第四章 PYTHON数据结构

字典

4.1 字典

字典是一个存储键值对集合的Python容器。它通过使用关键字来快速获取、删除和更新值。

一个字典对象中无序地存储了若干个条目，用一对花括号（{}）括起来。每个条目都是一个键值对，由类似“关键字:对应值”的结构组成，即一个关键字和一个对应值。关键字在字典中是唯一的，每个关键字唯一地匹配一个值。

字典类型通过Python的内置类dict来定义，因此使用dict函数也可以创建一个字典。

```
>>> x = {1:2,'1':4,(1,2):[1,2,3]}#字典的键是不可变对象
>>> y ={[]:2,1:3}#可变对象不能成为键
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    y ={[]:2,1:3}#可变对象不能成为键
TypeError: unhashable type: 'list'
>>> z = {1:2,1:3} #如果键有重复，只会留一个
>>> z
{1: 3}
>>> a = [1,2,3]
>>> b =list('abc')
>>> c = list(zip(a,b))
>>> c
[(1, 'a'), (2, 'b'), (3, 'c')]
>>> dict(c)
{1: 'a', 2: 'b', 3: 'c'}
```

1：键是不可变对象

2：键不能重复出现

3：zip函数用法

4：dict函数用法

4.2 字典的基本操作

添加、修改、获取、删除条目

- 关键字（键）在字典中的作用相当于列表中的下标。通过类似“字典对象名[关键字]”的语法可以读或写字典中的条目。
- 要从字典中删除一个条目，可使用del关键字进行删除。

字典推导式

- 与列表和集合相似，字典同样支持推导式生成。不同的是，字典推导式起始位置的表达式应该是“键:值”格式的。

```
>>> x = dict(c)
>>> x
{1: 'a', 2: 'b', 3: 'c'}
>>> x[1] #访问
'a'
>>> x[1]='abcdefg'#修改
>>> x
{1: 'abcdefg', 2: 'b', 3: 'c'}
>>> x[4]="12333" #创建一个新键4，其值为"12333"
>>> x
{1: 'abcdefg', 2: 'b', 3: 'c', 4: '12333'}
>>> del x[4] #删除键4及其值
>>> x
{1: 'abcdefg', 2: 'b', 3: 'c'}
>>> a = [1,2,3,4,5]
>>> {i:i**2 for i in a} #字典迭代
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

1：由键访问值

2：由键修改值

3：添加新键值对

4：删除键和值

5：字典迭代

4.3 字典的基本操作

in/not in运算符

- 使用in或not in运算符可以用来判断一个关键字是否在字典中。

比较运算符

- 可以使用运算符==和!=来检测两个字典中的条目是否相同。

字典也是无序对象

```
>>> 1 in x
True
>>> 'c' in x
False
>>> {1:2,3:'a'}=={3:'a',1:2}
True
>>> {1:2,3:'a'}=={3:'a',1:3}
False
```

4.4 字典相关的函数和方法

dict类的成员函数：

- clear()
- copy()
- get(key[, default])
- items()
- keys()
- pop(key[, default])
- popitem()
- setdefault(key[, default])
- update([other])
- values()

```
>>> x = 'qwexubhdk'
>>> y = {i:x.index(i) for i in x}
>>> y
{'q': 0, 'w': 1, 'e': 2, 'x': 3, 'u': 4,
 'b': 5, 'h': 6, 'd': 7, 'k': 8}
>>> y.pop('a')
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    y.pop('a')
KeyError: 'a'
>>> y.pop('q')
0
```

字典迭代

删除条目

```
>>> y.get('w')
1
>>> y.get('a')
>>> y['a']
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    y['a']
KeyError: 'a'
>>> y.get('a','No this key!')
'No this key!'
```

由键访值

没有键时可指定返回对象

```
>>> y.items()
dict_items([('w', 1), ('e', 2), ('x', 3),
            ('u', 4), ('b', 5), ('h', 6),
            ('d', 7), ('k', 8)])
>>> y.keys()
dict_keys(['w', 'e', 'x', 'u', 'b', 'h', 'd', 'k'])
>>> y.values()
dict_values([1, 2, 3, 4, 5, 6, 7, 8])
>>> list(y.values())
[1, 2, 3, 4, 5, 6, 7, 8]
>>> y.popitem()
('k', 8)
>>> y
{'w': 1, 'e': 2, 'x': 3, 'u': 4,
 'b': 5, 'h': 6, 'd': 7}
```

返回条目

返回键

返回值

删除条目

```
>>> y.setdefault('a',80)
80
>>> y
{'w': 1, 'e': 2, 'x': 3, 'u': 4,
 'b': 5, 'h': 6, 'd': 7, 'a': 80}
>>> y.setdefault('a',90)
80
>>> y
{'w': 1, 'e': 2, 'x': 3, 'u': 4,
 'b': 5, 'h': 6, 'd': 7, 'a': 80}
>>> x={'u': 4, 'b': 45, 'x': 40, 's': 5}
>>> x.update(y)
>>> x
{'u': 4, 'b': 5, 'x': 3, 's': 5, 'w': 1,
 'e': 2, 'h': 6, 'd': 7, 'a': 80}
>>> y
{'w': 1, 'e': 2, 'x': 3, 'u': 4,
 'b': 5, 'h': 6, 'd': 7, 'a': 80}
```

将键'a'设定默认值80

键'a'经存在，无变化

由y更新x，y不变

4.5列表，字典中的深复制和浅复制

```
>>> x = [1, 2, 3]
>>> y = x.copy()
>>> y
[1, 2, 3]
>>> x[1]=100
>>> y
[1, 2, 3]
>>> a = [1, [1, 2, 3], 3]
>>> b = a.copy()
>>> a[1].append(4)
>>> a
[1, [1, 2, 3, 4], 3]
>>> b
[1, [1, 2, 3, 4], 3]
```

浅复制，只复制最外面一层

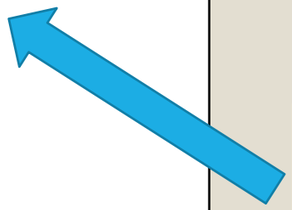
由于列表没有嵌套其它列表，因此产生了一个完整复制对象。

浅复制，只复制最外面一层

由于列表有嵌套其它列表，因此不能产生一个完整复制对象，列表中[1,2,3]为共享对象。

切片和列表迭代得到也是浅复制。

```
>>> from copy import deepcopy
>>> b = deepcopy(a)
>>> a
[1, [1, 2, 3, 4], 3]
>>> b
[1, [1, 2, 3, 4], 3]
>>> a[1][:2]=[]
>>> a
[1, [3, 4], 3]
>>> b
[1, [1, 2, 3, 4], 3]
```



深复制，产生了一个完整复制对象。

```
>>> x = {1:2, 3:[4, 5, 6]}
>>> y =x.copy()
>>> y[3].append(7)
>>> y
{1: 2, 3: [4, 5, 6, 7]}
>>> x
{1: 2, 3: [4, 5, 6, 7]}
```

浅复制。字典嵌套有列表，因此没有产生了一个完整复制对象。

深复制。完全复制。

```
>>> x
{1: 2, 3: [4, 5, 6, 7]}
>>> b = deepcopy(x)
>>> b
{1: 2, 3: [4, 5, 6, 7]}
>>> b[3][1]=50
>>> x
{1: 2, 3: [4, 5, 6, 7]}
>>> b
{1: 2, 3: [4, 50, 6, 7]}
```

4.6 有关Python 数据的补充说明

- str()将对象转变为字符串
- repr将对象用字符串表示出来，可用eval将原始对象还原

```
>>> y = '123'
>>> x = repr(y)
>>> x
"'123'"
>>> z = str(y)
>>> eval(x)
'123'
>>> eval(z)
123
>>> eval("+".join('1234'))
10
>>> eval(repr("+".join('1234')))
'1+2+3+4'
```

4.7 转义字符

有些字符无法直接输出，可以通过转义其它字符实现

转义字符	描述	转义字符	描述
\(在行尾时)	续行符	\n	换行
\\	反斜杠符号	\t	横向制表符
\’	单引号	\r	回车
\”	双引号	\f	换页
\a	响铃	\000	空
\b	退格	\other	使字符按普通形式输出

在字符串前加上r可以消除字符中的转义
语法:r“带转义字符的字符串”

```
>>> print("今天天气真好！\n 欧耶！")
今天天气真好！
 欧耶！
>>> print("今天天气真好！\t 欧耶！")
今天天气真好！     欧耶！
>>> print("今天天气真好！\\ 欧耶！")
今天天气真好！\ 欧耶！
>>> print(r"今天天气真好！\t 欧耶！")
今天天气真好！\t 欧耶！
```