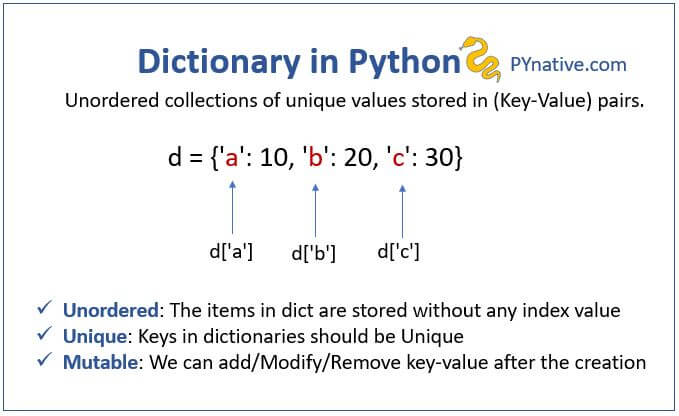
**Dictionaries in Python**

**Dictionaries are ordered 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.

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.



### Characteristics of dictionaries

* **Unordered** (In Python 3.6 and lower version): 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.
* **Ordered** (In Python 3.7 and higher version): dictionaries are ordered, which means that the items have a defined order, and that order will not change. A simple Hash Table consists of key-value pair arranged in pseudo-random order based on the calculations from Hash Function.
* **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 changeable collections, which implies that we can add or remove items after the creation.

## Table of contents

* [Characteristics of dictionaries](https://pynative.com/python-dictionaries/#h-characteristics-of-dictionaries)
* [Creating a dictionary](https://pynative.com/python-dictionaries/#h-creating-a-dictionary)
  + [Empty Dictionary](https://pynative.com/python-dictionaries/#h-empty-dictionary)
* [Accessing elements of a dictionary](https://pynative.com/python-dictionaries/#h-accessing-elements-of-a-dictionary)
  + [Get all keys and values](https://pynative.com/python-dictionaries/#h-get-all-keys-and-values)
  + [Iterating a dictionary](https://pynative.com/python-dictionaries/#h-iterating-a-dictionary)
* [Find a length of a dictionary](https://pynative.com/python-dictionaries/#h-find-a-length-of-a-dictionary)
* [Adding items to the dictionary](https://pynative.com/python-dictionaries/#h-adding-items-to-the-dictionary)
  + [Set default value to a key](https://pynative.com/python-dictionaries/#h-set-default-value-to-a-key)
* [Modify the values of the dictionary keys](https://pynative.com/python-dictionaries/#h-modify-the-values-of-the-dictionary-keys)
* [Removing items from the dictionary](https://pynative.com/python-dictionaries/#h-removing-items-from-the-dictionary)
* [Checking if a key exists](https://pynative.com/python-dictionaries/#h-checking-if-a-key-exists)
* [Join two dictionary](https://pynative.com/python-dictionaries/#h-join-two-dictionary)
  + [Using update() method](https://pynative.com/python-dictionaries/#h-using-update-method)
  + [Using \*\*kwargs to unpack](https://pynative.com/python-dictionaries/#h-using-kwargs-to-unpack)
  + [Join two dictionaries having few items in common](https://pynative.com/python-dictionaries/#h-join-two-dictionaries-having-few-items-in-common)
* [Copy a Dictionary](https://pynative.com/python-dictionaries/#h-copy-a-dictionary)
  + [Copy using the assignment operator](https://pynative.com/python-dictionaries/#h-copy-using-the-assignment-operator)
* [Nested dictionary](https://pynative.com/python-dictionaries/#h-nested-dictionary)
  + [Add multiple dictionaries inside a single dictionary](https://pynative.com/python-dictionaries/#h-add-multiple-dictionaries-inside-a-single-dictionary)
* [Sort dictionary](https://pynative.com/python-dictionaries/#h-sort-dictionary)
* [Dictionary comprehension](https://pynative.com/python-dictionaries/#h-dictionary-comprehension)
* [Python Built-in functions with dictionary](https://pynative.com/python-dictionaries/#h-python-built-in-functions-with-dictionary)
  + [max() and min()](https://pynative.com/python-dictionaries/#h-max-and-min)
  + [all()](https://pynative.com/python-dictionaries/#h-all)
  + [any()](https://pynative.com/python-dictionaries/#h-any)
  + [When to use dictionaries?](https://pynative.com/python-dictionaries/#h-when-to-use-dictionaries)
* [Summary of dictionary operations](https://pynative.com/python-dictionaries/#h-summary-of-dictionary-operations)

## Creating a dictionary

There are following three ways to create a dictionary.

* **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.
* **Using dict() constructor**:  Create a dictionary by passing the comma-separated key: value pairs inside the dict().
* **Using sequence** having each item as a pair (key-value)

Let’s see each one of them with an example.

**Example**:

*# create a dictionary using {}*person = {"name": "Jessa", "country": "USA", "telephone": 1178}  
print(person)  
*# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}  
  
# create a dictionary using dict()*person = dict({"name": "Jessa", "country": "USA", "telephone": 1178})  
print(person)  
*# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}  
  
# create a dictionary from sequence having each item as a pair*person = dict([("name", "Mark"), ("country", "USA"), ("telephone", 1178)])  
print(person)  
  
*# create dictionary with mixed keys  
# first key is string and second is an integer*sample\_dict = {"name": "Jessa", 10: "Mobile"}  
print(sample\_dict)  
*# output {'name': 'Jessa', 10: 'Mobile'}  
  
# create dictionary with value as a list*person = {"name": "Jessa", "telephones": [1178, 2563, 4569]}  
print(person)  
*# output {'name': 'Jessa', '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.

emptydict = {}

**print**(**type**(emptydict))

# Output 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.

**Example**

*# create a dictionary named person*person = {"name": "Jessa", "country": "USA", "telephone": 1178}  
  
*# access value using key name in []*print(person['name'])  
*# Output 'Jessa'  
  
# get key value using key name in get()*print(person.get('telephone'))  
*# Output 1178*

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. |

We can assign each method’s output to a separate variable and use that for further computations if required.

person = {"name": "Jessa", "country": "USA", "telephone": 1178}  
  
*# Get all keys*print(person.keys())  
*# output dict\_keys(['name', 'country', 'telephone'])*print(type(person.keys()))  
*# Output class 'dict\_keys'  
  
# Get all values*print(person.values())  
*# output dict\_values(['Jessa', 'USA', 1178])*print(type(person.values()))   
*# Output class 'dict\_values'  
  
# Get all key-value pair*print(person.items())  
*# output dict\_items([('name', 'Jessa'), ('country', 'USA'), ('telephone', 1178)])*print(type(person.items()))  
*# Output 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. Let us see this with an example.

person = {"name": "Jessa", "country": "USA", "telephone": 1178}  
  
*# Iterating the dictionary using for-loop*print('key', ':', 'value')  
for key in person:  
 print(key, ':', person[key])  
  
*# 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])

**OUTPUT:**

**key : value**

**name : Jessa**

**country : USA**

**telephone : 1178**

**key : value**

**name Jessa**

**country USA**

**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. Let us consider the personal details dictionary which we created in the above example and find its length.

person = {"name": "Jessa", "country": "USA", "telephone": 1178}  
  
*# count number of keys present in a dictionary*print(len(person))   
*# output 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.

Now, Let’s see how to add two new keys to the dictionary.

**Example**

person = {"name": "Jessa", 'country': "USA", "telephone": 1178}  
  
*# update dictionary by adding 2 new keys*person["weight"] = 50  
person.update({"height": 6})  
  
*# print the updated dictionary*print(person)  
*# output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50, 'height': 6}*

**Note**: We can also add more than one key using the update() method.

**Example**

person = {"name": "Jessa", 'country': "USA"}  
  
*# Adding 2 new keys at once  
# pass new keys as dict*person.update({"weight": 50, "height": 6})  
*# print the updated dictionary*print(person)  
*# output {'name': 'Jessa', 'country': 'USA', 'weight': 50, 'height': 6}  
  
# pass new keys as as list of tuple*person.update([("city", "Texas"), ("company", "Google")])  
*# print the updated dictionary*print(person)  
*# output {'name': 'Jessa', 'country': 'USA', 'weight': 50, 'height': 6, 'city': 'Texas', 'company': 'Google'}*

### **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.

**Example**

person\_details = {"name": "Jessa", "country": "USA", "telephone": 1178}  
  
*# set default value if key doesn't exists*person\_details.setdefault('state', 'Texas')  
  
*# key doesn't exist 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)

**OUTPUT:**

**name : Jessa**

**country : USA**

**telephone : 1178**

**state : Texas**

**zip : None**

**Note**: As seen in the above example the value of the setdefault() method has no effect on the ‘country’ as the key already exists.

## **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.

**Example**

person = {"name": "Jessa", "country": "USA"}  
  
*# updating the country name*person["country"] = "Canada"  
*# print the updated country*print(person['country'])  
*# Output 'Canada'  
  
# updating the country name using update() method*person.update({"country": "USA"})  
*# print the updated country*print(person['country'])  
*# Output '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 |

Now, Let’s see how to delete items from a dictionary with an example.

**Example**

person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50, 'height': 6}  
  
*# Remove last inserted item from the dictionary*deleted\_item = person.popitem()  
print(deleted\_item) *# output ('height', 6)  
# display updated dictionary*print(person)   
*# Output {'name': 'Jessa', 'country': 'USA', 'telephone': 1178, 'weight': 50}  
  
# Remove key 'telephone' from the dictionary*deleted\_item = person.pop('telephone')  
print(deleted\_item) *# output 1178  
# display updated dictionary*print(person)   
*# Output {'name': 'Jessa', 'country': 'USA', 'weight': 50}  
  
# delete key 'weight'*del person['weight']  
*# display updated dictionary*print(person)  
*# Output {'name': 'Jessa', 'country': 'USA'}  
  
# remove all item (key-values) from dict*person.clear()  
*# display updated dictionary*print(person) *# {}  
  
# Delete the entire dictionary*del person

## **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.

Let’s check whether the ‘country’ key exists and prints its value if found.

person = {'name': 'Jessa', 'country': 'USA', 'telephone': 1178}  
  
*# 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")  
*# Output country name is USA*

## **Join two dictionary**

We can add two dictionaries using the update() method or unpacking arbitrary keywords operator \*\*. Let us see each one with an example.

### **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.

Let’s see how to merge the second dictionary into the first dictionary

**Example**

dict1 = {'Jessa': 70, 'Arul': 80, 'Emma': 55}  
dict2 = {'Kelly': 68, 'Harry': 50, 'Olivia': 66}  
  
*# copy second dictionary into first dictionary*dict1.update(dict2)  
*# printing the updated dictionary*print(dict1)  
*# output {'Jessa': 70, 'Arul': 80, 'Emma': 55, 'Kelly': 68, 'Harry': 50, 'Olivia': 66}*

As seen in the above example the items of both the dictionaries have been updated and the dict1 will have the items from both the dictionaries.

### **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.

student\_dict1 = {'Aadya': 1, 'Arul': 2, }  
student\_dict2 = {'Harry': 5, 'Olivia': 6}  
student\_dict3 = {'Nancy': 7, 'Perry': 9}  
  
*# join three dictionaries*student\_dict = {\*\*student\_dict1, \*\*student\_dict2, \*\*student\_dict3}  
*# printing the final Merged dictionary*print(student\_dict)  
*# Output {'Aadya': 1, 'Arul': 2, 'Harry': 5, 'Olivia': 6, 'Nancy': 7, 'Perry': 9}*

As seen in the above example the values of all the dictionaries have been merged into one.

### **Join two dictionaries having few items in common**

**Note**: One thing to note here is that if both the dictionaries have a common key then the first dictionary value will be overridden with the second dictionary value.

**Example**

dict1 = {'Jessa': 70, 'Arul': 80, 'Emma': 55}  
dict2 = {'Kelly': 68, 'Harry': 50, 'Emma': 66}  
  
*# join two dictionaries with some common items*dict1.update(dict2)  
*# printing the updated dictionary*print(dict1['Emma'])  
*# Output 66*

As mentioned in the case of the same key in two dictionaries the latest one will override the old one.

In the above example, both the dictionaries have the key ‘Emma’ So the value of the second dictionary is used in the final merged dictionary dict1.

## **Copy a Dictionary**

We can create a copy of a dictionary using the following two ways

* Using copy() method.
* Using the dict() constructor

dict1 = {'Jessa': 70, 'Emma': 55}  
  
*# Copy dictionary using copy() method*dict2 = dict1.copy()  
*# printing the new dictionary*print(dict2)  
*# output {'Jessa': 70, 'Emma': 55}  
  
# Copy dictionary using dict() constructor*dict3 = dict(dict1)  
print(dict3)  
*# output {'Jessa': 70, 'Emma': 55}  
  
# Copy dictionary using the output of items() methods*dict4 = dict(dict1.items())  
print(dict4)  
*# output {'Jessa': 70, 'Emma': 55}*

### Copy using the assignment operator

We can simply use the '=' operator to create a copy.

Note: When you set dict2 = dict1, you are making them refer to the same dict object, so when you modify one of them, all references associated with that object reflect the current state of the object. So don’t use the assignment operator to copy the dictionary instead use the copy() method.

**Example**

dict1 = {'Jessa': 70, 'Emma': 55}  
  
*# Copy dictionary using assignment = operator*dict2 = dict1  
*# modify dict2*dict2.update({'Jessa': 90})  
print(dict2)  
*# Output {'Jessa': 90, 'Emma': 55}*print(dict1)  
*# Output {'Jessa': 90, 'Emma': 55}*

## **Nested dictionary**

Nested dictionaries are dictionaries that have one or more dictionaries as their members. It is a collection of many dictionaries in one dictionary.

Let us see an example of creating a nested dictionary ‘Address’ inside a ‘person’ dictionary.

**Example**

*# address dictionary to store person address*address = {"state": "Texas", 'city': 'Houston'}  
  
*# dictionary to store person details with address as a nested dictionary*person = {'name': 'Jessa', 'company': 'Google', 'address': address}  
  
*# Display dictionary*print("person:", person)  
  
*# Get nested dictionary key 'city'*print("City:", person['address']['city'])  
  
*# Iterating outer dictionary*print("Person details")  
for key, value in person.items():  
 if key == 'address':  
 *# Iterating through nested dictionary* print("Person Address")  
 for nested\_key, nested\_value in value.items():  
 print(nested\_key, ':', nested\_value)  
 else:  
 print(key, ':', value)

Output

person: {'name': 'Jessa', 'company': 'Google', 'address': {'state': 'Texas', 'city': 'Houston'}}

City: Houston

Person details

name: Jessa

company: Google

Person Address

state: Texas

city : Houston

As we can see in the output we have added one dictionary inside another dictionary.

### **Add multiple dictionaries inside a single dictionary**

Let us see an example of creating multiple nested 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.

**Example**

*# each dictionary will store data of a single student*jessa = {'name': 'Jessa', 'state': 'Texas', 'city': 'Houston', 'marks': 75}  
emma = {'name': 'Emma', 'state': 'Texas', 'city': 'Dallas', 'marks': 60}  
kelly = {'name': 'Kelly', 'state': 'Texas', 'city': 'Austin', 'marks': 85}  
  
*# Outer dictionary to store all student dictionaries (nested dictionaries)*class\_six = {'student1': jessa, 'student2': emma, 'student3': kelly}  
  
*# 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')

**Output**

Student 3 name: Kelly

Student 3 marks: 85

Class details

student1

name: Jessa

state: Texas

city: Houston

marks: 75

student2

name: Emma

state: Texas

city: Dallas

marks: 60

student3

name: Kelly

state: Texas

city: Austin

marks : 85

As seen in the above example, we are adding three individual dictionaries inside a single dictionary.

## **Sort dictionary**

The built-in method sorted() will sort the keys in the dictionary and returns a sorted list. In case we want to sort the values we can first get the values using the values() and then sort them.

**Example**

dict1 = {'c': 45, 'b': 95, 'a': 35}  
  
*# sorting dictionary by keys*print(sorted(dict1.items()))  
*# Output [('a', 35), ('b', 95), ('c', 45)]  
  
# sort dict eys*print(sorted(dict1))  
*# output ['a', 'b', 'c']  
  
# sort dictionary values*print(sorted(dict1.values()))  
*# output [35, 45, 95]*

As we can see in the above example the keys are sorted in the first function call and in the second the values are sorted after the values() method call.

## **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. The general syntax is as follows

output\_dictionary = {key : value **for** key,value **in** iterable [**if** key,value condition1]}

Let us see this with a few examples.

*# 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)  
  
*# output {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.

telephone\_book = [1178, 4020, 5786]

persons = ['Jessa', 'Emma', 'Kelly']

telephone\_Directory = {key: value **for** key, value **in** **zip**(persons, telephone\_book)}

**print**(telephone\_Directory)

# Output {'Jessa': 1178, 'Emma': 4020, 'Kelly': 5786}

In the above example, 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.

**dict** = {1:'aaa',2:'bbb',3:'AAA'}

**print**('Maximum Key',**max**(**dict**)) # 3

**print**('Minimum Key',**min**(**dict**)) # 1

[**Run**](https://pynative.com/online-python-code-editor-to-execute-python-code/)

As we can see max() and min() considered the keys to finding the maximum and minimum values respectively.

### 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.

*#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))

**OUTPUT:**

**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. Let us see what any() method will return for the above cases.

*#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))

**OUTPUT:**

**All True Keys:: True**

**One False Key :: True**

**Empty Dictionary :: False**

**With 0 in single quotes :: True**

**all false :: False**

As we can see this method returns true even if there is one true value and one thing to note here is that it **returns false for empty dictionary** and all false dictionary.

### **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**

**Below is the summary** of all operations that we learned in this lesson

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 |