Dictionary

Dictionaries are used to store data values in key: value pairs.

A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

Dictionaries are written with curly brackets, and have keys and values:

How to createDictionaries and print:

Ex:

thisdict = {  
 "brand": "us polo",  
 "model": "jackson",  
 "year": 1964  
}  
print(thisdict)

outpu:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

## Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key: value pairs, and can be referred to by using the key name.

Ex: Print the "brand" value of the dictionary:

thisdict = {  
 "brand": "us polo",  
 "model": "sekhar",  
 "year": 1964  
}  
print(thisdict["brand"])

output:

us polo

**Ordered or Unordered**

* As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.
* When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
* Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

**Changeable**

* Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

**Duplicates Not Allowed**

Dictionaries cannot have two items with the same key:

Ex: Duplicate values will overwrite existing values:

thisdict = {  
 "brand": "us polo",  
 "model": "jackson",  
 "year": 1964 ,  
 "year":2000  
}  
print(thisdict)

output:

{'brand': 'us polo', 'model': 'jackson', 'year': 2000}

## Dictionary Length :

To determine how many items a dictionary has, use the len() function:

Ex: Print the number of items in the dictionary

thisdict = {  
 "brand": "netplay",  
 "model": "jackson",  
 "year": 1964,  
 "year": 2020  
}  
print(len(thisdict))

output:

3

## Dictionary Items - Data Types

The values in dictionary items can be of any data type:

Ex : String, int, Boolean, and list data types

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

output:

{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}

## type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

Ex : Print the data type of a dictionary

thisdict = {  
 "brand": "mytriojas",  
 "model": "lakshmi",  
 "year": 1964  
}  
print(type(thisdict))

output:

<class 'dict'>

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* [List](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [Tuple](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [Set](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered and changeable. No duplicate members.
* When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Ex : Get the value of the "model" key

thisdict = {  
 "brand": "us polo",  
 "model": "sekhar",  
 "year": 1964  
}  
x = thisdict["model"]  
print("model name:",x)

output:

model name: sekhar

## Get Keys:

* **The keys() method will return a list of all the keys in the dictionary**.

**Ex** :

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

output:

dict\_keys(['brand', 'model', 'year'])

Add a new item to the original dictionary, and see that the keys list gets updated as well.

Ex:

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

print(x) #before the change  
  
car["color"] = "white"  
  
print(x) #after the change

output:

dict\_keys(['brand', 'model', 'year'])

dict\_keys(['brand', 'model', 'year', 'color'])

## Get Values :

* The values() method will return a list of all the values in the dictionary

Ex:

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

output:

dict\_values(['Ford', 'Mustang', 1964])

The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

Ex 1: Make a change in the original dictionary, and see that the values list gets updated as well

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values()   
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

output:

dict\_values(['Ford', 'Mustang', 1964])

dict\_values(['Ford', 'Mustang', 2020])

Ex: Add a new item to the original dictionary, and see that the values list gets updated as well

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values() # update the values  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

output:

dict\_values(['Ford', 'Mustang', 1964])

dict\_values(['Ford', 'Mustang', 1964, 'red'])

## Get Items :

* The items() method will return each item in a dictionary, as tuples in a list.

Ex :

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

output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

Make a change in the original dictionary, and see that the items list gets updated as well

Ex:  
car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

Output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2020)])

Add a new item to the original dictionary, and see that the items list gets updated as well:

Ex :  
car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

Output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])

Check if "model" is present in the dictionary:

Ex :  
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

# **Python - Change Dictionary Items**

## Change Values :

You can change the value of a specific item by referring to its key name:

Change the "year" to 2018

Ex:

thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
thisdict["year"] = 2018  
  
print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

## Update Dictionary

* The update() method will update the dictionary with the items from the given argument.
* The argument must be a dictionary, or an iterable object with key: value pairs.

Update the "year" of the car by using the update() method:  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
thisdict. Update({"year": 2020})  
  
print(thisdict)

Output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

EX: Add a color item to the dictionary by using the update() method:

thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
thisdict. Update({"color": "red"})  
  
print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

# Python - **Remove Dictionary Items :**

* There are several methods to remove items from a dictionary
* The pop() method removes the item with the specified key name.

Ex :

thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

output:

{'brand': 'Ford', 'year': 1964}

#The del keyword removes the item with the specified key name  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

Output:

{'brand': 'Ford', 'year': 1964}

The del keyword can also delete the dictionary completely  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

Output:

Traceback (most recent call last):

File "C:/Users/sekha/PycharmProjects/dict1/delcomplete dict.py", line 8, in <module>

print(thisdict) #this will cause an error because "thisdict" no longer exists.

NameError: name 'thisdict' is not defined

**Clear ()**:

#The clear() method empties the dictionary  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
thisdict.clear()  
print(thisdict)

Output:

{}

## Loop Through a Dictionary :

* You can loop through a dictionary by using a **for** loop.
* When looping through a dictionary, the return value are the ***keys*** of the dictionary, but there are methods to return the ***values*** as well.

#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

#You can also use the values() method to return values of a dictionar  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
for x in thisdict.values():  
 print(x)

Output:

Ford

Mustang

1964

#You 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

#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

## Copy a Dictionary :

* You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.
* There are ways to make a copy, one way is to use the built-in Dictionary method copy().

#Make a copy of a dictionary with the copy() method  
thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Another way to make a copy is to use the built-in function dict() :

Ex:

thisdict = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

## Nested Dictionaries :

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

Ex :Create a dictionary that contain three 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}}

Ex: Create three dictionaries, then create one dictionary that will contain the other three dictionaries

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}}

**Methods**

**1.Clear():**

Definition and Usage

* The clear() method removes all the elements from a dictionary.

Syntax :

dictionary. Clear()

Parameter Values

No parameters

Ex:

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

output:

{}

**2.Copy()**:

Definition and Usage

* The copy() method returns a copy of the specified dictionary.

**Syntax:**

dictionary. Copy()

Parameter Values

No parameters

Ex:

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

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

3.Fromkeys ():

Definition and Usage

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

**Syntax :**

dict.fromkeys(keys, value)

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| keys | Required. An iterable specifying the keys of the new dictionary |
| value | Optional. The value for all keys. Default value is None |

Ex:

x = ('key1', 'key2', 'key3')  
y = 0  
  
thisdict = dict.fromkeys(x, y)  
print(thisdict)

output:

{'key1': 0, 'key2': 0, 'key3': 0}

Ex1:

Same example as above, but without specifying the value  
x = ('key1', 'key2', 'key3')  
  
thisdict = dict.fromkeys(x)  
  
print(thisdict)

Output:

{'key1': None, 'key2': None, 'key3': None}

**4.Get()**:

Definition and Usage

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

**Syntax :**

dictionary.get(key name, value)

Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| Keyname | Required. The keyname of the item you want to return the value from |
| Value | Optional. A value to return if the specified key does not exist. Default value None |

Ex:

#Get the value of the "model" item:  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.get("model")  
  
print(x)

Output:

Mustang

**5.Items()**:

Definition and Usage

* The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.
* The view object will reflect any changes done to the dictionary, see example below.

**Syntax:**

dictionary. Items()

Parameter Values

No parameters

#Return the dictionary's key-value pairs:  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.items()  
  
print(x)

Output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

EX1:

#When an item in the dictionary changes value, the view object also gets updated  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.items()  
  
car["year"] = 2018  
  
print(x)

Output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2018)])

**6.keys()**:

Definition and Usage

* The keys() method returns a view object. The view object contains the keys of the dictionary, as a list.
* The view object will reflect any changes done to the dictionary, see example below.

**Syntax :**

dictionary.keys()

Parameter Values

No parameters

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

Output:

dict\_keys(['brand', 'model', 'year'])

Ex1:

#When an item is added in the dictionary, the view object also gets updated:  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.keys()  
  
car["color"] = "white"  
  
print(x)

Output:

dict\_keys(['brand', 'model', 'year', 'color'])

## 7.pop():-

## Definition and Usage

* The pop() method removes the specified item from the dictionary.
* The value of the removed item is the return value of the pop() method, see example below.

## Syntax :

**dictionary. Pop(key name, default value)**

## Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Keyname* | Required. The keyname of the item you want to remove |
| *Defaultvalue* | Optional. A value to return if the specified key do not exist.  If this parameter is not specified, and the no item with the specified key is found, an error is raised |

#Remove "model" from the dictionary:  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
car.pop("model")  
  
print(car)

Output:

{'brand': 'Ford', 'year': 1964}

#The value of the removed item is the return value of the pop() method  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.pop("model")  
  
print(x)

Output:

Mustang

## 8.Popitem():

## Definition and Usage

* The popitem() method removes the item that was last inserted into the dictionary. In versions before 3.7, the popitem() method removes a random item.
* The removed item is the return value of the popitem() method, as a tuple, see example below.

## Syntax:

dictionary. Popitem()

## Parameter Values

No parameters

#Remove the last item from the dictionary  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
car.popitem()  
  
print(car)

Output:

{'brand': 'Ford', 'model': 'Mustang'}

#The removed item is the return value of the pop() method  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
  
x = car.popitem()  
  
print(x)

Output:

('year', 1964)

## 9.Update():

## Definition and Usage

* The update() method inserts the specified items to the dictionary.
* The specified items can be a dictionary, or an iterable object with key value pairs.

## Syntax :

dictionary. Update(iterable)

## Parameter Values

|  |  |
| --- | --- |
| Parameter | Description |
| *Iterable* | A dictionary or an iterable object with key value pairs,  that will be inserted to the dictionary |

#Insert an item to the dictionary  
car = {  
 "brand": "uspolo",  
 "model": "BBW",  
 "year": 1964  
}  
  
car.update({"color": "REd"})  
  
print(car)

Output:

{'brand': 'uspolo', 'model': 'BBW', 'year': 1964, 'color': 'REd'}

## 10.Values():

## Definition and Usage

* The values() method returns a view object. The view object contains the values of the dictionary, as a list.
* The view object will reflect any changes done to the dictionary, see example below.

## Syntax:

dictionary. Values()

## Parameter Values

No parameters

#When a values is changed in the dictionary, the view object also gets updated  
car = {  
 "brand": "Ford",  
 "model": "Mustang",  
 "year": 1964  
}  
x = car.values()  
car["year"] = 2018  
print(x)

Output:

dict\_values(['Ford', 'Mustang', 2018])