Ques. What is Dictionaries?

- Dictionaries are written with curly **brackets{}**, and have keys and values.
- Dictionary items are **ordered**, **changeable**, and **does not allow duplicates**.
- Dictionaries are **changeable**, meaning that we can change, add or remove items after the dictionary has been created.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020,
  "electric": False,
  "colors": ["red", "white", "blue"]
}
print(thisdict)

Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 2020, 'electric':
False,'colors': ['red', 'white', 'blue']}
```

Ques. Dictionary Length?

- To determine how many items a dictionary has, use the **len()** function.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))    # Dictionary Length:- Output:- 3
```

The dict() Constructor

- It is also possible to use the **dict()** constructor to make a dictionary.

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
Output:- {'name': 'John', 'age': 36, 'country': 'Norway'}
```

Check if Key Exists

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

Output:- Yes, 'model' is one of the keys in the thisdict dictionary
```

Access Item of Dictionary

Ques. Access Item of Dictionary?

```
thisdict =  {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}

# Using key name:- We can access the items of a dictionary by referring to its
**key name**, inside square brackets.
x = thisdict["model"]
print(x)    # Output:- Mustang


# Using the **get()** Method
x = thisdict.get("model")
print(x)    # output:- Mustang


# Get **All Keys** of the Dictionary:- The **keys() method** will return a list of
all the keys in the dictionary.
x = thisdict.keys()
print(x)    # Output:- dict_keys(['brand', 'model', 'year'])


# Get **All Values** of the Dictionary and We can change the value of a specific
item by referring to its key name.
x = thisdict.values()
print(x)    # Output:- dict_values(['Ford', 'Mustang', 1964])


# Get **All Items** :- The items() method will return each item in a dictionary,
as tuples in a list.
x = thisdict.items()
print(x)    # Output:- dict_items([('brand', 'Ford'), ('model', 'Mustang'),
('year', 1964)])
```

Add Dictionary Items

Ques. Add Dictionary Items?

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

# Using key:- using key add the items
thisdict["color"] = "red"
print(thisdict)   # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'red'}


# Using Update() method:- Adding Items using **Update()** Method
thisdict.update({"color": "red"})
print(thisdict)   # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'red'}
```

Change Or Update Dictionary Items?

Change Or Update Dictionary Items

- Change Values:- We can change the value of a specific item by referring to its **key name**.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

# Using key name:-  We can change the value of a specific item by referring to its
key name.
thisdict["year"] = 2018
print(thisdict)   # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

# Using Update method:-
thisdict.update({"year": 2020})
print(thisdict)   # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Remove Dictionary Items?

Ques. Remove Dictionary Items?

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

# using pop() method:- The **pop()** method removes the item with the specified
**key name**:
thisdict.pop("model")
print(thisdict)  # Output:- {'brand': 'Ford', 'year': 1964}


# using **popitem()** methods:- method removes the last inserted item (in versions
before 3.7, a random item is removed instead).
thisdict.popitem()
print(thisdict)   # Output:-  {'brand': 'Ford', 'model': 'Mustang'}


# using del method:- Using **del** methods removes the item with the specified
**key name**.
del thisdict["model"]
print(thisdict)   # Output:-  {'brand': 'Ford', 'year': 1964}


# using del method without key name:- If we **can not difine the key name** del
method delete the dictionary completely.
# The Removing item using del methods keyword can also delete the dictionary
completely.
del thisdict
print(thisdict)   # Output:-  this will cause an error because "thisdict" no
longer exists.


# using clear method:- The Removing item using **clear()** methods method empties
the dictionary.
thisdict.clear()
print(thisdict)   # output:- Output:-  {}
```

Copy Dictionaries?

Ques. Copy Dictionaries?

```
thisdict =  {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

# using copy method:-
mydict = thisdict.copy()
print(mydict)   # Output:-  {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

# using dict() method:-
mydict = dict(thisdict)
print(mydict)   # Output:-  {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Loop Dictionaries?

Ques. Loop Dictionaries?

```
# print Dictionaries by key, value and item
a = {1:12 ,2:11 ,4:16 ,3:14 ,6:15 ,5:13 }
print(sorted(a.keys()))
print(sorted(a.values()))
print(sorted(a.items()))

Output:-
[1, 2, 3, 4, 5, 6]
[11, 12, 13, 14, 15, 16]
[(1, 12), (2, 11), (3, 14), (4, 16), (5, 13), (6, 15)]
```

- Print **all key names** in the dictionary, one by one.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)

Output:-
brand
model
year
```

- Print all values in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])

Output:-
Ford
Mustang
1964
```

- We can use the **keys()** method to return the keys of a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)

Output:-
brand
model
year
```

- You can also use the **values()** method to return values of a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)

Output:-
Ford
Mustang
1964
```

- Loop through both keys and values, by using the **items()** method.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)

Output:-
brand Ford
model Mustang
year 1964
```

Dictionary Methods?

Ques. Dictionary Methods?

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

```
car =   {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

# The **clear()** method removes all the elements from a dictionary.
car.clear()
print(car)    # Output:- {}


# The **copy()** method returns a copy of the specified dictionary.
x = car.copy()
print(x)    # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}


# The **get()** method returns the value of the item with the **specified key**.
x = car.get("model")
print(x)    # Output:- Mustang
#
x = car.get("price", 15000)
print(x)    # Output:- 15000
```

```
# The **pop()** method removes the specified item from the dictionary.
x = car.pop("model")
print(x)    # output:- Mustang
print(car)  # Output:- {'brand': 'Ford', 'year': 1964}


# The **popitem()** method removes the item that was last inserted into the
dictionary.
x = car.popitem()
print(x)    # Output:- ('year', 1964)
print(car)  # Output:- {'brand': 'Ford', 'model': 'Mustang'}
```

- The **fromkeys()** method returns a dictionary with the specified keys and the specified value.

```
# Example1:-
x = ('key1', 'key2', 'key3')
y = 0
thisdict = dict.fromkeys(x, y)
print(thisdict)

Output:- ['key1': 0, 'key2': 0, 'key3': 0]

# Example2:-
x = ('key1', 'key2', 'key3')
thisdict = dict.fromkeys(x)
print(thisdict)

Output:- ['key1': None, 'key2': None, 'key3': None]
```

- The **items()** method returns a view object. The view object contains the \n key-value pairs of the dictionary, as tuples in a list.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.items()
print(x)

Output:- dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

# example2:-
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
x = car.items()
car["year"] = 2018
print(x)

Output:- dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2018)])
```

- The **keys()** method returns a view object. The view object contains the keys of the dictionary, as a list.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.keys()
print(x)

car["color"] = "white"
print(x)

Output:- dict_keys(['brand', 'model', 'year'])
Output:- dict_keys(['brand', 'model', 'year', 'color'])
```

- The **setdefault()** method returns the value of the item with the specified key.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.setdefault("color", "White")
y = car.setdefault("model", "Bronco")

print(x)
print(y)

Output:- White
Output:- Mustang
```

- The **update()** method inserts the specified items to the dictionary.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
car.update({"color": "White"})
print(car)    # Output:- {'brand': 'Ford', 'model': 'Mustang', 'year': 1964,
'color': 'White'}
```

- The **values()** method returns a view object. The view object contains the values of the dictionary, as a list.

```
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = car.values()
print(x)    # Output:- dict_values(['Ford', 'Mustang', 1964])

# Example2:-
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.values()
car["year"] = 2018
print(x)    # Output:- dict_values(['Ford', 'Mustang', 2018])
```

Nested Dictionaries

Ques. Nested Dictionaries?

A dictionary can contain dictionaries, this is called nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)

Output:-
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year':
2007}, 'child3': {'name': 'Linus', 'year': 2011}}
----------------------------------------------------------------------------

# Example2:-
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily)

Output:-
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year':
2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

- Access Items in Nested Dictionaries

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Mohit",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily["child2"]["name"])
Output:- Mohit
```

Dictionaries Questions

Ques. How to Merging Or Adding two Dictionaries?

- Using **update()** method

```
dict_1 = {'John': 15, 'Rick': 10, 'Misa' : 12 }
dict_2 = {'Bonnie': 18,'Rick': 20,'Matt' : 16 }

dict_1.update(dict_2)
print(dict_1)

Output:- {'John': 15, 'Rick': 20, 'Misa': 12, 'Bonnie': 18, 'Matt': 16}
```

- Using ** operator:- The simplest way to merge two dictionaries in python is by using the unpack operator(**)

  ```
  dict_1 = {'John': 15, 'Rick': 20, 'Misa' : 12 }
  dict_2 = {'Bonnie': 18,'Rick': 10,'Matt' : 16 }
  dict_3 = {'Stefan': 19, 'Riya': 14, 'Lora': 17}
  dict_4 = {**dict_1,**dict_2, **dict_3}
  print (dict_4)

  Output:- {'John': 15, 'Rick': 10, 'Misa': 12, 'Bonnie': 18, 'Matt': 16,
  'Stefan': 19, 'Riya': 14, 'Lora': 17}
  ```

- Using for loop

```
dict1 = {'Alexandra' : 27, 'Shelina' : 22, 'James' : 29, 'Peterson' : 30 }
dict2 = {'Jasmine' : 19, 'Maria' : 26, 'Helena' : 30 }
dict3 = dict1.copy()

for key, value in dict2.items():
    dict3[key] = value

print(dict3)
Output:- {'Alexandra': 27, 'Shelina': 22, 'James': 29, 'Peterson': 30, 'Jasmine':
19, 'Maria': 26, 'Helena': 30}
```

- Unpacking the second dictionary

```
dict_1={'John': 15, 'Rick': 10, 'Misa' : 12 }
dict_2={'Bonnie': 18,'Rick': 20,'Matt' : 16 }
dict_3=dict(dict_1,**dict_2)
print (dict_3)
```

```
Output:- {'John': 15, 'Rick': 20, 'Misa': 12, 'Bonnie': 18, 'Matt': 16}
```

- Using collection.ChainMap() method

```
from collections import ChainMap
dict_1={'John': 15, 'Rick': 10, 'Misa' : 12 }
dict_2={'Bonnie': 18,'Rick': 20,'Matt' : 16 }
dict_3 = ChainMap(dict_1, dict_2)
print(dict_3)
print(dict(dict_3))

Output:-
ChainMap({'John': 15, 'Rick': 10, 'Misa': 12}, {'Bonnie': 18, 'Rick': 20, 'Matt':
16})
{'Bonnie': 18, 'Rick': 10, 'Matt': 16, 'John': 15, 'Misa': 12}
```

- Using itertools.chain()

```
import itertools
dict_1={'John': 15, 'Rick': 10, 'Misa': 12}
dict_2={'Bonnie': 18, 'Rick': 20, 'Matt': 16}
dict_3=itertools.chain(dict_1.items(),dict_2.items())
#Returns an iterator object
print (dict_3)
print(dict(dict_3))

Output:-
  <itertools.chain object at 0x7f34fd841220>
  {'John': 15, 'Rick': 20, 'Misa': 12, 'Bonnie': 18, 'Matt': 16}
```

- Using dictionary comprehension

```
dict_1={'John': 15, 'Rick': 10, 'Misa': 12}
dict_2={'Bonnie': 18, 'Rick': 20, 'Matt': 16}
dict_3={k:v for d in (dict_1,dict_2) for k,v in d.items()}
print (dict_3)

Output:- {'John': 15, 'Rick': 20, 'Misa': 12, 'Bonnie': 18, 'Matt': 16}
```

- Add values of common keys

```
dict_1 = {'John': 15, 'Rick': 10, 'Misa': 12}
dict_2 = {'Bonnie': 18, 'Rick': 20, 'Matt': 16}
```

```
def mergeDictionary(dict_1, dict_2):
   dict_3 = {**dict_1, **dict_2}
   for key, value in dict_3.items():
       if key in dict_1 and key in dict_2:
               dict_3[key] = [value , dict_1[key]]
   return dict_3

dict_3 = mergeDictionary(dict_1, dict_2)
print(dict_3)

Output:-
{'John': 15, 'Rick': [20, 10], 'Misa': 12, 'Bonnie': 18, 'Matt': 16}
```

- Using Zip function

```
d = {'k1': 1, 'k2': 2}

keys = ['k1', 'k3', 'k4']
values = [100, 3, 4]

d.update(zip(keys, values))
print(d)

Output:- {'k1': 100, 'k2': 2, 'k3': 3, 'k4': 4}
```

Ques. Combine two dictionary adding values for common keys?

- Using collections.Counter()

  ```
  from collections import Counter
  # initializing two dictionaries
  dict1 = {'a': 12, 'for': 25, 'c': 9}
  dict2 = {'Geeks': 100, 'geek': 200, 'for': 300}

  res = Counter(dict1) + Counter(dict2)
  print(res)

  Output:- Counter({'for': 325, 'geek': 200, 'Geeks': 100, 'a': 12, 'c': 9})
  ```

- Naive method

  ```
  dict1 = {'a': 12, 'for': 25, 'c': 9}
  dict2 = {'Geeks': 100, 'geek': 200, 'for': 300}
  # adding the values with common key
  for key in dict2:
      if key in dict1:
          dict2[key] = dict2[key] + dict1[key]
  ```

```
print(dict2)

Output:- {'Geeks': 100, 'geek': 200, 'for': 325}
```

Ques. What will the output of following Program?

```
dictlang = {'c#': 6, 'GO': 89, 'python': 4,'Rust':10,'Apple':51,'apple':21}

for i in sorted(dictlang):
    print (dictlang[i])

Output:-
51
89
10
21
6
4
```

Ques. Sort in dictionary with key, Value and Items.

- more sorted:- https://www.golinuxcloud.com/python-sort-dictionary-by-key/

```
dict = {6:'George' ,2:'John' ,1:'Potter' ,9:'Micheal' ,7:'Robert' ,8:'Gayle'}

b = sorted(dict.keys())
print("Sorted keys",b)

d = sorted(dict.values())
print("Sorted Values",d)

c = sorted(dict.items())
print("Sorted Values",c)

Output:-
Sorted keys [1, 2, 6, 7, 8, 9]
Sorted Values ['Gayle', 'George', 'John', 'Micheal', 'Potter', 'Robert']
Sorted Values [(1, 'Potter'), (2, 'John'), (6, 'George'), (7, 'Robert'), (8,
'Gayle'), (9, 'Micheal')]
```

- sort dictionary by key using for loop with **sorted()**

```
mydict_1 = {'key5': 'value5', 'key3': 'value3', 'key1': 'value1', 'key2':
'value2'}
new_dict = {}
sorted_value = sorted(mydict_1.keys())
```

```
for i in sorted_value:
   for key, value in mydict_1.items():
       if key == i:
           new_dict[key] = value

print(new_dict)
```

Ques. Can we use tuple as keys inside python dictionary?

- Yes, tuple can we used as key inside python dictionary, only if it contain only string, number or tuple. If a tuple contains any mutable datatype inside it like list, it can not be used as keys.

Ques. Can we use lists as keys inside python dictionary?

- No, python list can not be used as keys inside python dictionary, as they are mutable in nature.

Ques. What is enumarate function inside a dictionary?

- You can using enumarate function with dictionary to get position index and corresponding index at the same time.

```
for i in enumerate(dict1):
    print(i)

#output
(0, 'name')
(1, 'age')
(2, 'city')
```

Ques. What is zip function in python dictionary and how can you use it?

How can combine two dictionaries together?

```
questions = ['name', 'location', 'favorite language']
answers = ['Codersdaily', 'Indore', 'Python']
for q, a in zip(questions, answers):
    print('What is your {0}?  It is {1}.'.format(q, a))
```

Find Unique dictionary from list by id?

```
a = [
{'id':1,'name':'john', 'age':34},
{'id':1,'name':'john', 'age':34},
{'id':2,'name':'hanna', 'age':30},
]

b = {x['id']:x for x in a}.values()

print(b)

Output:-
dict_values([{'id': 1, 'name': 'john', 'age': 34}, {'id': 2, 'name': 'hanna',
'age': 30}])
```