Python 程序设计基础

Python Programming

字典的概念

- 字典是存储键 / 值对数据的映射类型。
- 字典中的每个元素是一个键 / 值对，元素之间没有顺序关系。键是关键字，值是与关键字相关的，一个键对应一个值。通过键可以访问与其关联的值，反之则不行。
- 字典中的每个元素的键必须是惟一的，不能重复，值可以重复。
- 字典是可变对象。字典中的每个元素的键必须是任何可哈希（ hash ）对象。整数、浮点数、布尔值、字符串、元组等都是可哈希对象，列表等都是非可哈希对象。
- 字典中的元素是依据哈希码来放置的。哈希码是根据可哈希对象的值计算出来的一个惟一值。

字典的概念

- **字典中的键 / 值对用逗号分隔并且由一对花括号（ {} ）括住。每个键 / 值对由一个关键字，然后跟着一个冒号，再跟着一个值组成。**

```
>>> dict1 = {}
>>> dict1
{}
>>> dict2 = {'x':1, 'y':2, 'z':3}
>>> dict2
{'x': 1, 'y': 2, 'z': 3}
>>> dict3 = {True:1, 2.0:"two", "three":3, (1, 2, 3):123}
>>> dict3
{True: 1, 2.0: 'two', 'three': 3, (1, 2, 3): 123}
```

字典的概念

- 还可以使用 dict 内置函数来创建字典。

```
>>> dict1 = dict()
>>> dict1
{}
>>> dict2 = dict((('x', 1), ('y', 2), ('z', 3)))
>>> dict2
{'x': 1, 'y': 2, 'z': 3}
>>> dict3 = dict([['x', 1], ['y', 2], ['z', 3]])
>>> dict3
{'x': 1, 'y': 2, 'z': 3}
>>> dict4 = dict(x=1, y=2, z=3)
>>> dict4
{'x': 1, 'y': 2, 'z': 3}
>>> dict5 = dict(zip(['x', 'y', 'z'], [1, 2, 3]))
>>> dict5
{'x': 1, 'y': 2, 'z': 3}
>>> dict6 = dict(zip("xyz", (1, 2, 3)))
>>> dict6
{'x': 1, 'y': 2, 'z': 3}
```

字典的概念

- 一个不包含任何元素的字典被称为空字典；可以用空的花括号 {} 或 dict() 创建一个空字典。
- 注意：在 Python 3.6 之前，字典的键是无序的。从 Python 3.6 开始，字典是有序字典了，键会按照插入的顺序排列。

字典的基本操作

- 使用内置函数。
  - hash 函数判断一个对象是否可以作为字典的键，若可以，返回一个整数值（哈希码），否则抛出“ TypeError” 异常。两个对象有相同的值，它们 hash 函数返回值也相等。

```
>>> hash(123)
123
>>> hash(123.456)
1051464412201451643
>>> hash(True)
1
>>> hash("abc")
-5075053650793273680
>>> hash((1, 2, 3))
2528502973977326415
>>> hash([1, 2, 3])
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    hash([1, 2, 3])
TypeError: unhashable type: 'list'
```

```
>>> s1 = "abc"
>>> s2 = "abc"
>>> s1 == s2
True
>>> hash(s1)
-5075053650793273680
>>> hash(s2)
-5075053650793273680
```

字典的基本操作

- **运算符。**
  - **使用 in 或 not in 运算符判断一个键是否在字典中。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> 'x' in dict1
True
>>> 'z' not in dict1
False
```

  - **使用 is 或 is not 运算符判断两个字典是否是同一个对象。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict2 = {'x':1, 'y':2, 'z':3}
>>> id(dict1)
2386086677240
>>> id(dict2)
2386087545088
>>> dict1 is dict2
False
```

字典的基本操作

- 使用关系运算符 == 和 != 判断两个字典是否包含相同的键 / 值对。下面例子中，尽管 dict1 和 dict2 的键 / 值对顺序不同，但是这两个字典包含相同的键 / 值对。

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict2 = {'z':3, 'x':1, 'y':2}
>>> dict1 == dict2
True
```

字典的基本操作

- **从字典中删除键 / 值对。**
  - **使用 pop(key) 方法删除字典中键 key 对应的键 / 值对并返回它的值，若键不存在，抛出“ KeyError” 异常。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> dict1.pop('x')
1
>>> dict1
{'y': 2, 'z': 3}
>>> dict1.pop('a')
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    dict1.pop('a')
KeyError: 'a'
```

字典的基本操作

- 使用 popitem 方法删除并返回字典中的一个键 / 值对，若字典为空，抛出“ KeyError” 异常。

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> dict1.popitem()
('z', 3)
>>> dict1.popitem()
('y', 2)
>>> dict1.popitem()
('x', 1)
>>> dict1.popitem()
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    dict1.popitem()
KeyError: 'popitem(): dictionary is empty'
```

字典的基本操作

- 使用 del 语句来删除字典中的键 / 值对，若键不存在，会导致“ KeyError” 异常

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> del dict1['x']
>>> dict1
{'y': 2, 'z': 3}
>>> del dict1['a']
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    del dict1['a']
KeyError: 'a'
```

字典的基本操作

- **del 语句也可以删除整个字典。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> del dict1
>>> dict1
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    dict1
NameError: name 'dict1' is not defined
```

- **使用 clear 方法删除字典中所有的键 / 值对。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> dict1.clear()
>>> dict1
{}
```

字典的基本操作

- **合并字典。**
  - **使用 update 方法合并字典。**

```
>>> dict1 = {"red":4, "blue":1, "green":2, "yellow":5}
>>> dict1
{'red': 4, 'blue': 1, 'green': 2, 'yellow': 5}
>>> dict2 = {"purple":6, "red":1, "blue":3}
>>> dict2
{'purple': 6, 'red': 1, 'blue': 3}
>>> dict1.update(dict2)
>>> dict1
{'red': 1, 'blue': 3, 'green': 2, 'yellow': 5, 'purple': 6}
>>> dict2
{'purple': 6, 'red': 1, 'blue': 3}
```

字典的基本操作

- update 方法可以有多个参数。

```
>>> dict1 = {"red":4, "blue":1, "green":2, "yellow":5}
>>> dict1
{'red': 4, 'blue': 1, 'green': 2, 'yellow': 5}
>>> dict2 = {"purple":6, "red":1, "blue":3}
>>> dict2
{'purple': 6, 'red': 1, 'blue': 3}
>>> dict3 = {"white":0, "black":15}
>>> dict3
{'white': 0, 'black': 15}
>>> dict1.update(dict2, **dict3)
>>> dict1
{'red': 1, 'blue': 3, 'green': 2, 'yellow': 5, 'purple': 6, 'white': 0, 'black': 15}
>>> dict2
{'purple': 6, 'red': 1, 'blue': 3}
>>> dict3
{'white': 0, 'black': 15}
```

第二个参数前面必须有 **，表示字典解包。

字典的基本操作

- **还可以使用如下方法合并修改字典。**

```
>>> dict1 = {"red":4, "blue":1, "green":2, "yellow":5}
>>> dict1
{'red': 4, 'blue': 1, 'green': 2, 'yellow': 5}
>>> dict1.update(purple=6, red=1, blue=5)
>>> dict1
{'red': 1, 'blue': 5, 'green': 2, 'yellow': 5, 'purple': 6}
```

字典的基本操作

- **字典解析。**
  - **字典解析提供了一种创建字典的简洁方式。**
  - **一个字典解析由花括号组成。花括号内包含后跟一个 for 子句的表达式，之后是 0 或多个 for 子句或 if 子句。字典解析产生一个由表达式求值结果组成的字典。**

    {key_expr:value_expr for iter_var in iterable}

    首先循环 iterable 里所有内容，每一次循环，都把 iterable 里相应内容放到 iter_var 中，再在 key_expr 和 value_expr 中应用该 iter_var 的内容，最后用 key_expr:value_expr 的计算值生成一个字典。

    {key_expr:value_expr for iter_var in iterable if cond_expr}

    加入了判断语句，只有满足条件的才把 iterable 里相应内容放到 iter_var 中，再在 key_expr 和 value_expr 中应用该 iter_var 的内容，最后用 key_expr:value_expr 的计算值生成一个字典。

字典的基本操作

```
>>> dict1 = {x:0.5 * x for x in range(5)}
>>> dict1
{0: 0.0, 1: 0.5, 2: 1.0, 3: 1.5, 4: 2.0}
>>> dict2 = {x:0.5 * x for x in range(5) if x < 3}
>>> dict2
{0: 0.0, 1: 0.5, 2: 1.0}
>>> dict3 = {key:value for (key, value) in zip("xyz", [1, 2, 3])}
>>> dict3
{'x': 1, 'y': 2, 'z': 3}
```

字典的基本操作

- **其他常用字典方法。**
  - **keys 方法获取字典的所有键。返回由所有键构成的一个可迭代对象 dict_keys 。**
  - **values 方法获取字典的所有值。返回由所有值构成的一个可迭代对象 dict_values 。**
  - **items 方法获取字典的所有键 / 值对。返回由所有键 / 值对构成的一个可迭代对象[illegible]式返回。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> dict1.keys()
dict_keys(['x', 'y', 'z'])
>>> dict1.values()
dict_values([1, 2, 3])
>>> dict1.items()
dict_items([('x', 1), ('y', 2), ('z', 3)])
```

字典的基本操作

- fromkeys 方法：第一个参数是字典的键；第二个参数是键对应的值，可选，若不提供，默认是 None 。创建并返回一个具有相同值的字典。
- setdefault 方法：第一个参数是字典的键；第二个参数是键对应的值，可选，若不提供，默认是 None 。对在字典中的键，返回对应值；对不在字典中的键，新建键 / 值对，返回对应值。

```
>>> dict1 = {}.fromkeys(['x', 'y', 'z'])
>>> dict1
{'x': None, 'y': None, 'z': None}
>>> dict2 = {}.fromkeys(['x', 'y', 'z'], 0)
>>> dict2
{'x': 0, 'y': 0, 'z': 0}
```

```
>>> dict1 = {'A':65, 'B':66, 'C':67}
>>> dict1
{'A': 65, 'B': 66, 'C': 67}
>>> dict1.setdefault('A')
65
>>> dict1.setdefault('D', 68)
68
>>> dict1
{'A': 65, 'B': 66, 'C': 67, 'D': 68}
```

字典的基本操作

- 和列表一样。为了避免可变对象之间使用 = 赋值存在的关联性问题。使用 copy 方法进行“浅复制”。

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> dict1
{'x': 1, 'y': 2, 'z': 3}
>>> dict2 = dict1.copy()
>>> dict2
{'x': 1, 'y': 2, 'z': 3}
>>> id(dict1)
2386087629520
>>> id(dict2)
2386087629448
>>> dict1['x'] = 111
>>> dict1
{'x': 111, 'y': 2, 'z': 3}
>>> dict2
{'x': 1, 'y': 2, 'z': 3}
```

字典的基本操作

- **遍历字典。**
  - **遍历字典中的所有键。**
  - **遍历字典中的所有值。**
  - **遍历字典中的所有键 / 值对。**

```
>>> dict1 = {'x':1, 'y':2, 'z':3}
>>> for key in dict1.keys():
        print(key, end=' ')

x y z
>>> for value in dict1.values():
        print(value, end=' ')

1 2 3
>>> for key, value in dict1.items():
        print(key, ":", value, sep='', end=' ')

x:1 y:2 z:3
```

例子

- **编写程序，输入若干个整数，整数之间以空格间隔，统计每个输入整数的出现次数。分行升序输出每个整数及其出现次数**
  - **整数作为键，出现次数作为对应的值，构成一个键 / 值对。字典存储键 / 值对。**

```
line = input().split()
numbers = [eval(x) for x in line]
d = {}
for number in numbers:
    if number in d:
        d[number] += 1
    else:
        d[number] = 1
counts = sorted(list(d.items()))
for i in range(len(counts)):
    print(counts[i][0], ':', counts[i][1], sep='')
```

```
11 22 35 68 97 63 22 68 11
11:2
22:2
35:1
63:1
68:2
97:1
```

通过列表解析，将整数存放在列表中。

对每一个整数，若整数作为键已经在字典中，则出现次数加 1 ；否则给字典新增一个键 / 值对，键为整数，值为出现次数 1 。

将字典转换为列表，即 [（整数， 出现次数），（整数， 出现次数），…] 。并按整数升序排序。