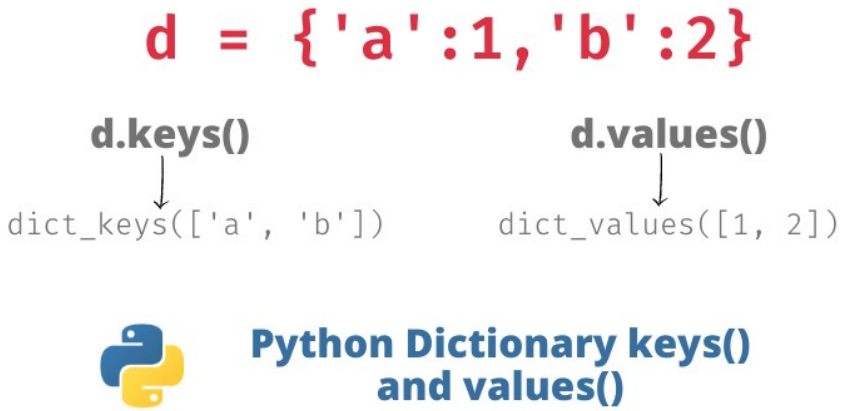
Dictionaries are unordered collections of unique values stored in (Key-Value) pairs.

Python dictionary represents a mapping between a key and a value. In simple terms, a Python dictionary can store pairs of keys and values. Each key is linked to a specific value. Once stored in a dictionary, you can later obtain the value using just the key.

In [23]:
```
from IPython.display import Image
Image("C:\\Users\\deepali\\OneDrive\\Desktop\\python-dict-keys-and-values.jpg",width=500)
```

Out[23]:



Characteristics of dictionaries

- **Unordered**: The items in dictionaries are stored without any index value, which is typically a range of numbers. They are stored as Key-Value pairs, and the keys are their index, which will not be in any sequence.
- **Unique**: As mentioned above, each value has a Key; the Keys in Dictionaries should be unique. If we store any value with a Key that already exists, then the most recent value will replace the old value.
- **Mutable**: The dictionaries are collections that are changeable, which implies that we can add or remove items after the creation.

Creating a dictionary

There are following three ways to create a dictionary.

1. Using curly brackets: The dictionaries are created by enclosing the comma-separated Key: Value pairs inside the {} curly brackets. The colon ':' is used to separate the key and value in a pair.
2. Using dict() constructor: Create a dictionary by passing the comma-separated key: value pairs inside the dict().
3. Using sequence having each item as a pair (key-value)

In [1]:
```
# create a dictionary using {}
person = {"Name": "Deepali", "Country": "India", "Telephone": 56789}
print(person)

# create a dictionary using dict()
person = dict({"Name": "Deepali", "Country": "India", "Telephone": 56789})
print(person)

# create a dictionary from sequence having each item as a pair
person = dict([("Name", "Harsh"), ("Country", "USA"), ("Telephone", 11789)])
print(person)

# create dictionary with mixed keys keys
# first key is string and second is an integer
sample_dict = {"Name": "Deepali", 10: "Mobile"}
print(sample_dict)

# create dictionary with value as a list
person = {"Name": "Deepali", "Telephones": [1178, 2563, 4569]}
print(person)

```

```
{'Name': 'Deepali', 'Country': 'India', 'Telephone': 56789}
{'Name': 'Deepali', 'Country': 'India', 'Telephone': 56789}
{'Name': 'Harsh', 'Country': 'USA', 'Telephone': 11789}
{'Name': 'Deepali', 10: 'Mobile'}
{'Name': 'Deepali', 'Telephones': [1178, 2563, 4569]}
```

Empty Dictionary

When we create a dictionary without any elements inside the curly brackets then it will be an empty dictionary.

In [2]:
```
emptydict = {}
print(type(emptydict))

```

```
<class 'dict'>
```

Note:

- A dictionary value can be of any type, and duplicates are allowed in that.
- Keys in the dictionary must be unique and of immutable types like string, numbers, or tuples.

Accessing elements of a dictionary

There are two different ways to access the elements of a dictionary.

1. Retrieve value using the key name inside the [] square brackets
2. Retrieve value by passing key name as a parameter to the get() method of a dictionary.

In [3]:
```
# create a dictionary named person
person = {"Name": "Deepali", "Country": "India", "Telephone": 56789}

# access value using key name in []
print(person['Name'])

#  get key value using key name in get()
print(person.get('Telephone'))

```

```
Deepali
56789
```

As we can see in the output, we retrieved the value ‘Jessa’ using key ‘name” and value 1178 using its Key ‘telephone’.

Get all keys and values

Use the following dictionary methods to retrieve all key and values at once

Method	Description
keys()	Returns the list of all keys present in the dictionary.
values()	Returns the list of all values present in the dictionary
items()	Returns all the items present in the dictionary. Each item will be inside a tuple as a key-value pair.

In [4]:
```
person = {"name": "Deepali", "country": "India", "telephone": 1178}

# Get all keys
print(person.keys())

print(type(person.keys()))


# Get all values
print(person.values())
print(type(person.values()))

# Get all key-value pair
print(person.items())
print(type(person.items()))

```

```
dict_keys(['name', 'country', 'telephone'])
<class 'dict_keys'>
dict_values(['Deepali', 'India', 1178])
<class 'dict_values'>
dict_items([('name', 'Deepali'), ('country', 'India'), ('telephone', 1178)])
<class 'dict_items'>
```

Iterating a dictionary

We can iterate through a dictionary using a for-loop and access the individual keys and their corresponding values.

In [5]:
```
person = {"name": "Deepali", "country": "India", "telephone": 1178}

# Iterating the dictionary using for-loop
print('key', ':', 'value')
for key in person:
    print(key, ':', person[key])

print("\n")

# using items() method
print('key', ':', 'value')
for key_value in person.items():
    # first is key, and second is value
    print(key_value[0], key_value[1])
```

```
key : value
name : Deepali
country : India
telephone : 1178


key : value
name Deepali
country India
telephone 1178
```

Find a length of a dictionary

In order to find the number of items in a dictionary, we can use the len() function.

In [6]:
```
person = {"name": "Deepali", "country": "India", "telephone": 1178}

# count number of keys present in  a dictionary
print(len(person))

```

```
3
```

Adding items to the dictionary

We can add new items to the dictionary using the following two ways.

- Using key-value assignment: Using a simple assignment statement where value can be assigned directly to the new key.
- Using update() Method: In this method, the item passed inside the update() method will be inserted into the dictionary. The item can be another dictionary or any iterable like a tuple of key-value pairs.

In [7]:
```
person = {"name": "Deepali", "country": "India", "telephone": 1178}

# update dictionary by adding 2 new keys
person["weight"] = 50
person.update({"height": 6})

# print the updated dictionary
print(person)

```

```
{'name': 'Deepali', 'country': 'India', 'telephone': 1178, 'weight': 50, 'height': 6}
```

Set default value to a key

Using the setdefault() method default value can be assigned to a key in the dictionary. In case the key doesn’t exist already, then the key will be inserted into the dictionary, and the value becomes the default value, and None will be inserted if a value is not mentioned.

In case the key exists, then it will return the value of a key.

In [8]:
```
person_details = {"name": "Deepali", "country": "India", "telephone": 1178}

# set default value if key doesn't exists
person_details.setdefault('state', 'Texas')

# key doesn't exists and value not mentioned. default None
person_details.setdefault("zip")

# key exists and value mentioned. doesn't  change value
person_details.setdefault('country', 'Canada')

# Display dictionary
for key, value in person_details.items():
    print(key, ':', value)
```

```
name : Deepali
country : India
telephone : 1178
state : Texas
zip : None
```

Modify the values of the dictionary keys

We can modify the values of the existing dictionary keys using the following two ways.

- Using key name: We can directly assign new values by using its key name. The key name will be the existing one and we can mention the new value.
- Using update() method: We can use the update method by passing the key-value pair to change the value. Here the key name will be the existing one, and the value to be updated will be new.

In [9]:
```
person = {"name": "Deepali", "country": "India"}

# updating the country name
person["country"] = "Canada"
# print the updated country
print(person['country'])

# updating the country name using update() method
person.update({"country": "USA"})
# print the updated country
print(person['country'])

```

```
Canada
USA
```

Removing items from the dictionary

There are several methods to remove items from the dictionary. Whether we want to remove the single item or the last inserted item or delete the entire dictionary, we can choose the method to be used.

Use the following dictionary methods to remove keys from a dictionary.

Method	Description
pop(key[,d])	Return and removes the item with the key and return its value. If the key is not found, it raises KeyError.
popitem()	Return and removes the last inserted item from the dictionary. If the dictionary is empty, it raises KeyError.
del key	The del keyword will delete the item with the key that is passed
clear()	Removes all items from the dictionary. Empty the dictionary
del dict_name	Delete the entire dictionary

In [10]:
```
person = {'name': 'Deepali', 'country': 'India', 'telephone': 2834, 'weight': 58, 'height': 6}

# Remove last inserted item from the dictionary
deleted_item = person.popitem()
print(deleted_item)  # output ('height', 6)
# display updated dictionary
print(person)

# Remove key 'telephone' from the dictionary
deleted_item = person.pop('telephone')
print(deleted_item)  # output 1178
# display updated dictionary
print(person)

# delete key 'weight'
del person['weight']
# display updated dictionary
print(person)

# remove all item (key-values) from dict
person.clear()
# display updated dictionary
print(person)  # {}

# Delete the entire dictionary
del person
```

```
('height', 6)
{'name': 'Deepali', 'country': 'India', 'telephone': 2834, 'weight': 58}
2834
{'name': 'Deepali', 'country': 'India', 'weight': 58}
{'name': 'Deepali', 'country': 'India'}
{}
```

Checking if a key exists

In order to check whether a particular key exists in a dictionary, we can use the keys() method and in operator. We can use the in operator to check whether the key is present in the list of keys returned by the keys() method.

In this method, we can just check whether our key is present in the list of keys that will be returned from the keys() method.

In [11]:
```
person = {'name': 'Deepali', 'country': 'India', 'telephone': 2834, 'weight': 50, 'height': 6}

# Get the list of keys and check if 'country' key is present
key_name = 'country'
if key_name in person.keys():
    print("country name is", person[key_name])
else:
    print("Key not found")

```

```
country name is India
```

Join two dictionary

We can add two dictionaries using the update() method or unpacking arbitrary keywords operator **.

Using update() method

In this method, the dictionary to be added will be passed as the argument to the update() method and the updated dictionary will have items of both the dictionaries.

In [12]:
```
dict1 = {'Deepali': 25, 'Arun': 80, 'Meena': 55}
dict2 = {'Kelvin': 68, 'Harry': 50, 'Om': 66}

# copy second dictionary into first dictionary
dict1.update(dict2)
# printing the updated dictionary
print(dict1)

```

```
{'Deepali': 25, 'Arun': 80, 'Meena': 55, 'Kelvin': 68, 'Harry': 50, 'Om': 66}
```

Using **kwargs to unpack

We can unpack any number of dictionary and add their contents to another dictionary using **kwargs. In this way, we can add multiple length arguments to one dictionary in a single statement.

In [13]:
```
student_dict1 = {'Deepali': 1, 'Arun': 2, }
student_dict2 = {'Farhan': 5, 'Om': 6}
student_dict3 = {'Nancy': 7, 'Priya': 9}

# join three dictionaries
student_dict = {**student_dict1, **student_dict2, **student_dict3}
# printing the final Merged dictionary
print(student_dict)

```

```
{'Deepali': 1, 'Arun': 2, 'Farhan': 5, 'Om': 6, 'Nancy': 7, 'Priya': 9}
```

Add multiple dictionaries inside a single dictionary

In this example, we will create a separate dictionary for each student and in the end, we will add each student to the ‘class_six’ dictionary. So each student is nothing but a key in a ‘class_six’ dictionary.

In order to access the nested dictionary values, we have to pass the outer dictionary key, followed by the individual dictionary key.

For example, class_six['student3']['name']

We can iterate through the individual member dictionaries using nested for-loop with the outer loop for the outer dictionary and inner loop for retrieving the members of the collection.

In [14]:
```
# each dictionary will store data of a single student
Deepali = {'Name': 'Deepali', 'State': 'Maharashtra', 'City': 'Mumbai', 'Marks': 75}
Emma = {'Name': 'Emma', 'State': 'Texas', 'City': 'Dallas', 'Marks': 60}
kelvin = {'Name': 'Kelvin', 'State': 'Texas', 'City': 'Austin', 'Marks': 85}

# Outer dictionary to store all student dictionaries (nested dictionaries)
class_six = {'student1': Deepali, 'student2': Emma, 'student3': kelvin}

# Get student3's name and mark
print("Student 3 name:", class_six['student3']['Name'])
print("Student 3 marks:", class_six['student3']['Marks'])

# Iterating outer dictionary
print("\nClass details\n")
for key, value in class_six.items():
    # Iterating through nested dictionary
    # Display each student data
    print(key)
    for nested_key, nested_value in value.items():
        print(nested_key, ':', nested_value)
    print('\n')

```

```
Student 3 name: Kelvin
Student 3 marks: 85

Class details

student1
Name : Deepali
State : Maharashtra
City : Mumbai
Marks : 75


student2
Name : Emma
State : Texas
City : Dallas
Marks : 60
```

```
student3
Name : Kelvin
State : Texas
City : Austin
Marks : 85
```

Dictionary comprehension

Dictionary comprehension is one way of creating the dictionary where the values of the key values are generated in a for-loop and we can filter the items to be added to the dictionary with an optional if condition.

In [15]:
```
# calculate the square of each even number from a list and store in dict
numbers = [1, 3, 5, 2, 8]
even_squares = {x: x ** 2 for x in numbers if x % 2 == 0}
print(even_squares)
```

```
{2: 4, 8: 64}
```

Here in this example, we can see that a dictionary is created with an input list (any iterable can be given), the numbers from the list being the key and the value is the square of a number.

We can even have two different iterables for the key and value and zip them inside the for loop to create a dictionary.

In [16]:
```
telephone_book = [1178, 4020, 5786]
persons = ['Deepali', 'Emma', 'Kelvin']

telephone_Directory = {key: value for key, value in zip(persons, telephone_book)}
print(telephone_Directory)
```

```
{'Deepali': 1178, 'Emma': 4020, 'Kelvin': 5786}
```

we are creating a telephone directory with separate tuples for the key which is the name, and the telephone number which is the value. We are zipping both the tuples together inside the for a loop.

Python Built-in functions with dictionary

max() and min() As the name suggests the max() and min() functions will return the keys with maximum and minimum values in a dictionary respectively. Only the keys are considered here not their corresponding values.

In [17]:
```
dict = {1:'aaa',2:'bbb',3:'AAA'}
print('Maximum Key',max(dict)) # 3
print('Minimum Key',min(dict)) # 1
```

```
Maximum Key 3
Minimum Key 1
```

all()

When the built-in function all() is used with the dictionary the return value will be true in the case of all – true keys and false in case one of the keys is false.

Few things to note here are

- Only key values should be true
- The key values can be either True or 1 or '0'
- 0 and False in Key will return false
- An empty dictionary will return true.

In [18]:
```
#dictionary with both 'true' keys
dict1 = {1:'True',1:'False'}

#dictionary with one false key
dict2 = {0:'True',1:'False'}

#empty dictionary
dict3= {}

#'0' is true actually
dict4 = {'0':False}

print('All True Keys::',all(dict1))
print('One False Key',all(dict2))
print('Empty Dictionary',all(dict3))
print('With 0 in single quotes',all(dict4))
```

```
All True Keys:: True
One False Key False
Empty Dictionary True
With 0 in single quotes True
```

any()

any() function will return true if dictionary keys contain anyone false which could be 0 or false.

In [19]:
```
#dictionary with both 'true' keys
dict1 = {1:'True',1:'False'}

#dictionary with one false key
dict2 = {0:'True',1:'False'}

#empty dictionary
dict3= {}

#'0' is true actually
dict4 = {'0':False}

#all false
dict5 = {0:False}

print('All True Keys::',any(dict1))
print('One False Key ::',any(dict2))
print('Empty Dictionary ::',any(dict3))
print('With 0 in single quotes ::',any(dict4))
print('all false :: ',any(dict5))
```

```
All True Keys:: True
One False Key :: True
Empty Dictionary :: False
With 0 in single quotes :: True
all false ::  False
```

When to use dictionaries?

Dictionaries are items stored in Key-Value pairs that actually use the mapping format to actually store the values. It uses hashing internally for this. For retrieving a value with its key, the time taken will be very less as O(1).

For example, consider the phone lookup where it is very easy and fast to find the phone number (value) when we know the name (key) associated with it.

So to associate values with keys in a more optimized format and to retrieve them efficiently using that key, later on, dictionaries could be used.

Summary of dictionary operations

Summary of dictionary operations

```
Assume d1 and d2 are dictionaries with following items.

d1 = {'a': 10, 'b': 20, 'c': 30}
d2 = {'d': 40, 'e': 50, 'f': 60}
```

Operations	Description
dict({'a': 10, 'b': 20})	Create a dictionary using a dict() constructor.
d2 = {}	Create an empty dictionary.
d1.get('a')	Retrieve value using the key name a.
d1.keys()	Returns a list of keys present in the dictionary.
d1.values()	Returns a list with all the values in the dictionary.
d1.items()	Returns a list of all the items in the dictionary with each key-value pair inside a tuple.
len(d1)	Returns number of items in a dictionary.
d1['d'] = 40	Update dictionary by adding a new key.
d1.update({'e': 50, 'f': 60})	Add multiple keys to the dictionary.
d1.setdefault('g', 70)	Set the default value if a key doesn't exist.
d1['b'] = 100	Modify the values of the existing key.
d1.pop('b')	Remove the key b from the dictionary.
d1.popitem()	Remove any random item from a dictionary.
d1.clear()	Removes all items from the dictionary.
'key' in d1.keys()	Check if a key exists in a dictionary.
d1.update(d2)	Add all items of dictionary d2 into d1.
d3= {**d1,* *d2}	Join two dictionaries.
d2 = d1.copy()	Copy dictionary d1 into d2.
max(d1)	Returns the key with the maximum value in the dictionary d1
min(d1)	Returns the key with the minimum value in the dictionary d1