Dictionary : Is like a Hash Table

Hash Table is studied in data structures (CS/ IT- Undergraduate). Python dictionary is an ***unordered collection*** (No indexing is possible) of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Dict = { Key : 'Value' } -----> We put curly braces to store (Here key is integer}

Dict = { 'Key' : 'Value' } ------> Here key is string

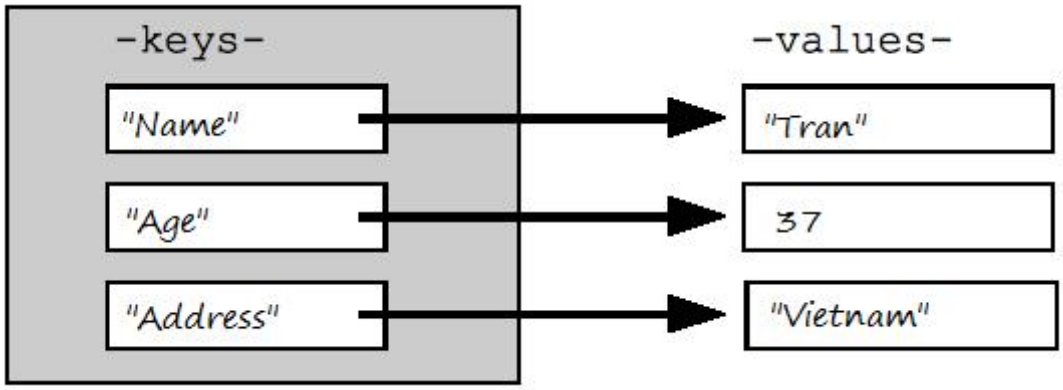
Dict could have mixed keys ¶

Dictonary can be thought of as a table containing Key and its corrosponding value with colon(:) in between key and value

In [1]:

```
from IPython.display import Image
Image(filename='C:\\Users\\Milan Joshi\\Desktop\\table.png')
```

Out[1]:



Dict Creation

In [15]:

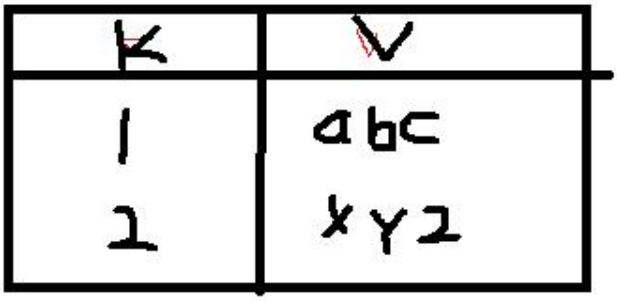
```
#empty dictionary
my_dict = {}

#dictionary with integer keys
my_dict = {1: 'abc', 2: 'xyz'}   # 1 is the key and 'abc' is its value also 2 is the key 'x


# For key 1 we store 'abc' and for key 2 we store 'xyz' by the way my_dict create a table w

print(my_dict)

#dictionary with mixed keys

my_dict = {'name': 'Milan', 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)
```

```
{1: 'abc', 2: 'xyz'}
{'name': 'Milan', 1: ['abc', 'xyz']}
{1: 'abc', 2: 'xyz'}
```

In [11]:

```
Image(filename='C:\\Users\\Milan Joshi\\Desktop\\dict.png')
```

Out[11]:



In [12]:

```
Image(filename='C:\\Users\\Milan Joshi\\Desktop\\dict.png')
```

Out[12]:

Dict Access

In [18]:

```
my_dict = {'name': 'Milan', 'age': 32, 'address': 'Shirpur'}  # key 1 = k1 , key2 = k2 , ke

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

```
Milan
```

In [19]:

```
#if key is not present it gives KeyError
print(my_dict['degree'])  # There is no key as degree  in my_dict above
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-a24c24b0fec2> in <module>()
      1 #if key is not present it gives KeyError
----> 2 print(my_dict['degree'])  # There is no key as degree  in my_dict ab
ove

KeyError: 'degree'
```

In [20]:

```
#another way of accessing key
print(my_dict.get('address'))   # .get is a function in dictonary
```

```
Shirpur
```

In [22]:

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

```
None
```

Dict Add or Modify Elements

In [27]:

```
my_dict = {'name': 'Milan', 'age': 32, 'address': 'Shirpur'}

#update name
my_dict['name'] = 'joshi'  # Update a value given key

print(my_dict)
```

```
{'name': 'joshi', 'age': 32, 'address': 'Shirpur'}
```

In [26]:

```
#add new key
my_dict['degree'] = 'Ph.D(Math)'

print(my_dict)
```

```
{'name': 'joshi', 'age': 32, 'address': 'Shirpur', 'degree': 'Ph.D(Math)'}
```

Dict Delete or Remove Element

In [29]:

```
#create a dictionary
my_dict = {'name': 'Milan', 'age': 32, 'address': 'Shirpur'}

#remove a particular item..Return the value but remove it
print(my_dict.pop('age'))

print(my_dict)
```

```
32
{'name': 'Milan', 'address': 'Shirpur'}
```

In [33]:

```
my_dict = {'name': 'Milan', 'age': 32, 'address': 'Shirpur'}

#remove an arbitarty key
my_dict.popitem()

print(my_dict)
```

```
{'name': 'Milan', 'age': 32}
```

In [14]:

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

#delete particular key
del squares[2]

print(squares)
```

```
{3: 9, 4: 16, 5: 25}
```

In [15]:

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

print(squares)
```

```
{}
```

In [34]:

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

#delete dictionary itself
del squares

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-34-6fac7747ecad> in <module>()
      4 del squares
      5
----> 6 print(squares) #NameError because dict is deleted

NameError: name 'squares' is not defined
```

Dictionary Methods

In [17]:

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

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

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

In [18]:

```
#fromkeys[seq[, v]] -> Return a new dictionary with keys from seq and value equal to v (def
subjects = {}.fromkeys(['Math', 'English', 'Hindi'], 0)
print(subjects)
```

```
{'Math': 0, 'English': 0, 'Hindi': 0}
```

In [35]:

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

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

In [20]:

```
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 [21]:

```
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 [24]:

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

```
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__do
c__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__
repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclassh
ook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popite
m', 'setdefault', 'update', 'values']
```

Dict Comprehension

In [44]:

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

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

```
('a', 1)
('b', 2)
('c', 3)
```

In [50]:

```
#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 > 1}
print(new_dict)
```

```
{'b': 2, 'c': 3, 'd': 4}
```

In [51]:

```
#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}  # k+'c' concatinate c with the keys wh
# greater than 2 and multiply vlaue then by 2

print(d)
```

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

In [ ]:

```
```