Dictionary

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

In [ ]:

```
lists are mutables , indexable & ordered
dictionaries are mutable , Non-indexable & unoredered
tuples are immutables , indexable & ordered sequence
sets are mutables , non-indexable & un-ordered
strings are immutables , indexable & ordered.
```

In [ ]:

```
lists & dictionary are mutables
tupes & sets are immutables

lists & tuples are indexable
dictionary & sets are Not-indexable
```

Dict Creation

In [ ]:

```
#empty dictionary
my_dict = {}

#dictionary with integer keys
my_dict = {1: 'abc', 'k': 'xyz'}
print(my_dict)

#dictionary with mixed keys
my_dict = {'name': 'veera', 1: ['abc', 'xyz']}
print(my_dict)


#create empty dictionary using dict()
my_dict = dict()

my_dict = dict([(1, 'abc'), (2, 'xyz')])    #create a dict with list of tuples
print(my_dict)
```

Dict Access

In [2]:

```
my_dict = {'name': 'veera', 'age': 35, 'address': 'bangalore'}

#get name
print(my_dict['name'])
```

```
veera
```

In [3]:

```
#if key is not present it gives KeyError
print(my_dict['degree'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-3-ac4c061783ba> in <module>()
      1 #if key is not present it gives KeyError
```

```
----> 2 print(my_dict['degree'])

KeyError: 'degree'
```

In [4]:

```
#another way of accessing key
print(my_dict.get('address'))
```

```
bangalore
```

In [5]:

```
#if key is not present it will give None using get method
print(my_dict.get('degree'))
```

```
None
```

Dict Add or Modify Elements

In [6]:

```
my_dict = {'name': 'veera', 'age': 35, 'address': 'bangalore'}

#update name
my_dict['name'] = 'likhit'

print(my_dict)
```

```
{'name': 'likhit', 'age': 35, 'address': 'bangalore'}
```

In [ ]:

```
#add new key
my_dict['degree'] = 'B.Tech'

print(my_dict)
```

Dict Delete or Remove Element

In [7]:

```
#create a dictionary
my_dict = {'name': 'veera', 'age': 35, 'address': 'bangalore'}

#remove a particular item
print(my_dict.pop('age'))

print(my_dict)
```

```
35
{'name': 'veera', 'address': 'bangalore'}
```

In [16]:

```
my_dict = {'name': 'veera', 'age': 35, 'address': 'bangalore',5:22}

#remove an arbitarty key
my_dict.popitem()

print(my_dict)
```

```
{'name': 'veera', 'age': 35, 'address': 'bangalore'}
```

In [21]:

```
squares = {2: [4,5], 3: 9, 4: 16, 5: 25,2:200,3:20}

#delete particular key
del squares[2]

print(squares)
```

```
{3: 20, 4: 16, 5: 25}
```

In [ ]:

```
#remove all items
squares.clear()

print(squares)
```

In [ ]:

```
squares = {2: 4, 3: 9, 4: 16, 5: 25}

#delete dictionary itself
del squares

print(squares) #NameError because dict is deleted
```

In [22]:

```
squares = {}

dir (squares)
```

Out[22]:

```
['__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault'
```

```
 'setdefault',
 'update',
 'values']
```

Dictionary Methods

In [24]:

```
squares = {2: 4, 3: 9, 4: 16, 5: 25}

my_dict = squares.copy()
print(my_dict)
print (squares)

print (id(squares))
print (id(my_dict))

del squares[2]
print (my_dict)
print (squares)
```

```
{2: 4, 3: 9, 4: 16, 5: 25}
{2: 4, 3: 9, 4: 16, 5: 25}
2026250509696
2026250508256
{2: 4, 3: 9, 4: 16, 5: 25}
{3: 9, 4: 16, 5: 25}
```

In [30]:

```
#fromkeys[seq[, v]] -> Return a new dictionary with keys from seq and value equal to v (defaults t
o None).
subjects = {}.fromkeys(['python', 'go-lang', 'shell'], 1,2)
print(subjects)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-30-ebc9d2fed852> in <module>()
      1 #fromkeys[seq[, v]] -> Return a new dictionary with keys from seq and value equal to v
(defaults to None).
----> 2 subjects = {}.fromkeys(['python', 'go-lang', 'shell'], 1,2)
      3 print(subjects)

TypeError: fromkeys expected at most 2 arguments, got 3
```

In [31]:

```
subjects = {2:4, 3:9, 4:16, 5:25}
print(subjects.items()) #return a new view of the dictionary items (key, value)
```

```
dict_items([(2, 4), (3, 9), (4, 16), (5, 25)])
```

In [32]:

```
subjects = {2:4, 3:9, 4:16, 5:25}
print(subjects.keys()) #return a new view of the dictionary keys
```

```
dict_keys([2, 3, 4, 5])
```

In [33]:

```
subjects = {2:4, 3:9, 4:16, 5:25}
print(subjects.values()) #return a new view of the dictionary values
```

```
dict_values([4, 9, 16, 25])
```

In [ ]:

```
#get list of all available methods and attributes of dictionary
d = {}
print(dir(d))
```

Dict Comprehension

In [ ]:

```
#Dict comprehensions are just like list comprehensions but for dictionaries

d = {'a': 1, 'b': 2, 'c': 3}
for pair in d.items():
    print(pair)
```

In [34]:

```
#Creating a new dictionary with only pairs where the value is larger than 2
d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
new_dict = {k:v for k, v in d.items() if v > 2}
print(new_dict)
```

```
{'c': 3, 'd': 4}
```

In [35]:

```
#We can also perform operations on the key value pairs
d = {'a':1,'b':2,'c':3,'d':4,'e':5}
d = {k + 'c':v * 2 for k, v in d.items() if v > 2}
print(d)
```

```
{'cc': 6, 'dc': 8, 'ec': 10}
```

In [ ]:

```
#dictionary with mixed keys
my_dict = {'name': 'veera', 1: ['abc', 'xyz'],20.2:"veera",'diff_name': 'nani',20:"lveera"}

k=id(my_dict)
print (k)
l= id(my_dict["name"])
print (l)
m= id(my_dict[20.2])
print (m)

n= id(my_dict["diff_name"])
print (n)

o=id(my_dict[20])
print (o)
```

In [ ]:

```
my_dict = dict([(1, 'abc'), (2, '[1,2]')])

print (id("2"))
print (id(2)[0])
```

In [36]:

```
my_dict = dict([(1, 'abc'), (2, [1,2])])

my_dict[2].append([20,30])
print (my_dict)
```

```
{1: 'abc', 2: [1, 2, [20, 30]]}
```

In [42]:

```
d = {2: [(10,100)], 'b': [2]}
#print (d)
#d[2].append(5)
#print (d)

print(d[2].pop(0))

print (d)
```

```
(10, 100)
{2: [], 'b': [2]}
```

In [2]:

```
d1={1:"hello",2:"world"}
d2={3:"welcome to",4:"MG"}
d3={**d1,**d2}
print (d3)
```

```
{1: 'hello', 2: 'world', 3: 'welcome to', 4: 'MG'}
```