5 字典

字典 `dictionary` ，在一些编程语言中也称为 `hash` ， `map` ，是一种由键值对组成的数据结构。

顾名思义，我们把键想象成字典中的单词，值想象成词对应的定义，那么——

一个词可以对应一个或者多个定义，但是这些定义只能通过这个词来进行查询。

5.1 基本操作

5.1.1 空字典

Python 使用 `{}` 或者 `dict()` 来创建一个空的字典：

{<键1>:<值1>, <键2>:<值2>, … , <键n>:<值n>}

其中，键和值通过冒号连接，不同键值对通过逗号隔开。

```
a = {}  # 空字典
type(a)
```

```
dict
```

```
a = dict()  # 空字典
type(a)
```

```
dict
```

有了dict之后，可以用索引键值的方法向其中添加元素，也可以通过索引来查看元素的值：

5.1.2 插入键值

```
a["one"] = "this is number 1"
a["two"] = "this is number 2"
```

```
a
```

```
{'one': 'this is number 1', 'two': 'this is number 2'}
```

5.1.3 查看键值

```
a['one']
```

```
'this is number 1'
```

5.1.4 更新键值

```
a["one"] = "this is number 1, too"
a
```

```
{'one': 'this is number 1, too', 'two': 'this is number 2'}
```

5.1.5 使用{}初始化字典

可以看到，Python使用`key: value`这样的结构来表示字典中的元素结构，事实上，可以直接使用这样的结构来初始化一个字典：

```
b = {'two': 'this is number 2', 'one': 'this is number 1'}
b['one']
```

```
'this is number 1'
```

```
Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}

print(Dcountry)
print(Dcountry["中国"])
```

```
{'中国': '北京', '美国': '华盛顿', '法国': '巴黎'}
北京
```

5.1.5 使用 dict 初始化字典

除了通常的定义方式，还可以通过 `dict()` 转化来生成字典：

```
inventory = dict(
    [('foozelator', 123),
     ('frombicator', 18),
     ('spatzleblock', 34),
     ('snitzelhogen', 23)
    ])
inventory
```

```
{'foozelator': 123, 'frombicator': 18, 'snitzelhogen': 23, 'spatzleblock': 34}
```

5.2 字典没有顺序

当我们 `print` 一个字典时，**Python**并不一定按照插入键值的先后顺序进行显示,因为字典中的键本身不一定是有序的。

```
print(a)
```

```
{'one': 'this is number 1, too', 'two': 'this is number 2'}
```

```
print(b)
```

```
{'two': 'this is number 2', 'one': 'this is number 1'}
```

因此，**Python**中不能用支持用数字索引按顺序查看字典中的值，而且数字本身也有可能成为键值，这样会引起混淆：

```
# 会报错
a[0]
```

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

KeyError                                  Traceback (most recent call last)

<ipython-input-16-cc39af2a359c> in <module>()
      1 # 会报错
----> 2 a[0]


KeyError: 0
```

5.3 键必须是不可变的类型

出于hash的目的，Python中要求这些键值对的**键**必须是**不可变**的，而值可以是任意的Python对象。

一个表示近义词的字典：

```
synonyms = {}
synonyms['mutable'] = ['changeable', 'variable', 'varying', 'fluctuating',
                       'shifting', 'inconsistent', 'unpredictable', 'inconstant',
                       'fickle', 'uneven', 'unstable', 'protean']
synonyms['immutable'] = ['fixed', 'set', 'rigid', 'inflexible',
                         'permanent', 'established', 'carved in stone']
synonyms
```

```
{'immutable': ['fixed',
  'set',
  'rigid',
  'inflexible',
  'permanent',
  'established',
  'carved in stone'],
 'mutable': ['changeable',
  'variable',
  'varying',
  'fluctuating',
  'shifting',
  'inconsistent',
  'unpredictable',
  'inconstant',
```

```
  'fickle',
  'uneven',
  'unstable',
  'protean']}
```

另一个例子：

```
# 定义四个字典
e1 = {'mag': 0.05, 'width': 20}
e2 = {'mag': 0.04, 'width': 25}
e3 = {'mag': 0.05, 'width': 80}
e4 = {'mag': 0.03, 'width': 30}
# 以字典作为值传入新的字典
events = {500: e1, 760: e2, 3001: e3, 4180: e4}
events
```

```
{500: {'mag': 0.05, 'width': 20},
 760: {'mag': 0.04, 'width': 25},
 3001: {'mag': 0.05, 'width': 80},
 4180: {'mag': 0.03, 'width': 30}}
```

键（或者值）的数据类型可以不同：

```
people = [
    {'first': 'Sam', 'last': 'Malone', 'name': 35},
    {'first': 'Woody', 'last': 'Boyd', 'name': 21},
    {'first': 'Norm', 'last': 'Peterson', 'name': 34},
    {'first': 'Diane', 'last': 'Chambers', 'name': 33}
]
people
```

```
[{'first': 'Sam', 'last': 'Malone', 'name': 35},
 {'first': 'Woody', 'last': 'Boyd', 'name': 21},
 {'first': 'Norm', 'last': 'Peterson', 'name': 34},
 {'first': 'Diane', 'last': 'Chambers', 'name': 33}]
```

5.4 适合做键的类型

5.4.1 整型和字符串

在不可变类型中，整数和字符串是字典中最常用的类型；而浮点数通常不推荐用来做键，原因如下：

```
data = {}
data[1.1 + 2.2] = 6.6
# 会报错
data[3.3]
```

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

KeyError                                  Traceback (most recent call last)

<ipython-input-3-a48e87d01daa> in <module>()
      2 data[1.1 + 2.2] = 6.6
      3 # 会报错
----> 4 data[3.3]


KeyError: 3.3
```

5.4.2 元组作为键

事实上，观察`data`的值就会发现，这个错误是由浮点数的精度问题所引起的：

```
data
```

```
{3.3000000000000003: 6.6}
```

有时候，也可以使用元组作为键值，例如，可以用元组做键来表示从第一个城市飞往第二个城市航班数的多少：

```
connections = {}
connections[('New York', 'Seattle')] = 100
connections[('Austin', 'New York')] = 200
connections[('New York', 'Austin')] = 400
```

元组是有序的，因此`('New York', 'Austin')`和`('Austin', 'New York')`是两个不同的键：

```
print(connections[('Austin', 'New York')])
```

```
print(connections[('New York', 'Austin')])
```

```
200
400
```

5.5 不能充当键的类型

浮点数,列表list,字典dict,集合set

5.6 字典方法

5.6.1 `get` 方法

之前已经见过，用索引可以找到一个键对应的值，但是当字典中没有这个键的时候，Python会报错，这时候可以使用字典的 `get` 方法来处理这种情况，其用法如下：

```
`d.get(key, default = None)`
```

返回字典中键 `key` 对应的值，如果没有这个键，返回 `default` 指定的值（默认是 `None` ）。

```
a = {}
a["one"] = "this is number 1"
a["two"] = "this is number 2"
```

索引不存在的键值会报错：

```
a["three"]
```

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

KeyError                                  Traceback (most recent call last)

<ipython-input-8-8a5f2913f00e> in <module>()
----> 1 a["three"]


KeyError: 'three'
```

改用get方法：

```
print(a.get("three"))
```

```
None
```

指定默认值参数：

```
a.get("three", "undefined")
```

```
'undefined'
```

5.6.2 `pop` 方法删除元素

`pop` 方法可以用来弹出字典中某个键对应的值，同时也可以指定默认参数：

```
`d.pop(key, default = None)`
```

删除并返回字典中键 `key` 对应的值，如果没有这个键，返回 `default` 指定的值（默认是 `None` ）。

```
a
```

```
{'one': 'this is number 1'}
```

弹出并返回值：

```
a.pop("two")
```

```
'this is number 2'
```

```
a
```

```
{'one': 'this is number 1'}
```

弹出不存在的键值：

```
a.pop("two", 'not exist')
```

```
'not exist'
```

与列表一样，`del` 函数可以用来删除字典中特定的键值对，例如：

```
del a["one"]
a
```

```
{}
```

5.6.3 `update` 方法更新字典

之前已经知道，可以通过索引来插入、修改单个键值对，但是如果想对多个键值对进行操作，这种方法就显得比较麻烦，好在有 `update` 方法：

```
`d.update(newd)`
```

将字典 `newd` 中的内容更新到 `d` 中去。

```
person = {}
person['first'] = "Jmes"
person['last'] = "Maxwell"
person['born'] = 1831
print(person)
```

```
{'first': 'Jmes', 'last': 'Maxwell', 'born': 1831}
```

把'first'改成'James'，同时插入'middle'的值'Clerk'：

```
person_modifications = {'first': 'James', 'middle': 'Clerk'}
```

```
person.update(person_modifications)
print(person)
```

```
{'first': 'James', 'last': 'Maxwell', 'born': 1831, 'middle': 'Clerk'}
```

5.6.4 `in` 查询字典中是否有该键

```
barn = {'cows': 1, 'dogs': 5, 'cats': 3}
```

`in` 可以用来判断字典中是否有某个特定的键：

```
'chickens' in barn
```

```
False
```

```
'cows' in barn
```

```
True
```

5.6.5 `keys` 方法， `values` 方法和 `items` 方法

```
`d.keys()`
```

返回一个由所有键组成的列表；

```
`d.values()`
```

返回一个由所有值组成的列表；

```
`d.items()`
```

返回一个由所有键值对元组组成的列表；

```
barn.keys()
```

```
dict_keys(['cows', 'dogs', 'cats'])
```

```
barn.values()
```

```
dict_values([1, 5, 3])
```

```
barn.items()
```

```
dict_items([('cows', 1), ('dogs', 5), ('cats', 3)])
```

5.7 字典的遍历

与其他组合类型一样，字典可以通过for…in语句对其元素进行遍历，基本语法结构如下：

```
for <变量名> in <字典名>:
语句块
```

```
for key in barn:
    print(key)
```

```
cows
dogs
cats
```

```
for key in barn.keys():
    print(key, ":", barn[key])
```

```
cows : 1
dogs : 5
```

```
cats : 3
```

```
for key in barn.items():
    print(key)
```

```
('cows', 1)
('dogs', 5)
('cats', 3)
```

```
for k,v in barn.items():
    print(k,":", v)
```

```
cows : 1
dogs : 5
cats : 3
```

```
for i,k in enumerate(barn):  # enumerate 函数用于遍历序列中的元素以及它们的下标:
    print(i,":", k, barn[k])
```

```
0 : cows 1
1 : dogs 5
2 : cats 3
```

字典是实现键值对映射的数据结构，请理解如下基本原则：

- 字典是一个键值对的集合，该集合以键为索引，一个键信息只对应一个值信息；
- 字典中元素以键信息为索引访问；
- 字典长度是可变的，可以通过对键信息赋值实现增加或修改键值对。