst) # taking list elements as keys in the dict dic2

  {1: None, 2: None, 3: None, 4: None, 5: None}

update():

- The `update()` method updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.
- The current dictionary will be updated with the all key:value pairs from another dictionary.

  For example:

  >>> stuDetails={'Id':100,'Name':'Sai', 'subjects':['SQL Server', 'Oracle', 'Python']}

  >>> stuDetails1={'id':1000,'name':'nani'}

  >>> stuDetails.update(stuDetails1)

  >>>print(stuDetails)

  {'Id': 100, 'Name': 'Sai', 'subjects': ['SQL Server', 'Oracle', 'Python'], 'id': 1000, 'name': 'nani'}

Note: The `update()` method adds element(s) to the dictionary if the key is not in the dictionary. If the key is in the dictionary, it updates the key with the new value.

update() When Tuple is Passed:

```
dictionary = {'x': 2}

dictionary.update([('y', 3), ('z', 0)])

print(dictionary)
```

Output:

```
{'x': 2, 'y': 3, 'z': 0}
```

Here, we have passed a list of tuples `[('y', 3), ('z', 0)]` to the `update()` function. In this case, the first element of tuple is used as the key and the second element is used as the value.

Using For – Loop:

```
>>> d1 = {1:10, 2:20}
>>> d2 = {2:100, 3:30}
>>> for I in d2 :
        d1[ i ] = d2 [ i ]
>>> print( d1 )
```

Output:

{1: 10, 2: 100, 3: 30}

Merging two Dictionaries into a New Dictionary using ** symbol :

```
>>> d1 = {1:10, 2:20}

>>> d2 = {2:100, 3:30}

>>> d3 = {**d1, **d2}

>>> d3
{1: 10, 2: 100, 3: 30}
```

Explanation :

- This is generally considered a trick in Python where a single expression is used to merge two dictionaries and stored in a third dictionary.
- The single expression is **. This does not affect the other two dictionaries.
- ** implies that an argument is a dictionary. Using ** [double star] is a shortcut that allows you to pass multiple arguments to a function directly using a dictionary.
- For more information refer **kwargs in Python. Using this we first pass all the elements of the first dictionary into the third one and then pass the second dictionary into the third. This will replace the duplicate keys of the first dictionary.

Merging two Dictionaries into a New Dictionary using | symbol in Python 3.9

In the latest update of python now we can use "|" operator to merge two dictionaries. It is a very convenient method to merge dictionaries.

```
>>> d1 = {1:10,2:20}
>>> d2 = {2:100,3:30}
>>> d3 = d1 | d2
>>> d3
{1: 10, 2: 100, 3: 30}
```

Q) How to perform arithmetic operations on the values of a dictionary?

```
>>> d1={'sub1':80,'sub2':90,'sub3':70,'sub4':80}
>>> print(d1)
{'sub1': 80, 'sub2': 90, 'sub3': 70, 'sub4': 80}
>>> s = sum(d1.values())
>>> print(s)                    320
>>> mx = max(d1.values())
>>> print(mx)                   90
>>> mn = min(d1.values())
>>> print(mn)                   70
>>> cnt = len(d1.values())
>>> print(cnt)                  4
```

All data types are comes under either mutable nor immutable:

Class	Description	Mutable?	Immutable?
int	Integer value	No	Yes
float	Floating-point number	No	Yes
bool	Boolean value	No	Yes
complex	Complex value	No	Yes
str	Character string	No	Yes
tuple	Sequence of objects	No	Yes
list	Sequence of objects	Yes	No
set	Un-Ordered set of unique objects	Yes	No
Frozenset	Immutable form set	No	Yes
dictionary	Key : value pairs	Yes	No
dict_keys		No	Yes
dict_values		Yes	Yes

All Data types deviding like 2 types. They are,

1. Fundamental behavior data types

Type	mutable	immutable
int	No	Yes
float	No	Yes
bool	No	Yes
complex	No	Yes
str	No	Yes

2. Collection behavior data types

Type	mutable	immutable	duplicates	insertion-order	indexing
tuple	No	Yes	Yes	Yes	Yes
list	Yes	No	Yes	Yes	Yes
set	Yes	No	No	No	No
dict	Yes	No	-	No	No
dict-keys	No	Yes	No	No	Yes
dict-values	Yes	Yes	Yes	No	No
frozenset	No	Yes	No	No	No

All Datatypes Revision:

- Mutable objects
  list, set, dictionary
- Immutable objects
  int , float , bool , complex , string , tuple , frozenset
- Dictionary_keys
  Immutable types only
- Dictionary_values
  Mutable or Immutable types
- Fundamental objects
  int, float,bool, complex, str
- Collection objects
  list, tuple, set, dict

========== ========== ==========

Types	str	list	tuple	set	frozenset	dict	dcit_keys	dict_values
mutable	No	Yes	No	Yes	No	Yes	No	Yes
immutable	Yes	No	Yes	No	Yes	No	Yes	Yes
duplicate	Yes	Yes	Yes	No	No	--	No	Yes
insertion_order	Yes	Yes	Yes	No	No	Yes		
empty object	''	[ ]	( )	set()	frozenset()	{}		

	str	list	tuple	set	frozenset	dict
indexing	Yes	Yes	Yes	No	No	keys
slicing	Yes	Yes	Yes	No	No	No
packing	Yes	Yes	Yes	Yes	Yes	Yes
unpacking	Yes	Yes	Yes	Yes	Yes	Yes
concatination	Yes	Yes	Yes	No	No	No
multiplication	Yes	Yes	Yes	No	No	No
membership	Yes	Yes	Yes	Yes	Yes	Yes

	str	list	tuple	set	frozenset	dict
clear()	No	Yes	No	Yes	Yes	Yes
del indexing	No	Yes	No	Yes	No	keys
deep_copy =	Yes	Yes	Yes	Yes	Yes	Yes
shallow_copy copy()	No	Yes	No	Yes	**Yes**	Yes (Same_momory)

========== ==========

Identify the Difference between methods : Interview methods

========= ====== ========== **********

remove() vs pop() ---- list

remove() vs discard() --- set

append() vs extend()-----list

index(value, insex) vs insert(index, value) ------list

index() vs find()------string

reversed() vs reverse() ----list,

sorted() vs sort()------list

pop() vs popitem() --- dict
count() in string, list, tuple
deep copy vs   shalllow copy