Reboot

python基础教程_03

woniuppp

Reboot

上节回顾---List

- 定义一个List
- 通过索引取值
- 可遍历
- 成员是否存在
- 求长度，最大值，最小值
- 删除元素

Reboot

上节回顾---List

- 修改值和切片
- 切片批量赋值
- 列表的方法
  - append
  - count
  - index
  - extend
  - insert
  - pop
  - reverse

Reboot

复习一下上节任务，小练习

- 用户密码登陆系统，
  - 密码错误三次，锁定用户，不能登陆
- 购物车
  - 用户登陆之后，才能看到商品列表
  - 可以输入商品名，把商品加入购物车
  - 打印购物车列表
- 数组排序(附加题)

Reboot

新的数据结构---tuple

- 元组是另一种有序的列表
- 和list非常类似，但是tuple一旦创建，就不能修改

```
>>> t = ('wd','pc','me')
>>> t1 = tuple('abc')
```

- 创建tuple和list不同之处，就是用()代替[]
- 通过t[0]等方式访问元素，和list类似
- 现在，t就不能修改了,append,pop等方法都没有

Reboot

- 创建一个单元素的tuple

```
>>> t = ()
>>> print t
>>>
>>> t = (1)
>>> print t
```

- 好像打印的，不是tuple，为啥呢
- ()既可以表示tuple，也可以作为括号运算，所以（1）被python计算出结果1

```
>>> t = (1,)
>>> print t
>>> t = (1,2,3,)
>>> print t
```

Reboot

可变的tuple

```
t = (1,2,[3,4])
t1 = t[2]
t1[0] = 5
print t
```

- 复制和引用

```
l = [1,2,3,4]
l1 = l
l1[0] = 5
print l1
```

Reboot

下面问题来了

存储很多有一一对应关系的账号密码，用只能根据索引list略low

- 只是单纯的想把账号和密码联系起来，用两个list不太方便
- 我们需要的是，可以直接通过账号，直接找到密码

Reboot

dict(字典)隆重登场

- 用大括号定义

```
d = {
  'wd':123,
  'pc':456
}
```

- 通过dict定义

```
lst = [('wd','pc'),[1,2]]
dct = dict(l)
print dct
```

- 字典的基本行为和list类似
- len(d)返回总数
- 名字称为key，对应的密码称为value
- 通过key，找value
- 通过key，可以更新value
- key可以是任何不可变类型
  - 字符串，数字，元组等
- del 删除元素
- k in d 检测字典d里，是否有k这个key

Reboot

访问字典

- 通过key获取value

```
d = {
  'wd':123,
  'pc':456,
  'me':789
}
print d['wd']
print d['wd1']
```

Reboot

判断元素

- key不存在会报错
- 所以先用in判断一下

```
if 'wd' in d:
  print d['wd']
```

- 或者用get，不会报错

```
d.get('wd')
d.get('wd1')
```

Reboot

dict特点

- 查找速度非常快，1个元素，和10W的元素，查找速度基本一样
- list查找速度随着数量增加而变慢
- dict占用内存
- dict是没有顺序的
- dict的key是不可变的（list就不能当key，报错）
- key不能重复

Reboot

更新dict

```
>>> d['wd'] = 9
>>> d
>>>
```

- 如果key不存在，就创建，如果存在，就更新值

Reboot

字段方法

clear

- 清除字典中所有的项
- 操作原字典

```
d = {}
d['name'] = 'wd'
d['age'] = 12
print d
d.clear()
print d
```

Reboot

clear的用处

```
x = {}
y = x
x['name'] = 'wd'
print y
x = {}
print y
```

```
x = {}
y = x
x['name'] = 'wd'
print y
x.clear()
print y
```

Reboot

copy，返回一个副本（和直接复制的区别）

```
x = {'name':'wd'}
y = x.copy()
y['age'] = 12
print y
print x
```

Reboot

formkeys 使用给定的键建立新的字典，默认值对应None

```
>>> {}.fromkeys(['name','age'])
>>> {}.fromkeys(['name','age'],'wd')
```

get

更宽松的通过key访问value的方式，key不存在不会报错，可以提供默认值

```
d = {}
print d['name']
print d.get('name')
print d.get('name','wd')
d['name'] = 'pc'
print d.get('name','wd')
```

Reboot

has_key

has_key和in的功能一样

```
d = {}
d.has_key('name')
```

Reboot

items 将字典所有项以列表的形式返回，每一项是(key,value)

```
d = {name:'wd',age:'12'}
print d.items()
```

Reboot

keys 字典中的键，以列表的形式返回

```
d = {name:'wd',age:'12'}
print d.keys()
```

Reboot

values 以列表的形式返回字典中的value（和keys对应）

Reboot

popitem 类似于list.pop，弹出一个随机的项

```
d = {name:'wd',age:'12'}
print d.popitem()
print d
```

Reboot

setdefault 类似于get，可以赋默认值

```
d = {}
print d.setdefault('name','wd')
print d
print d.setdefault('name','pc')
print d
```

Reboot

update 可以用一个字典去更新另外一个字典

```
d = {name':wd'}
d1 = {name':pc'}
d.update(d1)
print d
```

Reboot

字符串

```
str1 ='hello'
print str[0]
print str[0:2]
print 'e' in 'hello'
print 'e' not in 'hello'
```

字符串

join

```
'|'.join(['hello','world'])
```

字符串

- find(a) 检测 str 是否包含a，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1

```
'helloworld'.find('e')
'helloworld'.find('s')
```

Reboot

- index 用法和find一样，但是不存在的时候，报错

```
'helloworld'.index('e')
'helloworld'.index('s')
```

Reboot

字符串

capitalize

```
'hello'.capitalize()
```

字符串

- count(a) 统计a出现的次数

```
'hello'.count('l')
```

字符串

- replace 字符串替换(默认替换全部)，第三个参数，限定替换次数

```
'hello,world'.replace('ll','ww')
'hello,wollrld'.replace('ll','ww')
'hello,wollrld'.replace('ll','ww',1)
```

Reboot

- split 分隔字符串

```
'hello,world,pc,wd'.split(',')
```

- rstrip 删除串尾的空格
- lstrip 删除串首的空格
- strip 综合上面两个

```
'   hello   world               '.rstrip()
'   hello   world               '.lstrip()
'   hello   world               '.strip()
```

Reboot

微信公众号

Q & A

〈Thank You!〉

Powered By nodePPT v0.9.8-3