DICTIONARY

- It is used to store elements in **key – value** pair.
- **Keys should be unique** but value can be anything.
- **Keys are immutable** in dictionary.
- If we try to duplicate a key then the latest value will be considered.
- Dictionary is a **mutable** data structure.
- It also allows heterogeneous elements.
- **Insertion order is preserved** from 3.7 version of python.
- It doesn't support indexing and slicing.
- The notation of dictionary is **{}** where key & value are separated by colon.

Syntax to create an empty dictionary:

```
dict_name = {}
 or
dict_name = dict()
```

Syntax to create dictionary:

```
d = {1:'Python' , 2:'Java' , 3:10 , 4:7.8}
```

where,
1,2,3,4 = keys and Python,Java,10,7.8 = values

Iterating over the dictionary:

Consider the foll' dictionary,

```
d = {1:'one',2:'two',3:'three'}
```

Iterating dictionary	**Iterating keys**	**Iterating values**	**Iterating keys & values**
for i in d: print(i)	for k in d.keys(): print(k)	for v in d.values(): print(i)	for k,v in d.items(): print(k,'→',v)
OUTPUT: 1 2 3	OUTPUT: 1 2 3	OUTPUT: one two three	OUTPUT: 1 → one 2 → two 3 → three

Accessing Elements from Dictionary:

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets [] or with the get() method.

If we use the square brackets [], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

e.g. d = {1:'one',2:'two',3:'three'}
print(d[1]) #one
print(d.get(1)) #one

print(d[4]) #KeyError
print(d.get(4)) #None

Accessing an element of a nested dictionary:

In order to access the value of any key in the nested dictionary, use indexing [ ] syntax.

e.g.
sampleDict = {"class":{"student":{"name":"Mike","marks":{"physics":70,

"history":80}}}}

To access the value of key 'history':

print(sampleDict['class']['student']['marks']['history'])

Changing and Adding Dictionary elements:

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.

If the key is already present, then the existing value gets updated. In case the key is not present, a new (**key: value**) pair is added to the dictionary.

e.g. d = {1:'one',2:'two',3:'three'}
d[1] = Python
print(d) #{1:'Python',2:'two',3:'three'}
d[4] = 'four'
print(d) #{1:'Python',2:'two',3:'three',4:'four'}

Dictionary methods:

Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.
fromkeys(seq[, v])	Returns a new dictionary with keys from seq and value equal to v (defaults to None).
get(key[,d])	Returns the value of the key. If the key does not exist, returns d (defaults to None).
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
pop(key[,d])	Removes the item with the key and returns its value or d if key is not found. If d is not provided and the key is not found, it raises KeyError.
popitem()	Removes and returns the last (**key, value**) pair. Raises KeyError if the dictionary is empty.
setdefault(key[,d])	Returns the corresponding value if the key is in the dictionary. If not, inserts the key with a value of d and returns d (defaults to None).
update([other])	Updates the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Returns a new object of the dictionary's values

Dictionary Built-in Functions:

Built-in functions like all(), any(), len(), sorted(), etc. are commonly used with dictionaries to perform different tasks.

Function	Description
all()	Return True if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
sorted()	Return a new sorted list of keys in the dictionary.

Nested Dictionary:

```
India = {'Maharashtra':['Pune','Mumbai'] , 'Gujarat' : ['Surat','Bhuj']}
USA = {'Texas' : ['Austin','Houston'] , 'Washington':['Olympia','Seattle']}
World = {'India' : India , 'USA' : USA}
for k,v in World.items():
        print(k)
        for state,cities in v.items():
                print(state)
                for  city in cities:
                        print(city)
```

OUTPUT:

```
India
Maharashtra
Pune
Mumbai
Gujarat
Surat
Bhuj
USA
Texas
Austin
Houston
Washington
Olympia
Seattle
```