# Python字典dictionary

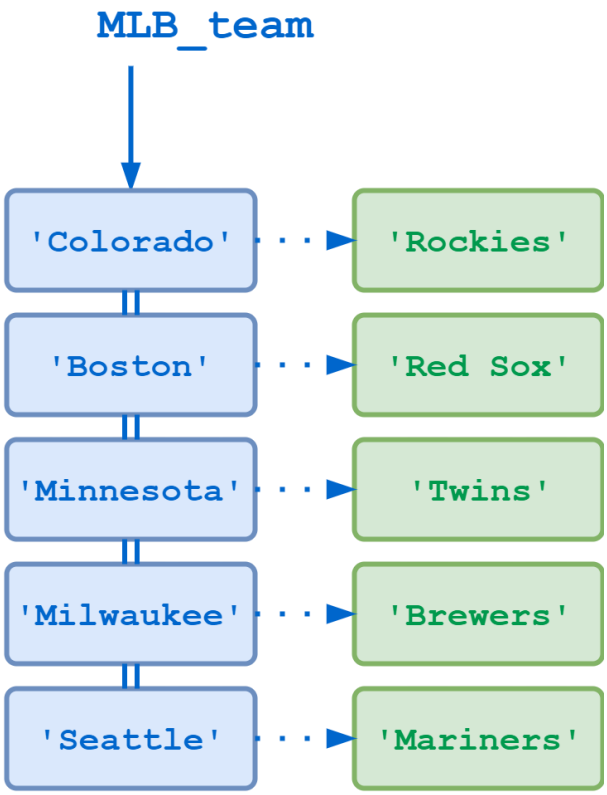
Dictionaries in Python

python-dictionary

python3 字典(dict)基础

Python3 字典

```
d = {
    <key>: <value>,
    <key>: <value>,
      .
      .
      .
    <key>: <value>
}
```




创建字典 `{ }`

通过元组元素列表

```
MLB_team = dict([
               ('Colorado', 'Rockies'),
               ('Boston', 'Red Sox'),
               ('Minnesota', 'Twins'),
               ('Milwaukee', 'Brewers'),
               ('Seattle', 'Mariners')])
print(MLB_team)
```

```
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee': 'Brewers',
'Seattle': 'Mariners'}
```

`zip()` 打包 `list`

```
a = [1,2,3,4,5,6]
b = ['a','b','c','d']
print(list(zip(a,b)))
```

```
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')]
```

赋值法

```
MLB_team = dict(
    Colorado='Rockies',
    Boston='Red Sox',
    Minnesota='Twins',
    Milwaukee='Brewers',
    Seattle='Mariners')

print(MLB_team)
```

```
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Twins', 'Milwaukee': 'Brewers',
'Seattle': 'Mariners'}
```

```
print(type(MLB_team))
```

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

```
# Define a blank dictionary with no elements
blank_dict = {}

# Define a dictionary with numeric keys
num_dict = {1: 'soccer', 2: 'baseball'}

# Define a dictionary with keys having different types
misc_dict = {'class': 'senior secondary', 1: [1, 2, 3,4,5]}

# Create a dictionary with dict() method
get_dict_from_func = dict({1:'veg', 2:'non-veg'})
```

```
# Create a dictionary from a sequence of tuples
make_dict_from_seq = dict([(1,'jake'), (2,'john')])
```

```
dict = {'Student Name': 'Berry', 'Roll No.': 12, 'Subject': 'English'}
print("dict['Student Name']: ", dict['Student Name'])
print("dict['Roll No.']: ", dict['Roll No.'])
print("dict['Subject']: ", dict['Subject'])
```

```
dict['Student Name']:  Berry
dict['Roll No.']:  12
dict['Subject']:  English
```

```
dict = {'Student Name': 'Berry', 'Roll No.': 12, 'Subject': 'English'}
print("The student who left:", dict.get('Student Name'))
dict['Student Name'] = 'Larry'
print("The student who replaced: [Update]", dict.get('Student Name'))
dict['Student Name'] = 'Jarry'
print("The student who joined: [Addition]", dict.get('Student Name'))
```

```
The student who left: Berry
The student who replaced: [Update] Larry
The student who joined: [Addition] Jarry
```

## 字典的键 `key`

在Python中，几乎任何类型的值都可以用作字典键。

```
foo = {42: 'aaa', 2.78: 'bbb', True: 'ccc'}
print(foo)
```

```
{42: 'aaa', 2.78: 'bbb', True: 'ccc'}
```

甚至可以使用内置对象，如类型和函数:

```
d = {int: 1, float: 2, bool: 3}
print(d)
print(d[float])

d = {bin: 1, hex: 2, oct: 3}
print(d[oct])
```

```
{<class 'int'>: 1, <class 'float'>: 2, <class 'bool'>: 3}
2
3
```

然而，字典键必须遵守一些限制。

首先，给定的键只能在字典中出现一次。不允许重复钥匙。字典将每个键映射到一个对应的值，因此多次映射特定键没有意义。
在上面看到，当将一个值赋给一个已经存在的dictionary键时，它不会第二次添加该键，而是替换现有的值：

```
MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'
}

MLB_team['Minnesota'] = 'Timberwolves'
print(MLB_team)
```

```
{'Colorado': 'Rockies', 'Boston': 'Red Sox', 'Minnesota': 'Timberwolves', 'Milwaukee':
'Brewers', 'Seattle': 'Mariners'}
```

其次，dictionary键的类型必须是不可变的。元组也可以是字典键，因为元组是不可变的:

```
d = {(1, 1): 'a', (1, 2): 'b', (2, 1): 'c', (2, 2): 'd'}
print(d[(1,1)])
print(d[(2,1)])
```

```
a
c
```

类似地，如果在初始创建字典时第二次指定一个键，第二次出现的值将覆盖第一次出现的值

`list` 和其他 `dictionary` 都不能作为 `dictionary` 键，因为 `list` 和 `dictionary` 是可变的

```
d = {[1, 1]: 'a', [1, 2]: 'b', [2, 1]: 'c', [2, 2]: 'd'}
print(d)
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-37-b50a34bc2c48> in <module>()
----> 1 d = {[1, 1]: 'a', [1, 2]: 'b', [2, 1]: 'c', [2, 2]: 'd'}
      2 print(d)
```

```
TypeError: unhashable type: 'list'
```

技术说明:为什么错误消息说“unhashable”？
从技术上讲，说一个对象必须是不可变的才能用作字典键是不太正确的。更准确地说，对象必须是可hashable的，这意味着它可以传递给散列函数。 哈希函数获取任意大小的数据，并将其映射到一个相对简单的固定大小值，称为哈希值(或简单的哈希)，用于表查找和比较。

Python内置的hash()函数返回一个可hashable对象的hash值，并为一个不可hashable对象抛出异常:

```
print(hash('foo'))
print(hash([1, 2, 3]))
```

```
1857151007243512327
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-38-dc2b5a49f1e8> in <module>()
      1 print(hash('foo'))
----> 2 print(hash([1, 2, 3]))
```

```
TypeError: unhashable type: 'list'
```

## 字典值的限制

相反，字典值没有限制。完全没有。dictionary值可以是Python支持的任何类型的对象，包括list和dictionary之类的可变类型，以及用户定义的对象

```
d = {0: 'a', 1: 'a', 2: 'a', 3: 'a'}
print(d)

print(d[0] == d[1] == d[2])
```

```
{0: 'a', 1: 'a', 2: 'a', 3: 'a'}
True
```

## 运算符和内置函数

许多可以与字符串、列表和元组一起使用的操作符和内置函数。其中一些也适用于字典。例如，in和not in操作符根据指定的操作数是否作为键出现在字典中，返回True或False

```
MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'}

print('Milwaukee' in MLB_team)
print('Toronto' in MLB_team)
print('Toronto' not in MLB_team)
```

```
True
False
True
```

可以使用in操作符和短路评估，以避免在试图访问一个不在字典中的键时引发错误:

```
print(MLB_team['Toronto'])
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-42-01952ce60be2> in <module>()
----> 1 print(MLB_team['Toronto'])
```

```
KeyError: 'Toronto'
```

```
print('Toronto' in MLB_team and MLB_team['Toronto'])
```

```
False
```

在第二种情况下，由于短路求值，表达式MLB_team['Toronto']没有求值，因此不会发生KeyError异常。 `len()` 函数的作用是:返回字典中键值对的个数:

```
MLB_team = {
    'Colorado' : 'Rockies',
    'Boston'   : 'Red Sox',
    'Minnesota': 'Twins',
    'Milwaukee': 'Brewers',
    'Seattle'  : 'Mariners'
}
print(len(MLB_team))
```

```
5
```

## 内置函数

与字符串和列表一样，可以在字典上调用几个内置方法。事实上，在某些情况下，list和dictionary方法共享相同的名称。

`d.clear()` 清空字典

```
d = {'a': 10, 'b': 20, 'c': 30}
print(d)

d.clear()
print(d)
```

```
{'a': 10, 'b': 20, 'c': 30}
{}
```

获取字典值 `d.get(<key>[, <default>])`

如果没有就返回None

```
d = {'a': 10, 'b': 20, 'c': 30}

print(d.get('b'))
print(d.get('z'))
```

```
20
None
```

如果没有找到，并且指定了可选的参数，则返回该值，而不是None:

```
print(d.get('z', -1))
```

```
-1
```

`d.items()` 返回字典中的键值对列表。

items()返回包含d中的键值对的元组列表，每个元组中的第一项是键，第二项是键的值:

```
d = {'a': 10, 'b': 20, 'c': 30}
print(d)

print(list(d.items()))
print(list(d.items())[1][0])
print(list(d.items())[1][1])
```

```
{'a': 10, 'b': 20, 'c': 30}
[('a', 10), ('b', 20), ('c', 30)]
b
20
```

`d.keys()` 返回字典中的键列表

```
d = {'a': 10, 'b': 20, 'c': 30}
print(d)

print(list(d.keys()))
```

```
{'a': 10, 'b': 20, 'c': 30}
['a', 'b', 'c']
```

`d.values()` 返回字典中的值列表

```
d = {'a': 10, 'b': 20, 'c': 30}
print(d)

print(list(d.values()))
```

```
{'a': 10, 'b': 20, 'c': 30}
[10, 20, 30]
```

```
d = {'a': 10, 'b': 10, 'c': 10}
print(d)

print(list(d.values()))
```

```
{'a': 10, 'b': 10, 'c': 10}
[10, 10, 10]
```

技术说明:.items()、.keys()和.values()方法实际上返回的是视图对象。dictionary视图对象或多或少类似于键和值的窗口。出于实用目的，可以将这些方法视为返回字典键和值的列表。

`d.pop(<key>[, <default>])` 从字典中删除键(如果存在)，并返回其值。

如果d中存在，则d.pop()删除并返回其关联值:

```
d = {'a': 10, 'b': 20, 'c': 30}
d.pop('b')

print(d)
```

```
{'a': 10, 'c': 30}
```

d.pop()如果不在d中，则会引发KeyError异常:

```
d = {'a': 10, 'b': 20, 'c': 30}

print(d.pop('z'))
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-53-f4ea6cf177e0> in <module>()
      1 d = {'a': 10, 'b': 20, 'c': 30}
      2
----> 3 print(d.pop('z'))
```

```
KeyError: 'z'
```

`d.popitem()` 从字典中删除键值对

popitem()从d中删除一个随机的任意键值对，并以元组的形式返回:

```
d = {'a': 10, 'b': 20, 'c': 30}
d.popitem()
print(d)

d.popitem()
print(d)
```

```
{'a': 10, 'b': 20}
{'a': 10}
```

如果d为空，d.popitem()会引发一个KeyError异常:

```
d = {}
print(d.popitem())
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-55-b63a888cdf9f> in <module>()
      1 d = {}
----> 2 print(d.popitem())
```

```
KeyError: 'popitem(): dictionary is empty'
```

`d.update(<obj>)` 将字典与另一个字典或键值对的迭代器合并

如果是一个字典，d.update()将来自的条目合并为d。对于中的每个键:

```
如果键不在d中，则将中的键值对添加到d中。
如果键已经出现在d中，那么该键在d中的对应值将更新。
```

```
d1 = {'a': 10, 'b': 20, 'c': 30}
d2 = {'b': 200, 'd': 400}

d1.update(d2)
print(d1)
```

```
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

```
d1 = {'a': 10, 'b': 20, 'c': 30}
d1.update([('b', 200), ('d', 400)])
print(d1)
```

```
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

要合并的值可以指定为关键字参数列表:

```
d1 = {'a': 10, 'b': 20, 'c': 30}
d1.update(b=200, d=400)
print(d1)
```

```
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

## 删除元素

Delete/Remove Elements From A Python Dictionary

```
# Create a Python dictionary
sixMonths = {1:31, 2:28, 3:31, 4:30, 5:31, 6:30}

# Delete a specific element
print(sixMonths.pop(6))
print(sixMonths)

# Delete an random element
print(sixMonths.popitem())
print(sixMonths)

# Remove a specific element
del sixMonths[5]
print(sixMonths)

# Delete all elements from the dictionary
sixMonths.clear()
print(sixMonths)

# Finally, eliminate the dictionary object
del sixMonths
print(sixMonths)
```

```
30
{1: 31, 2: 28, 3: 31, 4: 30, 5: 31}
(5, 31)
{1: 31, 2: 28, 3: 31, 4: 30}
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-4-52423233a7e2> in <module>
     11
     12 # Remove a specific element
---> 13 del sixMonths[5]
     14 print(sixMonths)
     15
```

```
KeyError: 5
```

```
dict = {'Name': 'Runoob', 'Age': 7, 'Class': 'First'}

del dict['Name'] # 删除键 'Name'
dict.clear()     # 清空字典
del dict         # 删除字典

print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-76-a6e66fa9296e> in <module>()
      5 del dict         # 删除字典
      6
----> 7 print ("dict['Age']: ", dict['Age'])
      8 print ("dict['School']: ", dict['School'])
```

```
TypeError: 'type' object is not subscriptable
```

迭代

```
dict = {'Student Name': 'Berry', 'Roll No.': 12, 'Subject': 'English'}
print("Keys are:")
for key in dict:
    print(key)
```

```
Keys are:
Student Name
Roll No.
Subject
```

```
dict = {'Student Name': 'Berry', 'Roll No.': 12, 'Subject': 'English'}
print("{Keys}:{Values}")
for key, value in dict.items():
    print({key},":",{value})
```

```
{Keys}:{Values}
{'Student Name'} : {'Berry'}
{'Roll No.'} : {12}
{'Subject'} : {'English'}
```

```
{str(iter):iter for iter in [11,22,33,44,55]}
```

```
{'11': 11, '22': 22, '33': 33, '44': 44, '55': 55}
```

```
weekdays = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat']
{w:len(w) for w in weekdays}
```

```
{'sun': 3, 'mon': 3, 'tue': 3, 'wed': 3, 'thu': 3, 'fri': 3, 'sat': 3}
```

```
{w : i for i, w in enumerate(weekdays)}
```

```
{'sun': 0, 'mon': 1, 'tue': 2, 'wed': 3, 'thu': 4, 'fri': 5, 'sat': 6}
```

```
weekdays = {'fri': 5, 'tue': 2, 'wed': 3, 'sat': 6, 'thu': 4, 'mon': 1, 'sun': 0}
# Output: True
print('fri' in weekdays)
# Output: False
print('thu' not in weekdays)
# Output: True
print('mon' in weekdays)
```

```
True
False
True
```

访问字典值

```
# accesing a element from a Dictionary

# Creating a Dictionary
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key
print("Acessing a element using key:")
print(Dict['name'])

# accessing a element using key
print("Acessing a element using key:")
print(Dict[1])

# accessing a element using get()
# method
print("Acessing a element using get:")
print(Dict.get(3))
```

```
print(Dict.get(5))
print(Dict[5])
```

```
Acessing a element using key:
For
Acessing a element using key:
Geeks
Acessing a element using get:
Geeks
None
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-6-0f9163adb99d> in <module>()
     17 print(Dict.get(3))
     18 print(Dict.get(5))
---> 19 print(Dict[5])
```

```
KeyError: 5
```

使用get()访问元素不存在时返回None

如果直接用字典的键key来访问则会报错 `KeyError`

## Python 字典合并merge

```
# Creating a Nested Dictionary
# as shown in the below image
Dict = {1: 'Geeks', 2: 'For',  3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

print(Dict)
```

```
{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
```

使用 `**`

### How To Combine Two Dictionary Variables in Python

```
x = {1: 'Geeks', 2: 'For'}
y = {3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Geeks'}}

print({**x, **y})
```

```
{1: 'Geeks', 2: 'For', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
```

```
myDict1 = {'one': 'Ram', 'two': 13, 'three': 'Jordge'}
myDict2 = {'four': 'Gill', 'five': 33, 'six': 'Steve'}
newDict = {**myDict1, 'seven': 'Billy', 'eight': 'Oman', **myDict2}
print(newDict)
```

```
{'one': 'Ram', 'two': 13, 'three': 'Jordge', 'seven': 'Billy', 'eight': 'Oman', 'four': 'Gill',
'five': 33, 'six': 'Steve'}
```

使用 `update()`

### How to merge two dictionaries in a single expression?

```
x = {'a':1, 'b': 2}
y = {'b':10, 'c': 11}
z = x.update(y)
print(z)
print(x)
```

```
None
{'a': 1, 'b': 10, 'c': 11}
```

从以上可以看出update更新元素并没有返回新的对象，也就是，更新的操作是在原有的对象基础上完成的

同时遇到 `key` 值相同的情况，则只保留最后的一个值

如果想要保持原有数据不变，可以使用 `.copy()` 函数

```
z = x.copy()
z.update(y)
```

### The Update() Method

```
from itertools import chain
from collections import defaultdict
dict1 = {'bookA': 1, 'bookB': 2, 'bookC': 3}
dict2 = {'bookC': 2, 'bookD': 4, 'bookE': 5}
dict3 = defaultdict(list)
for k, v in chain(list(dict1.items()), list(dict2.items())):
    dict3[k].append(v)

for k, v in list(dict3.items()):
    print((k, v))
```

```
('bookA', [1])
('bookB', [2])
('bookC', [3, 2])
('bookD', [4])
('bookE', [5])
```

`.items()`

```
x = {'a':1, 'b': 2}
y = {'b':10, 'c': 11}
z = dict(list(x.items()) + list(y.items()))
print(z)
```

```
{'a': 1, 'b': 10, 'c': 11}
```

Is there any pythonic way to combine two dicts (adding values for keys that appear in both)?

```
A = {'a':1, 'b':2, 'c':3}
B = {'b':3, 'c':4, 'd':5}
c = {x: A.get(x, 0) + B.get(x, 0) for x in set(A).union(B)}
print(c)
```

```
{'b': 5, 'd': 5, 'c': 7, 'a': 1}
```

## 获取具有最大值的Key

Get Key With Maximum Value in Dictionary in Python

使用 `max()` 函数的 `key` 参数

```
myDict = {'one': 24, 'two': 13, 'three': 54, 'four': 9}
MaxDictVal = max(myDict, key=myDict.get)
print(MaxDictVal)
```

```
three
```

## 检查键 `key` 是否存在

How to Check If Key Exists In Dictionary Using Python

```
mykey = 'four';
myDict = {'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill', 'five': 33, 'six':
'Steve'}
for key, value in myDict.items():
    if key == mykey:
        print('The given key already exists')
```

```
The given key already exists
```

```
mykey = 'four';
myNewDict = {"one": "Ram", "two": 43, "three": 10, "four": "Bilal", "five": 13.2, "six":
"Feroz"}
if mykey in myNewDict:
    print('The given key exists')
else:
    print('Not Exists')
```

```
The given key exists
```

删除单一键值

```
myDict = {'one': 'Sally', 'two': 13, 'three': 'Dingra', 'four': 'Lilop'}
myDict.pop('two')
print(myDict)
```

```
{'one': 'Sally', 'three': 'Dingra', 'four': 'Lilop'}
```

`clear()` 清除字典

```
myDict = {'one': 'Sally', 'two': 13, 'three': 'Dingra', 'four': 'Lilop'}
myDict.clear()
print(myDict)
```

```
{}
```

字典的循环

How to Loop Through Dictionary Elements in Python

```
myDict = { "one": "Ram", "two": "Shyam", "three": 10, "fore": "Bilal", "five": 13.2, "six":
"Feroz"}
for key, value in myDict.items():
    print(key)
```

```
one
two
three
fore
five
six
```

字典添加元素

How to Add Element To Key In Dictionary Using Python

直接利用中括号 `[]` 赋值

```
myDict = {'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill'}
myDict['four'] = 'Dev'
print(myDict)
```

```
{'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Dev'}
```

`.update()`

```
myDict = {'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill'};
myDict.update({'five':'Kelly'});
print(myDict)
```

```
{'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill', 'five': 'Kelly'}
```

## 字典元素更新

`.update()`

```
myDict = {'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill'};
myDict.update({'four':'Kelly'});
print(myDict)
```

```
{'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Kelly'}
```

通过赋值

```
myDict = {'one': 'Ram', 'two': 13, 'three': 'Jordge', 'four': 'Gill'};
myDict['three'] = "Royals"
print(myDict)
```

```
{'one': 'Ram', 'two': 13, 'three': 'Royals', 'four': 'Gill'}
```

## 字典的排序

### How to Sort Python Dictionaries by Key or Value

按键对Python字典排序

如果我们想根据字典对象的键对其排序，最简单的方法是使用Python内置的sort方法，该方法将接受任何可迭代的值并返回已排序的值列表(默认情况下按升序排列)。没有用于排序字典的类方法，但是排序方法的工作方式是相同的。

```
# This is the same as calling sorted(numbers.keys())

numbers = {'first': 1, 'second': 2, 'third': 3, 'Fourth': 4}
print(sorted(numbers))
```

```
['Fourth', 'first', 'second', 'third']
```

我们可以看到这个方法给了我们一个按升序排列的键列表，几乎按字母顺序排列，这取决于我们定义的“字母顺序”。还请注意，我们按它的键对它的键列表进行了排序——如果我们想按它的键对它的值列表进行排序，或者按它的值对它的键列表进行排序，我们必须改变使用排序方法的方式。我们来看看排序的不同方面。

根据值对Python字典进行排序

就像我们对键所做的一样，我们可以使用sort对Python字典的值进行排序:

```
# We have to call numbers.values() here
print(sorted(numbers.values()))
```

```
[1, 2, 3, 4]
```

这是默认顺序的值列表，按升序排列。这些都是非常简单的例子，所以现在让我们来研究一些稍微复杂一些的情况，在这些情况下，我们对dict对象进行排序。

值value升序排列

```
x = {'a':5, 'b':3, 'c':7}

print([(l,k) for k,l in sorted([(j,i) for i,j in x.items()])])
```

```
[('b', 3), ('a', 5), ('c', 7)]
```

值value降序排列

```
print([(l,k) for k,l in sorted([(j,i) for i,j in x.items()], reverse=True)])
```

```
[('c', 7), ('a', 5), ('b', 3)]
```

`most_common()` 最常出现的

```
from collections import Counter
x = Counter({'a':5, 'b':3, 'c':7})
print(x.most_common())
```

```
[('c', 7), ('a', 5), ('b', 3)]
```

出现次数最多的两个

```
print(x.most_common(2))
```

```
[('c', 7), ('a', 5)]
```

```
from collections import Counter, OrderedDict

x = Counter({'a':5, 'b':3, 'c':7})
y = OrderedDict(x.most_common())
print(y)
```

```
OrderedDict([('c', 7), ('a', 5), ('b', 3)])
```

使用Python字典自定义排序算法

如果我们简单地给排序方法一个键/值作为参数，它将执行一个简单的排序，但是通过使用它的其他参数(即键和反向)，我们可以让它执行更复杂的排序。

sort的键参数(不要与dictionary的键混淆)允许我们定义在排序项时使用的特定函数，作为迭代器(在dict对象中)。在上面的两个例子中，键和值都是要排序的项和用于比较的项，但是如果我们想使用dict值对dict键排序，那么我们会告诉sort通过它的键参数来完成。如:

```
# Use the __getitem__ method as the key function
print(sorted(numbers, key=numbers.__getitem__))
# In order of sorted values: [1, 2, 3, 4]
```

```
['first', 'second', 'third', 'Fourth']
```

使用这个语句，我们告诉sort来对dict(它的键)进行排序，并通过使用numbers的类方法来检索值来对它们进行排序——本质上，我们告诉它“对于数字中的每个键，使用数字中的对应值进行比较来对其进行排序”。

我们还可以根据键对数字中的值进行排序，但是使用键参数会更加复杂(没有字典方法像列表那样通过使用某个值来返回键)。指数法)。相反，我们可以使用列表理解来保持简单:

```
# Uses the first element of each tuple to compare
print([value for (key, value) in sorted(numbers.items())])
```

```
[4, 1, 2, 3]
```

另一个要考虑的论点是相反的。如果这是真的，那么顺序将被颠倒(降序)，否则如果这是假的，那么顺序将是默认的(升序)，就这么简单。例如，与前面的两种分类一样:

```
print(sorted(numbers, key=numbers.__getitem__, reverse=True))

print([value for (key, value) in sorted(numbers.items(), reverse=True)])
```

```
['Fourth', 'third', 'second', 'first']
[3, 2, 1, 4]
```

这些排序仍然相当简单，但是让我们来看看一些特殊的算法，我们可能会使用字符串或数字排序我们的字典。

使用字符串和数字算法对Python字典进行排序

按字母顺序排序字符串是非常常见的，但是使用已排序可能不会按“适当的”字母顺序对dict的键/值排序，即忽略大小写。为了忽略这种情况，我们可以再次使用key参数和string .lower(或string .upper)方法，以便所有字符串在比较时都是相同的情况:

```
# Won't change the items to be returned, only while sorting
print(sorted(numbers, key=str.lower))
```

```
['first', 'Fourth', 'second', 'third']
```

为了将字符串与数字关联，我们需要一个额外的上下文元素来正确地关联它们。让我们先创建一个新的dict对象:

```
month = dict(one='January',
                 two='February',
                 three='March',
                 four='April',
                 five='May')
print(month)
```

```
{'one': 'January', 'two': 'February', 'three': 'March', 'four': 'April', 'five': 'May'}
```

它只包含字符串键和值，因此如果没有额外的上下文，就无法将月份按正确的顺序排列。为此，我们可以简单地创建另一个字典来将字符串映射到它们的数值，并使用该字典的_getitem__方法来比较我们的month字典中的值:

```
numbermap = {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
print(sorted(month, key=numbermap.__getitem__))
```

```
['one', 'two', 'three', 'four', 'five']
```

正如我们所看到的，它仍然返回了第一个参数(month)的排序列表。为了按顺序返回月份，我们将再次使用列表理解:

```
# Assuming the keys in both dictionaries are EXACTLY the same:
print([month[i] for i in sorted(month, key=numbermap.__getitem__)])
```

```
['January', 'February', 'March', 'April', 'May']
```

如果我们想按每个字符串中重复字母的数量对键/值字符串进行排序，我们可以定义我们自己的自定义方法，用于排序后的键参数:

```
def repeats(string):
    # Lower the case in the string
    string = string.lower()

    # Get a set of the unique letters
    uniques = set(string)

    # Count the max occurrences of each unique letter
    counts = [string.count(letter) for letter in uniques]

    return max(counts)
```

```
print(sorted(month.values(), key=repeats, reverse=True))
```

```
['January', 'February', 'March', 'April', 'May']
```

更高级的排序功能

现在假设我们有一个字典来记录每个月每个班级的学生人数，就像这样:

```
trans = dict(January=33, February=30, March=24, April=0, May=7)
print(trans)
```

```
{'January': 33, 'February': 30, 'March': 24, 'April': 0, 'May': 7}
```

如果我们想先用偶数和奇数来组织类的大小，我们可以这样定义:

```
def evens1st(num):
    # Test with modulus (%) two
    if num == 0:
        return -2
    # It's an even number, return the value
    elif num % 2 == 0:
        return num
    # It's odd, return the negated inverse
    else:
        return -1 * (num ** -1)
```

使用evens1排序函数，得到如下输出:

```
# Max class size first
print(sorted(trans.values(), key=evens1st, reverse=True))
```

```
[30, 24, 33, 7, 0]
```

按照value的值从大到小的顺序来排序

```
dic = {'a':31, 'bc':5, 'c':3, 'asd':4, 'aa':74, 'd':0}
dic2= sorted(dic.items(), key=lambda d:d[1], reverse = True)
print(dic2)
```

```
[('aa', 74), ('a', 31), ('bc', 5), ('asd', 4), ('c', 3), ('d', 0)]
```

```
print(dic.items()) 得到[(键，值)]的列表。
然后用sorted方法，通过key这个参数，指定排序是按照value，也就是第一个元素d[1的值来排序。
reverse = True表示是需要翻转的，默认是从小到大，翻转的话，那就是从大到小。
```

对字典按键（key）排序

```
dic = {'a':31, 'bc':5, 'c':3, 'asd':4, 'aa':74, 'd':0}
dic3= sorted(list(dic.items()), key=lambda d:d[0])
print(dic3)
```

```
[('a', 31), ('aa', 74), ('asd', 4), ('bc', 5), ('c', 3), ('d', 0)]
```

```
a = {6:2,8:0,1:4,-5:6,99:11,4:22}
print(  sorted(a.items()) )
print(  sorted(a.items(),key=lambda x:x[1]) )
print(  sorted(a.items(),key=lambda x:x[0]) )
```

```
[(-5, 6), (1, 4), (4, 22), (6, 2), (8, 0), (99, 11)]
[(8, 0), (6, 2), (1, 4), (-5, 6), (99, 11), (4, 22)]
[(-5, 6), (1, 4), (4, 22), (6, 2), (8, 0), (99, 11)]
```

## dict(或对象)与json之间的互相转化

Python之dict(或对象)与json之间的互相转化

```
loads()：将json数据转化成dict数据
dumps()：将dict数据转化成json数据
load()：读取json文件数据，转成dict数据
dump()：将dict数据转化成json数据后写入json文件
```

dict字典转json数据

```
import json

def dict_to_json():
    dict = {}
    dict['name'] = 'many'
    dict['age'] = 10
    dict['sex'] = 'male'
    print(dict)  # 输出：{'name': 'many', 'age': 10, 'sex': 'male'}
    j = json.dumps(dict)
    print(j)  # 输出：{"name": "many", "age": 10, "sex": "male"}


if __name__ == '__main__':
    dict_to_json()
```

```
{'name': 'many', 'age': 10, 'sex': 'male'}
{"name": "many", "age": 10, "sex": "male"}
```

json数据转成dict字典

```
import json

def json_to_dict():
    j = '{"id": "007", "name": "007", "age": 28, "sex": "male", "phone": "13000000000", "email":
"123@qq.com"}'
    dict = json.loads(s=j)
    print(dict)  # {'id': '007', 'name': '007', 'age': 28, 'sex': 'male', 'phone':
'13000000000', 'email': '123@qq.com'}


if __name__ == '__main__':
    json_to_dict()
```

```
{'id': '007', 'name': '007', 'age': 28, 'sex': 'male', 'phone': '13000000000', 'email':
'123@qq.com'}
```

json的load()与dump()方法的使用

dump()方法的使用

```python
import json

def dict_to_json_write_file():
    dict = {}
    dict['name'] = 'many'
    dict['age'] = 10
    dict['sex'] = 'male'
    print(dict)  # {'name': 'many', 'age': 10, 'sex': 'male'}
    with open('1.json', 'w') as f:
        json.dump(dict, f)  # 会在目录下生成一个1.json的文件，文件内容是dict数据转成的json数据


if __name__ == '__main__':
    dict_to_json_write_file()
```

```
{'name': 'many', 'age': 10, 'sex': 'male'}
```

load()的使用

```python
import json

def json_file_to_dict():
    with open('1.json', 'r') as f:
        dict = json.load(fp=f)
        print(dict)  # {'name': 'many', 'age': 10, 'sex': 'male'}


if __name__ == '__main__':
    json_file_to_dict()
```

```
{'name': 'many', 'age': 10, 'sex': 'male'}
```

## 字符串转字典

How to convert a string to dictionary in Python?

```python
import ast
x = ast.literal_eval("{'foo' : 'bar', 'hello' : 'world'}")
print(x)
print(type(x))
```

```
{'foo': 'bar', 'hello': 'world'}
<class 'dict'>
```

使用json.loads()

注意json加载过程的单引号和双引号

```python
import json
x = json.loads('{"foo": "bar", "hello" : "world"}')
print(x)
print(type(x))
```

```
{'foo': 'bar', 'hello': 'world'}
<class 'dict'>
```

```
x = json.loads("{'foo' : 'bar', 'hello' : 'world'}")
print(x)
print(type(x))
```

```
---------------------------------------------------------------------------

JSONDecodeError                           Traceback (most recent call last)

<ipython-input-90-5b2297aee525> in <module>()
----> 1 x = json.loads("{'foo' : 'bar', 'hello' : 'world'}")
      2 print(x)
      3 print(type(x))
```

```
d:\Anaconda3\lib\json\__init__.py in loads(s, encoding, cls, object_hook, parse_float,
parse_int, parse_constant, object_pairs_hook, **kw)
    352             parse_int is None and parse_float is None and
    353             parse_constant is None and object_pairs_hook is None and not kw):
--> 354         return _default_decoder.decode(s)
    355     if cls is None:
    356         cls = JSONDecoder
```

```
d:\Anaconda3\lib\json\decoder.py in decode(self, s, _w)
    337
    338         """
--> 339         obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    340         end = _w(s, end).end()
    341         if end != len(s):
```

```
d:\Anaconda3\lib\json\decoder.py in raw_decode(self, s, idx)
    353         """
    354         try:
--> 355             obj, end = self.scan_once(s, idx)
    356         except StopIteration as err:
    357             raise JSONDecodeError("Expecting value", s, err.value) from None
```

```
JSONDecodeError: Expecting property name enclosed in double quotes: line 1 column 2 (char 1)
```

## 字典根据value去重

python字典怎么根据value去重复

利用字典键值的唯一性

```
d={'d':0,'b':0,'c':1,'d':1,'e':0}
func = lambda z: dict([(x, y) for y, x in z.items()])
print(d)
print(func(d))
print(func(func(d)))
```

```
{'d': 1, 'b': 0, 'c': 1, 'e': 0}
{1: 'c', 0: 'e'}
{'c': 1, 'e': 0}
```

使用列表推导式过滤字典

```
d = {'a': [1,2,1], 'b': [3,4,1], 'c': [5,6,2]}
print(d)
```

```
{'a': [1, 2, 1], 'b': [3, 4, 1], 'c': [5, 6, 2]}
```

根据值过滤

```
print([x for x in d['b'] if x >2])
```

```
[3, 4]
```

多重条件过滤

```
print(dict(list(zip(d['a'],d['b']))))
```

```
{1: 1, 2: 4}
```

```
print([(x,y) for x, y in zip(d['a'],d['b']) if x == 1 and y > 1])
```

```
[(1, 3)]
```

筛选所有值都大于1的字典

```
print([(k,v) for k,v in d.items() if all(x > 1 for x in v)])
```

```
[('c', [5, 6, 2])]
```

any()函数，如果任何值满足条件，它将返回True。

```
print([(k,v) for k,v in d.items() if any(x > 2 for x in v)])
```

```
[('b', [3, 4, 1]), ('c', [5, 6, 2])]
```

```
from collections import Counter
print([i for i in dictA if Counter(dictA[i])['duck'] > 1])
```

```
['a', 'c', 'e']
```

利用列表推导式筛选数据

```
mylist = [{'a': 1, 'b': 2}, {'a': 3, 'b': 4}, {'a': 5, 'b': 6}]
print(mylist)
```

```
[{'a': 1, 'b': 2}, {'a': 3, 'b': 4}, {'a': 5, 'b': 6}]
```

```
print([i['a'] for i in mylist if 'a' in i if i['a'] > 1])
```

```
[3, 5]
```

统计字典值的个数

count the number of occurrences of a certain value in a dictionary in python?

使用 `collections Counter`

```
from collections import Counter
D = {'a': 97, 'c': 0 , 'b':0,'e': 94, 'r': 97 , 'g':0}

print(Counter(D.values())[0])
```

```
3
```

列表 `count()`

```
D = {'a': 97, 'c': 0 , 'b':0,'e': 94, 'r': 97 , 'g':0}
print(list(D.values()).count(0))
```

```
3
```

遍历法

```
print(sum([1 for i in D.values() if i == 0]))
```

```
3
```

Python count items in dict value that is a list

```
d = {'T1': ['eggs', 'bacon', 'sausage'], 'T2': ['spam', 'ham', 'monty', 'python']}
print(sum(map(len, d.values())))
```

```
7
```

```
dict = {'T1': ['eggs', 'bacon', 'sausage'], 'T2': ['bread', 'butter', 'tosti']}

total = 0

for value in dict:
    value_list = dict[value]
    count = len(value_list)
    total += count

print(total)
```

```
6
```

```
dic ={"Name": [[], [], [(3, 2, 0)], [(3, 1, 0)], [], [], [(4, 3, 2), (4, 3, 0)], [(4, 2, 0)]]}
x = list(map(len, dic.values()))
print(x[0])
```

```
8
```

```
print(dic.values())
```

```
dict_values([[[], [], [(3, 2, 0)], [(3, 1, 0)], [], [], [(4, 3, 2), (4, 3, 0)], [(4, 2, 0)]]])
```

```
print(sum(map(len, list(dic.values())[0])))
```

```
5
```

字典数据追加写入到csv

```
p = [1,2,3]
s = [1,2,3,4,5,6]
w = ['a','b', 'c', 'd', 'e', 'f', 'g', 'h']

import csv
rows = []
```

```
for i in range(len(p)):
    for j in range(len(s)):
        for k in range(len(w)):
            item = {'paragraph': i, 'sentence': j, 'number': k, 'word': w[k]}

            headers = ['paragraph', 'sentence', 'number', 'word']
            rows.append(item)

            # writer = csv.writer(open(' movies_test.csv', 'w', newline='', encoding='utf-8'))
            # fields = headers
            # writer.writerow(fields)

with open("test_01.csv", 'a', newline='') as f:  # newline='' 去除空行，"a"表示追加写入
    f_csv = csv.DictWriter(f, headers)
    f_csv.writeheader()
    f_csv.writerows(rows)

f.close
print('write over')
```

```
write over
```