Python核心数据类型-字典
相对于前面学的有序集合列表，字典是无序的集合，差别在于字典的值是通过键的索引键的形式来获取，而不是通过偏移来获取， 字典的数据格式为 key:value

字典的主要特点：
1.通过键而不是通过偏移来读取
字典通过键将一系列的值联系起来，这样就可以使用键从字典中取出一项，而不是像列表那样使用相对偏移

```
In [55]: x
Out[55]: {'a': 'apple'}

In [56]: x['a']
Out[56]: 'apple'
```

2. 任意对象的无序集合： 与前面讲的列表不同 存储在字典中的项没有明确的顺序， 键提供了字典中元素的象征性位置而不是物理位置

3.无序 序列运算无效
与列表不同 保存在字典中的项没特定的顺序 ， 字典是映射机制而不是序列，字典元素之间没有顺序的概念， 类似 + 切片这些操作在字典中是无法操作的

3.变长 异构 任意嵌套
与列表类似 字典可以在原处增长或缩短 可以包含任意类型的对象， 支持任意深度的嵌套

```
In [57]: x = {'a': {'b': {'c': 1}}}

In [58]: x
Out[58]: {'a': {'b': {'c': 1}}}
```

4.字典属于可变类型映射

```
In [58]: x
Out[58]: {'a': {'b': {'c': 1}}}
In [59]: x['a'] = apple
In [61]: x
Out[61]: {'a': 'apple'}
```

5.字典的常用方法

```
In [1]: a = {}
In [51]: x.
x.clear          x.has_key        x.itervalues   x.setdefault   x.viewkeys
x.copy           x.items          x.keys         x.update
x.viewvalues
x.fromkeys       x.iteritems      x.pop          x.values
x.get            x.iterkeys       x.popitem      x.viewitems
```

1. clear:

用于删除字典内的元素

```
In [6]: a = {'a': 100, 'b': 200}
In [7]: len(a)
Out[7]: 2
In [8]: a.clear()
In [9]: len(a)
Out[9]: 0
```

2. fromkeys:

如果所有键的值都相同 可以用这个特殊的形式对字典进行初始化 传入一个键的列表 以及所有键的初始值

```
In [35]: c = {}
In [36]: d = c.fromkeys(['a', 'b'], 100)
In [37]: d
Out[37]: {'a': 100, 'b': 100}
```

3. has_key 或 直接用 in 如 ‘a’ in {}
判断键是否在字典中 如果在则返回true 不在则返回false

```
In [38]: d
Out[38]: {'a': 100, 'b': 100}
In [39]: d.has_key('a')
Out[39]: True
In [40]: d.has_key('c')
Out[40]: False
```

4. items()
返回字典中的键 值对

```
In [52]: d
Out[52]: {'a': 100, 'b': 100}
In [53]: d.items()
Out[53]: [('a', 100), ('b', 100)]
```

5. iteritems

iteritems返回的是一个迭代器

```
In [55]: d.iteritems()
Out[55]: <dictionary-itemiterator at 0x25c57e0>   #返回一个迭代器
In [56]: for j in d.iteritems():
   ....:       print j
   ....:
('a', 100)
('b', 100)
```

6 keys

```
In [70]: b
Out[70]: {'a': 100, 'b': 200}
In [71]: b.keys()
Out[71]: ['a', 'b']
```

7. iterkeys

```
In [72]: b.iterkeys()
Out[72]: <dictionary-keyiterator at 0x25c9050>   ##返回一个迭代器
In [74]: for j in b.iterkeys():
   ....:       print j
   ....:
a
b
```

8. values

```
In [75]: b
Out[75]: {'a': 100, 'b': 200}
In [76]: b.values()
Out[76]: [100, 200]
```

9. itervalues

```
b.itervalues():   #返回一个迭代器
In [77]: for j in b.itervalues():
   ....:       print j
   ....:
100
200
```

10. pop

```
In [88]: d = {'a': 100, 'b': 200, 'c': 300, 'd': 400}
In [89]: d.pop('c')   ##删掉指定的key和value
Out[89]: 300
In [90]: d
Out[90]: {'a': 100, 'b': 200, 'd': 400}
```

11.popitem

```
In [92]: d.popitem()   ##随机删除掉字典中的   key  和  value的组合
Out[92]: ('a', 100)
In [93]: d
Out[93]: {'b': 200, 'd': 400}
```

12.get函数返回指定键的值 如果值在字典中不存在则返回默认值

13. setdefault 如果键不存在于字典中 将添加键并设置默认值

```
In [1]: a = {}
In [2]: a.get('a', 100)
Out[2]: 100
In [3]: a
Out[3]: {}

In [4]: a.setdefault('a', 100)
Out[4]: 100
In [5]: a
Out[5]: {'a': 100}
```

14. update
把字典 b 的键\值 更新到 a中

```
In [15]: a
Out[15]: {'a': 100}
In [16]: b
Out[16]: {'b': 100}
In [17]: a.update(b)
In [18]: a
Out[18]: {'a': 100, 'b': 100}
```

15. copy
返回一个字典的复制

```
In [36]: a
Out[36]: {'a': 20, 'b': 100}
In [37]: e = a.copy()
In [38]: e
Out[38]: {'a': 20, 'b': 100}
```

Viewkeys viewvalues viewitems ##返回一个view对象 优点是如果字典发生了变化， view也会同步发生变化

```
In [74]: x
```

```
Out[74]: {'a': 'apple', 'b': 'banana'}

In [75]: y = x.keys()

In [76]: z = x.viewkeys()

In [77]: x.pop('a')
Out[77]: 'apple'

In [78]: y
Out[78]: ['a', 'b']

In [79]: z
Out[79]: dict_keys(['b'])
```

字典用法注意事项

1. 序列运算无效：字典是映射机制 不是序列 字典的元素之间没有顺序的概念

2. 新索引赋值会添加新的项目

3. key不一定是字符串， 任何不可变的类型其实都可以 数字 字符串 元组

```
In [3]: x = {}
In [4]: x[(1, 2, 3)] = 'z'
In [5]: x
Out[5]: {(1, 2, 3): 'z'}
```

创建字典的四种方法

{’name’: ‘mel’, ‘age’: 45}

知道字典的 key 与 value 组成的字典

```
d ={}
d[’name’] = ‘mel'
```

d[‘age’] = 45
动态建立字典的方法

dict(name=‘liu’, age=45) #需要键都是字符串

dict([(’name’, ‘mel’), (‘age’, 45)])

```
In [47]: a = ('a', 'b', 'c')
In [48]: b = ('apple', 'big', 'cat')
In [49]: zip(a, b)
Out[49]: [('a', 'apple'), ('b', 'big'), ('c', 'cat')]
In [50]: dict(zip(a, b))
Out[50]: {'a': 'apple', 'b': 'big', 'c': 'cat'}
```

#在程序运行时 把键和值逐步建成序列的时候 此种创建方式比较有用

避免key-error错误

尝试获取一个字典中不存在的key的值时将报 key-error错误

```
In [55]: a
Out[55]: {'age': 45, 'name': 'liu'}
In [56]: a['cat']
<ipython-input-56-933689bb4a88> in <module>()
----> 1 a['cat']
KeyError: 'cat'
```

以下列举三种方法避免keyerror错误

1.通过if进行判断

```
In [57]: a
Out[57]: {'age': 45, 'name': 'liu'}

In [58]: if 'age' in a:
   ....:     print 'yes'
   ....: else:
   ....:     print 'no'
   ....:
yes
In [59]: if 'cat' in a:
   ....:     print 'yes'
   ....: else:
```

```
   ....:       print 'no'
no
```

2. try判断

```
In [62]: try:
      print a['age']
except:
      print 'no'
    ....:
45

In [63]: try:
      print a['cat']
except:
      print 'no'
    ....:
no
```

3.

```
In [66]: a
Out[66]: {'age': 45, 'name': 'liu'}
In [67]: a['cat'] = a.get('cat', 100)
In [68]: a
Out[68]: {'age': 45, 'cat': 100, 'name': 'liu'}
```

通过 字典的 get方法为键设置初始值

内置函数

1. len() ##返回字典长度
2. del

```
In [66]: x
Out[66]: {'a': 'apple', 'b': 'banana'}

In [67]: del x['a']
```

```
In [68]: x
Out[68]: {'b': 'banana'}

In [69]: del x

In [70]: x
```

3. 判断 key 在不在字典中

```
In [71]: x = {'a': 'apple', 'b': 'banana'}
In [72]: 'a' in x
Out[72]: True

In [73]: 'c' in x
Out[73]: False
```