

Phthon 学习笔记

目录

Phthon 学习笔记	1
1.运行第一个 python 程序。	2
2.编译.py 文件	3
2.1 编写 compile.py (名称随便),文件内容如下:	3
2.2 直接执行 python -O -m py_compile test.py	3
3.变量的定义	3
4.四则运算	4
5. 变量类型	4
5.1 整型.....	4
5.2 复数.....	5
5.3 字符串	5
6.元组	7
7.列表	8
8.字典	10
9.流程控制	13
9.1 if-else 语句.....	13
9.2 for 循环	14
9.3 for 循环和 if 语句.....	16
9.4 for 循环和 else, 以及 break, continue, pass.....	17
9.5 while & else.....	18
10.函数.....	18
10.1 参数及书写规则	18
10.2 局部变量, 全局变量, *t 元组参数传递, 格式化打印	19
10.3 使用字典作为参数, 处理冗余参数	20
10.4 lambda 匿名函数	20
10.5 使用字典构建 python 的 switch 语句	20
10.6 内建函数	21
11.引入模块	26
12.正则表达式	27
12.1 元字符。	27
12.2 字符集&多次匹配.....	28
12.3 将正则编译成类	30
12.4 查看某个类函数&变量的方法。	32
12.5 正则的其他编译属性&分组	32
12.7 爬虫下载网页图片	34
13.拷贝.....	35
13.1 浅拷贝.....	35

14.文件操作	36
15.OS 模块	39
15.1 常用 os 函数	39
15.2 遍历目录.....	39
15.异常处理	40
16.MySqlLdb.....	43

1.运行第一个 python 程序。

Linux 下。

#可以在命令行下直接执行 `python test.py`

文件内容如下

```
print 'hello'
```

#或者将文件 `chmod u+x test.py` 后直接执行

```
./test.py
```

文件内容如下。

```
#!/user/bin/python
print 'hello'
```

#需要引入 **python** 程序，以上两种方法是通过 **python** 直接解释，不需要编译。

2.编译.py 文件

Windows 下，在 **python** 路径下。

2.1 编写 **compile.py**（名称随便），文件内容如下：

```
Import py_compile
py_compile.compile('test.py')
```

#然后 **python compile.py** 开始编译。

生成 **test.pyc** 文件，是二进制文件。

#可以在 **python** 路径下 直接执行 **python test.pyc** 来运行。

2.2 直接执行 **python -O -m py_compile test.py**

会生成 **test.pyo** 文件。

#可以在 **python** 路径下 直接执行 **python test.pyo** 来运行。

#-O 一定要大写

3.变量的定义

变量只能用数字、字母、下划线，字母不能开头。不能用关键字（保留字）。

变量定义方法：**a=1**

变量对应的数值是具体存储在内存中的。每个变量名称相当于标签，指向内存中的地址来读取数值。

相同的数值一般只存储一次。不同的变量（标签）可以指向同一个数值（内存地址）。

例如：

```
>>> a=12
>>> b=12
>>> id(a)
3997284
>>> id(b)
3997284
```

相同的内存地址，应用了相同的数值。

+ **-** ***** **/** **%** 和 **java** 基本相同，不过可以实现 **2.0/4 = 0.5** 的结果
// 为整除符号 ****** 为乘方 如 **3**2=9**

代表字符 **b** 出现 **3** 次

```
>>> 'b'*3  
'bbb'
```

and **or** **not** 与或非。

<< **>>** 左移 右移

& **|** **^** 按位 与 或 异或

4.四则运算

编写如下脚本：

```
a=raw_input()  
b=raw_input()
```

```
print a+b  
print a-b  
print a*b  
print a/b
```

在进行减法的时候会报错。因为 **a,b** 默认是字符串。使用如下方法就可以改为 **int** 类型了。

```
a=int(raw_input())  
b=int(raw_input())  
print a+b  
print a-b  
print a*b  
print a/b
```



5. 变量类型

5.1 整型

范围：**-2,147,483,648** 到 **2,147,483,647**

长整型范围：几乎所有整数都可以

#变量类型不需要事先定义，而是根据存储模式来判断是那种类型。

#可以定义一个 **int** 类型的 **long** 类型的变量。

```
>>> a=1
```

```
>>> type(a)
<type 'int'>
>>> a=121212121212121212
>>> type(a)
<type 'long'>
>>> a=12l
>>> type(a)
<type 'long'>
>>> a=12L
>>> type(a)
<type 'long'>
```

5.2 复数

```
>>> c=12j
>>> type(c)
<type 'complex'>
```

5.3 字符串

字符串可以使用'&““ 或者””””“””(这个可以用来生成注释)
当字符串中含有特殊字符的时候一般使用”“
当需要保持文本格式的时候议案使用””””“””
有特殊字符的时候使用\（转移字符）来屏蔽器特殊含义。

```
>>> say="let's go"
>>> say
"let's go"
>>> print say
let's go
>>> mail="tom:\n hello\n i am jack"
>>> mail
'tom:\n hello\n i am jack'
>>> print mail
tom:
  hello
  i am jack
>>> mail="""tom:
...     i an tom
...     goodbye
... """
>>> print mail
```

```
tom:
    i an tom
    goodbye

>>> mail
'tom:\n\ti an tom \n\tgoodbye\n'
```

字符串是序列化得变量，可以使用如下方式访问其中的一项（+为字符串连接符）

```
>>> a='asdf'
>>> a[1]
's'
>>> a[0]
'a'
>>> a[0]+a[1]
'as'
```

对序列切片，正数的时候序号从 **0** 开始。三个参数依次是，初始位置，结束位置，步长。结束位置的那项是截取不到的，初始位置必须小于结束位置。默认是从左到右截取。

```
>>> a="abcdef"
>>> a[1:3]
'bc'
>>> a[:3]
'abc'
>>> a[1:]
'bcdef'
>>> a[4:1]
''
```

负数的时候序号从 **-1** 开始。三个参数依次是，初始位置，结束位置，步长。结束位置的那项是截取不到的，初始位置必须小于结束位置。默认是从左到右截取。

```
>>> a="abcdef"
>>> a[-4:-1]
'cde'
>>> a[-4:]
'cdef'
>>> a[:-1]
'abcde'
```

步长是正数的时候：

不加步长的时候表示从初始位置（初始位置也截取）截取到结束位置减一的那个位置。
加步长的时候表示从初始位置开始（初始位置也截取）每隔（步长-1）个数据截取一个直到结束位置来构成新的数据。

步长是负数的时候：

加步长的时候表示从初始位置开始（初始位置也截取）每隔（|步长|-1）个数据截取一个直到结束位置来构成新的数据。默认是从右到左截取。初始位置必须大于绝对位置。

```
>>> a="abcdefghijkl"
>>> a[-1:-8:-1]
'lkjihgf'
>>> a[-1:-8:-2]
'ljhf'
```

当初始位置小于结束位置的时候，选出来的是空值，其他的和正数步长相同。

```
>>> a[-8:-1:-2]
''
```

6.元组

元组的值是不可改变的

序列定义需要加(), 使用方法如下

len()方法

in

max() min() cmp()

```
>>> t1=(1,"xihuan","haha",2)
>>> len(t1)
4
>>> t1[1]*3
'xihuanxihuanxihuan'
>>> 2 in t1
True
>>> max(t1)
'xihuan'
>>> min(t1)
1
```

比较大小

```
>>> cmp(2,1)
1
>>> cmp(2,2)
0
>>> cmp(2,2)
0
```

定义空的元组，定义单个元组

```
>>> t2=()
>>> print t2
```

```
()
>>> t3=(1)
>>> type(t3)
<type 'int'>
>>> t4=(2,)
>>> type(t4)
<type 'tuple'>
```

从元组获取变量值，需要注意的是，必须全部获取，否则会报错。。。。。

```
>>> t1
(1, 'xihuan', 'haha', 2)
>>> a,b,c,d=t1
>>> a
1
>>> b
'xihuan'
>>> c
'haha'
>>> d
2
>>> a,b=t1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
>>>
```

7.列表

和元组一样都是序列，所以也可以调用 **len,cmp,in,max,min** 等函数。

定义方法

```
>>> list=[15921874381,"yuanchun.yao"]
>>> list
[15921874381L, 'yuanchun.yao']
>>> print list
```

调用

```
>>> list[0]
15921874381L
```

使用 **len** 方法

```
>>>len(list)
2
>>> len(list[1])
```


12

添加

```
>>> list.append("laomei1")
>>> list.append("laomei2")
>>> list
['yuanchun', 15921874381L, 'laomei1', 'laomei2']
```

删除，三种删除方式。

```
>>> list
['yuanchun', 15921874381L, 'laomei1']
>>> list.remove("laomei1") #只删除出现的第一个
>>> list
['yuanchun', 15921874381L]
>>> list.remove(list[1])
>>> list
['yuanchun']
```

判定存在

```
>>> "yuanchun" in list
True
```

修改值

```
>>> list=["yuanchun.yao",15956423241]
>>> list[1]=15921874381
>>> list
['yuanchun.yao', 15921874381L]
```

列表修改某一项的值之后，依旧存在当前的列表中，内存地址不变。

但是元组不可以修改某一项的值，元组只能从新定义，但是定义后的元组，就不存放在以前的地址中了。

列表

```
>>> list
['yuanchun.yao', 15921874381L]
>>> id(list)
40870056
>>> list[0]="laomei"
>>> id(list)
40870056
```

元组

```
>>> t=(1,2,3)
>>> id(t)
4898528
```

```
>>> t=(1,2,4)
>>> id(t)
40617304
```

8.字典

想要生成一一对应的关系，类似 **java** 中的 **map**。可以使用字典 **key**，**value**（键，值对）的方式。

我们可以使用列表来够成简单的对应功能，但是他们的值之间没有必然的联系。

```
>>> l1=[1,2,3]
>>> l2=["yuanchun","dadio","zhandouliqiang"]
>>> zip(l1,l2)
[(1, 'yuanchun'), (2, 'dadio'), (3, 'zhandouliqiang')]
>>> t1=zip(l1,l2)
>>> t1
[(1, 'yuanchun'), (2, 'dadio'), (3, 'zhandouliqiang')]
```

字典是张无序的哈希表。定义方式如下

```
>>> dic1={"name":"yuanchun.yao","age":24}
>>> dic1
{'age': 24, 'name': 'yuanchun.yao'}
>>> dic1["age"]
24
```

默认不带引号的字母构成的键必须是事先定义好的变量。否则会报错。

```
>>> name="mingzi"
>>> dic2={name:"yuanchun.yao","age":24,2:"shabi"}
>>> dic2
{'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> dic2[name]
'yuanchun.yao'
>>> dic2["mingzi"]
'yuanchun.yao'
```

字典创建的其他方法（不常用）

工厂方法

例如：**fdict=dict([“x”,1],[“y”,2])**

内建方法,用来创建值相同的字典

例如：**ddict={}.fromkeys(("x","y"),-1)**

遍历字典

```
>>> dic2
```

```
{'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> for k in dic2:
...     print k
...
age
2
Mingzi
>>> for k in dic2:
...     dic2[k]
...
24
'shabi'
'yuanchun.yao'
```

在字典中增加一个值，修改一个值。（字典是无序的，不一定会在最后面增加）

```
>>> dic2
{'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> dic2["zengjia"]="add"
>>> dic2
{'zengjia': 'add', 'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> dic2["zengjia"]="++"
>>> dic2
{'zengjia': '++', 'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
```

在字典中删除

```
>>> dic2
{'zengjia': '++', 'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> del(dic2["zengjia"])
>>> dic2
{'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
```

使用 **pop** 删除并返回（弹出）

```
>>> dic2
{'age': 24, 2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> a=dic2.pop("age")
>>> a
24
>>> dic2
{2: 'shabi', 'mingzi': 'yuanchun.yao'}
```

###可以指定没有 **key** 时弹出的默认值，如果没有这个 **key** 且没有指定默认弹出值，会报 **keyerror**。

```
>>> dict1.pop(100,"100")
'100'
```

清空，删除字典

```
>>> dic2
{2: 'shabi', 'mingzi': 'yuanchun.yao'}
>>> dic2.clear()
>>> dic2
{}
>>> del(dic2)
>>> dic2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'dic2' is not defined
```

使用 **dict.fromkeys()**。来创建相同值得字典

```
>>> dict1=dict.fromkeys(seq,"ab")
>>> dict1
{1: 'ab', 2: 'ab', 3: 'ab', 4: 'ab', 5: 'ab'}
>>>
```

使用 **get()**返回键值的对应值。没有的话返回默认值

```
>>> dict1.get(1,"error")
'ab'
>>> dict1.get(12,"error")
'error'
```

判断字典中是否含有某个 **key**。建议使用 **in** 和 **not in**

```
>>> dict1
{1: 'ab', 2: 'ab', 3: 'ab', 4: 'ab', 5: 'ab'}
>>> dict1.has_key(1)
True
>>> 2 not in dict1
False
>>> 2 in dict1
True
```

返回字典键值对应的元组的列表

```
>>> dict1.items()
[(1, 'ab'), (2, 'ab'), (3, 'ab'), (4, 'ab'), (5, 'ab')]
```

返回字典中 **key** 的列表

```
>>> dict1.keys()
[1, 2, 3, 4, 5]
```

返回字典中 **values** 的列表

```
>>> dict1.values()
['cd', 'cd', 'cd', 'cd', 'cd', '100']
```

Iter*:迭代器。通过 **list** 函数可以访问。效率高。

```
>>> keys=dict1.iterkeys()
>>> list(keys)
[1, 2, 3, 4, 5]
>>> items=dict1.iteritems()
>>> list(items)
[(1, 'ab'), (2, 'ab'), (3, 'ab'), (4, 'ab'), (5, 'ab')]
>>> values=dict1.itervalues()
>>> list(values)
['ab', 'ab', 'ab', 'ab', 'ab']
```

若 **key** 存在返回 **value**，不存在，设置添加键值的到字典，并返回默认值。同 **set**

```
>>> dict1.setdefault(100,"100")
'100'
>>> dict1
{1: 'ab', 2: 'ab', 3: 'ab', 4: 'ab', 5: 'ab', 100: '100'}
>>> dict1.setdefault(1,"100")
'ab'
```

添加一个字典到另一个，重复项覆盖。

```
>>> dict2=dict.fromkeys(seq,"cd")
>>> dict2
{1: 'cd', 2: 'cd', 3: 'cd', 4: 'cd', 5: 'cd'}
>>> dict1.update(dict2)
>>> dict1
{1: 'cd', 2: 'cd', 3: 'cd', 4: 'cd', 5: 'cd', 100: '100'}
```

9.流程控制

9.1 if-else 语句

If elif else 语句，语法如下(**if** 语句下一行缩进的 **tab** 或者四个空格 为本次 **if** 有效内容)

```
def fun():
    return 0

if fun():
    print "ok"
else:
    print "no"
```

```
if 2 >= 2:
    print "2>=2"
if 2 >= 1:
    print "2>=1"
else:
    print "false"
```

第一个条件匹配的时候，**elif** 就不会执行了，这点它有 **else** 的特性。

```
if 3>2:
    print "3>2"
elif 3<2 :
    print "3<2"
else:
    print "i do not know"
```

还有就是 **and or not** 逻辑控制语句。夹杂在表达式，**if** 语句中使用。

9.2 for 循环

语法格式如下：

```
for x in "abcdef":
    print x
```

执行效果如下：

```
E:\python>python for.py
a
b
c
d
e
f
```

或者

```
for x in [1,2,3,4]:
    print x
```

执行效果如下：

```
E:\python>python for.py
1
2
3
4
```

使用 **range(l,j,k)**：i 初始值，j 终止值，k 步长。(j 是终止值，但是不包含这个值，一般取

终止值加 **1**，和切片中的终止值类似都不包含)

```
for x in range(10):
```

```
    print x
```

执行效果如下：

```
E:\python>python for.py
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
for x in range(1,11,1):
```

```
    print x
```

执行效果如下：

```
E:\python>python for.py
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
for x in range(1,11,2):
```

```
    print x
```

执行效果如下：

```
E:\python>python for.py
```

```
1
```

```
3
```

```
5
```

```
7
```

```
9
```

9.3 for 循环和 if 语句

for 和 **if** 搭配使用。

```
for x in range(1,11,2):  
    if x>3:  
        print x
```

执行效果如下：

```
E:\python>python for.py  
5  
7  
9
```

使用元组长度作为 **range** 参数。

```
t=(1,2,3,4,5)  
for x in range(len(t)):  
    if x>3:  
        print x
```

执行效果如下：

```
E:\python>python for.py  
4
```

同样列表也可以。自己试一试。

使用 **for** 循环遍历字典。方法如下：

```
遍历 key 和 value  
d={1:11,2:22,3:33}  
for x in d:  
    print x  
    print d[x]
```

执行效果如下：

```
E:\python>python for.py  
1  
11  
2  
22  
3  
33
```

或者使用 **items** 函数来遍历

```
d={1:11,2:22,3:33}  
print d.items()  
for x,y in d.items():
```



```
print x
print y
```

遍历效果如下：

```
E:\python>python for.py
[(1, 11), (2, 22), (3, 33)]
1
11
2
22
3
33
```

9.4 for 循环和 else, 以及 break, continue, pass

下列代码中,

1. **for** 可以和 **else** 搭配, **for** 语句执行完了自动执行 **else** 语句。
2. **if** 语句不支持空语句, 必须用 **pass** 关键字占位置, 俗称代码桩
3. **continue** 语句或停止当前循环, 不执行后面的语句, 执行下一个循环。
4. **break** 语句会跳出整个循环。
5. **exit()**会结束整个程序。

```
Import time
s="ABCDEFG"
t=(1,2,3,"b","c")
l=[4,5,6,"d","e","f"]
d={1:11,2:22,3:33}
```

```
for x in range(1,11,2):
    time.sleep(1)
    print x
    if x>2:
        pass
    if x == 7:
        continue
    print "continue"
    if x == 5:
        break
    if x == 1:
        exit()
else:
    print "else ending"
```

9.5 while & else

基本上和 **for** 差不多，只不过可以按照条件判断来执行。如下：

```
import time
x=raw_input("please enter a letter")
while x != "":
    time.sleep(0.1)
    print "haha"
    if x == "":
        break
    if x=="q":
        exit()
    if x=="c":
        continue
    if x=="q":
        exit()
    x=raw_input("please enter a letter")
```

10.函数

10.1 参数及书写规则

注意字母小写，使用驼峰标识.格式如下：

需要注意的是如果不写 **return 1** 默认返回空值。那么下面的 **if** 语句就无法执行了。

```
def add():
    a=int(raw_input("please enter a number"))
    b=int(raw_input("please enter a number"))
    c=a+b
    print c
#    return 1
```

```
if add():
    print "ok"
```

```
def machine(x=3,y="goushi"):
    print 'buy ',x,'$',y,'icecreame'
```

```
machine(3,"cholo") #正常赋值
```

```
machine() #按照默认值赋值
machine(5) #默认赋值给第一个参数
machine("goushi") #goushi 也默认赋值给第一个参数
```

给参数默认赋值的时候必须从右到左。比如：

```
def machine(y="goushi"):
    print 'buy ',x,'$',y,'icecreame'
```

下面的例子就不行，因为参数赋值是从左到右的。

```
def machine(x=3):
    print 'buy ',x,'$',y,'icecreame'
```

你传一个值给他，他会赋值给 **x,y** 就没有值了，所以还是需要传送两个值，这样定义默认值就没有意义了。

下面这种就可以，调换参数位置。语句调用的时候是不在意参数的位置的。

```
def machine(y,x=3):
    print 'buy ',x,'$',y,'icecreame'
machine("goushi")
```

这样就会报错，因为赋值顺序是从左到右。系统会把参数给 **X**，这样 **Y** 就没有值了。

```
def machine(x=3,y):
    print 'buy ',x,'$',y,'icecreame'
machine("goushi")
```

10.2 局部变量，全局变量，*t 元组参数传递，格式化打印

使用 **global** 关键字定义全局变量，注意局部变量的作用域，当存在相同变量名称时，局部变量优先级高于全局变量。

***t**：元组可以当做函数参数传递，传递的是元组的内容，而不是元组本身。

Print 格式化打印。

```
def dayin(x,y):
    c=x
    d=y
    print "%s : %s " % (x,y) : 打印 xy，%代表获取 xy 的值。
    global b ###先声明，在赋值，不然。。。。
    b=200
```

```
a=100
t=(3,4)
dayin(*t)
dayin(1,2)
print a
print b
```

10.3 使用字典作为参数，处理冗余参数

字典作为参数,参数顺序只要和字典里面的 **key** 相同，顺序可以随便写。字典元素个数不能比函数参数个数多。

```
def print1(x,y):  
    print x  
    print y  
d={"x":1,"y":2}  
print1(**d)
```

处理冗余参数，注意后面传递给字典的参数不要和前面的默认参数重名，会报错的。

```
def zidian(x,*arg,**keyArg):  
    print x  
    print arg  
    print keyArg
```

`zidian(1,2,3,4,5,6,z=7,y=8)` #前面的 1 赋值给 x,然后 23456 赋值给 arg 作为元组，剩下的给 keyArg 当做字典。

>>>>>>输入参数的时候还必须要安装默认参数的顺序。比如在 `y=8` 之后在加一个 5,就会报错。

10.4 lambda 匿名函数

```
f=lambda x,y:x*y
```

其中 x,y 是参数 x*y 是返回值

reduce():输入两个参数，第一个参数为函数，第二个为 **list**。第一次取 **list** 里面的两个值，给函数，作为函数参数。

第二次将函数自己的一个返回值和接着取出 **list** 里面的一个值，作为新的参数，传给函数。

```
>>> l=range(1,11)  
>>> reduce(lambda x,y:x+y,l)    ###实践证明，传三个参数，还真不行  
55
```

10.5 使用字典构建 python 的 switch 语句

字典自动对传过来的参数进行判断，相当于 **switch** 语句。

```
def jia(x,y):  
    return x+y  
def jian(x,y):  
    return x-y
```

```
def chen(x,y):
    return x*y
def chu(x,y):
    return x/y
operator={"+":jia,"-":jian,"*":chen,"/":chu}
def calculator(x,o,y):
    print operator[o](x,y)

calculator(1,"+",2)
```

10.6 内建函数

绝对值

```
>>> abs(-123)
123
```

最大最小值

```
>>> l=[1,23,4]
>>> min(l)
1
>>> max(l)
23
```

求序列长度

```
>>> len(l)
3
```

除法，求模

```
>>> divmod(5,2)
(2, 1)
```

次幂，以及次幂的模

```
>>> pow(2,3)
8
```

```
>>> pow(2,3,5)
3
```

四舍五入

```
>>> round(1)
1.0
>>> round(1.5)
```

2.0

```
>>> round(1.456464165,3)
1.456
```

测试某一个函数是否存在。

```
>>> callable(min)
True
```

属于某种类型

```
>>> isinstance(12,int)
True
```

比较大小

```
>>> cmp(1,2)
-1
>>> cmp(1,1)
0
>>> cmp(2,1)
1
```

返回序列，返回序列对象

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> xrange(10)
xrange(10)
```

类型转化函数

查看类型：**type**

```
>>> l=range(10)
>>> type(l)
<type 'list'>
```

转化成 **int** 型

```
>>> a=12l
>>> type(a)
<type 'long'>
>>> int(a)
12
>>> b=int(a)
>>> type(b)
<type 'int'>
>>> b=a
>>> type(b)
<type 'long'>
```

转化为 **long** 类型

```
>>> b=33
>>> type(b)
<type 'int'>
>>> a=long(b)
>>> type(a)
<type 'long'>
```

转化为 **float** 类型

```
>>> a=12
>>> type(a)
<type 'int'>
>>> b=float(a)
>>> type(b)
<type 'float'>
```

转化为复数

```
>>> a=12
>>> type(a)
<type 'int'>
>>> b=complex(a)
>>> type(b)
<type 'complex'>
```

还有：

```
str()
list()
tuple()
hex():十六进制
oct():八进制
chr()
ord()
```

capitalize 首字母大学：

```
>>> a="hello"
>>> a.capitalize()
'Hello'
```

字符串替换

替换所有的。

```
>>> b="java sun hello java"
>>> b.replace("java","haha")
'haha sun hello haha'
```

从左到右，按个数替换（是一共替换多少个，不是第几个）

```
>>> b.replace("java","haha",1)
'haha sun hello java'
```

截取：默认的是以空格截取

```
>>> a="study java happy"
>>> a.split()
['study', 'java', 'happy']
```

可以指定截取符号和截取次数：

```
>>> b="yuanchun,xiaoqiang,amao,xiaojianren"
>>> b.split(',')
['yuanchun', 'xiaoqiang', 'amao', 'xiaojianren']
>>> b.split(',',2)
['yuanchun', 'xiaoqiang', 'amao,xiaojianren']
>>> b.split(',',1)
['yuanchun', 'xiaoqiang,amao,xiaojianren']
>>> b.split(',',0)
['yuanchun,xiaoqiang,amao,xiaojianren']
```

序列长度

```
>>> l=[1,2,"abc",4]
>>> len(l)
4
>>> len(l[2])
3
```

最大值

```
>>> l=[2,3,4,56,3]
>>> max(l)
56
>>> min(l)
2
```

条件限制函数 **filter**：

```
def f(x):
    if x>3:
        return x
```

```
l=range(1,10)
print filter(f,l)
```


效果：

```
E:\python>python test1.py
```

```
[4, 5, 6, 7, 8, 9]
```

使用 **filter** 加 **lambda** 模式。

```
>>> l=range(1,11)
```

```
>>> filter(lambda x:x%2==0,l)
```

```
[2, 4, 6, 8, 10]
```

PS **filter** 只能进行条件判断, 就算将条件语句变成表达式, 它也只能判断表达式的布尔值, 从而决定返回值, 而不是单纯的返回表达式的值, 这点和 **map** 不同。

压缩函数 **zip**, 短的序列不压缩

```
>>> l1=[1,2,3]
```

```
>>> l2=[4,5,6]
```

```
>>> zip(l1,l2)
```

```
[(1, 4), (2, 5), (3, 6)]
```

```
>>> l3=[7,8]
```

```
>>> zip(l1,l2,l3)
```

```
[(1, 4, 7), (2, 5, 8)]
```

压缩函数 **map**, 功能和 **zip** 类似, 但是可以通过定义默认值, 来压缩短的序列。

```
>>> map(None,l1,l2,l3)
```

```
[(1, 4, 7), (2, 5, 8), (3, 6, None)]
```

Map 还可以和 **filter** 类似, 做函数处理功能。(这种方法很不错)

```
>>> a=[1,2,4]
```

```
>>> b=[4,5,6]
```

```
>>> def mf(x,y):
```

```
...     return x*y
```

```
...
```

```
>>> map(mf,a,b)
```

```
[4, 10, 24]
```

Map 和 **lambda** 搭配, 实现函数处理功能。

```
>>> l=range(1,11)
```

```
>>> map(lambda x:x*2,l)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Reduce 处理连加, 阶乘：

```
>>> reduce(lambda x,y:x+y,l)
```

```
5050
```

两种看起来很简洁的写法。

```
>>> foo=range(1,11)
>>> print [x*2 + 10 for x in foo]
[12, 14, 16, 18, 20, 22, 24, 26, 28, 30]

>>> print [x for x in foo if x%3 ==0 ]
[3, 6, 9]
```

11.引入模块

Python 中的模块就是另一个.py 或者.pyc,.pyo 文件。引入的时候可以调用里面的内容。

对于模块里面的其他执行语句, 需要对__name__变量进行判断, 如果是他的值是__main__, 那么就是直接调用这个脚本, 需要执行相应语句。如果__name__的值是其他名称, 那么它就是作为一个模块被调用, 不需要执行本身的非函数语句。如下:

文件 1 : module.py

```
def add(x,y):
    return x+y

if __name__ == "__main__":
    print add(1,2)
```

文件 2 : test1.py

```
import module
print module.add(3,4)
```

执行效果:

```
E:\python>python test1.py
7
```

PS:查找模块的时候会优先查找当前目录, 再到配置目录查找。需要注意模块名称不要冲突。

其他引入方法:

```
import module
#import module as mokuai
#from module import add
```

```
print module.add(3,4)
#print mokuai.add(5,6)
```

```
#print add(7,8)
```

引入包 python 模块可以按照目录组织为包

创建一个包的步骤是:

1. 建立一个名字为包名字的文件夹。
2. 在文件夹下创建一个__init__.py 的文件。空的就好
3. 根据需要在该文件夹下存放脚本文件, 一边以扩展及子包。

4. Import pack.m1, pack.m2,pack.m3

12.正则表达式

12.1 元字符。

[]:中括号。用来表示字符集。元字符在字符集中不起作用。补集匹配不在范围内的字符。使用^[^]模式表示非。

^:匹配行首。

\$:匹配行尾。

```
>>> import re
>>> s="abc"
>>> re.findall(s,"abcddddabc")
['abc', 'abc']
或者如下：没有发现什么区别
>>> import re
>>> s=r"abc"
>>> re.findall(s,"abcddddabc")
['abc', 'abc']
```

```
>>> st="top tip taq twq teq"
>>> res="t[oi]p"
>>> re.findall(res,st)
['top', 'tip']
& 非
>>> res="t[^oi]q"
>>> re.findall(res,st)
['taq', 'twq', 'teq']
```

开头匹配

```
>>> s="hello world,hello boy"
>>> r="^hell"
>>> re.findall(r,s)
['hell']
>>> r="^hello"
>>> re.findall(r,s)
['hello']
```

测试元字符在字符集中不起作用

```
>>> r="t[abc$]"
>>> re.findall(r,"tc")
['tc']
```

```
>>> re.findall(r,"tb")
['tb']
>>> re.findall(r,"t$")
['t$']
&
>>> r="t[abc^]"
>>> re.findall(r,"ta")
['ta']
>>> re.findall(r,"t^")
['t^']
&
>>> r="x[a-zA-Z0-9]"
>>> re.findall(r,"xl,xd,x.,xx")
['xl', 'xd', 'xx']
```

12.2 字符集&多次匹配

\:反斜杠搭配表示特殊意义，或者作为转义字符。

\d:表示十进制数，**0-9**，相当于**[0-9]**

\D:表示非十进制数，非**0-9**，相当于**[^0-9]**

\s:表示空白字符，相当于**[\t\n\r\v]**

\S:表示非空白字符，相当于**[^\t\n\r\v]**

\w:表示任意数字字母，相当于**[0-9a-zA-Z]**

\W:表示任意非数字字母，相当于**[^0-9a-zA-Z]**

.：匹配任何字符（**\n** 除外）

*****：匹配**0**次或者多次前面出现的正则表达式

^：匹配字符串起始部分

\$：匹配字符串终止部分

+：匹配**1**次或多次前面出现的正则表达式

?：匹配**0**次或都**1**次前面出现的正则表达式

验证**\d{8}**:表示重复**8**次。

```
>>> import re
>>> r="^010-\d{8}$"
>>> re.findall(r,"010-12345678")
['010-12345678']
>>> re.findall(r,"010-123456789")    (超过 8 位不符合正则匹配，所以输出为空)
[]
```

验证**+**:表示匹配一次或者多次。

```
>>> import re
```

```
>>> r="^\\d{3}-+\\d{8}$"    (前面的-出现一次或多次)
>>> re.findall(r,"121-12345678")
['121-12345678']
>>> re.findall(r,"121----12345678")
['121----12345678']
```

转移字符，屏蔽特殊意义。

```
>>> r="^010-\\+12345678$"
>>> re.findall(r,"010-+12345678")
['010-+12345678']
```

验证*：可以出现 0 次或多次。

```
>>> r="^010-*12345678$"
>>> re.findall(r,"01012345678")
['01012345678']
>>> re.findall(r,"010--12345678")
['010--12345678']
>>>
```

```
>>> r="^010-*\\d{8}$"
>>> re.findall(r,"010-12345678")
['010-12345678']
>>> re.findall(r,"01012345678")
['01012345678']
```

验证?：可以出现 0 次或 1 次。

```
r="^\\d{3}-?\\d{8}$"
>>> re.findall(r,"010-12345678")
['010-12345678']
>>> re.findall(r,"01012345678")
['01012345678']
>>> re.findall(r,"010--12345678")
[]
```

{m,n}:至少重复 m 次，最多出现 n 次。

{0,}: 相当于* {1,}: 相当于+ {0,1}: 相当于?

PS:最好不要使用这种方式。

```
>>> r="^\\d{1,3}-?\\w{5}$"
>>> re.findall(r,"12bcdef")
['12bcdef']
>>> re.findall(r,"12--abcde")    (超过 1 次，输出空)
[]
>>> re.findall(r,"12-abcde")
['12-abcde']
```

12.3 将正则编译成类

直接调用 **findall** 方法。

```
>>> rc="^\\d{3,4}-?\\d{8}$"
>>> r=re.compile(rc)
>>> r
<_sre.SRE_Pattern object at 0x01C14920>
>>> r.findall("123-12345678")
['123-12345678']
>>> r.findall("1234-12345678")
['1234-12345678']
```

编译正则的时候，可以加参数实现一些属性
如下忽略大小写

```
>>>
>>> raA.findall("abc")
['abc']
>>> raA.findall("aBc")
['aBc']
```

或者是定义规则的时候用土方法。

```
>>> r=r"[aA][bB][cC]"
>>> re.findall(r,"abc")
['abc']
>>> re.findall(r,"aBc")
['aBc']
&&
>>> r="[aA][bB][cC]"
>>> re.findall(r,"abc")
['abc']
```

反斜杠的麻烦

— 反斜杠的麻烦

- 字符串前加 "r" 反斜杠就不会被任何特殊方式处理

字符	阶段
\\section	要匹配的字符串
\\section	为 re.compile 取消反斜杠的特殊意义
"\\\\section"	为"\\section"的字符串实值(string literals)取消反斜杠的特殊意义

match 方法。查找字符串的头部是不是有匹配字段。有的话就返回一个 **match** 对象。
如果正则不在字段的头部的话，什么都不返回。

```
>>> re.match("abc", "abc hello")
```

```
<_sre.SRE_Match object at 0x004BEA30>
>>> re.match("abc","hello abc")
>>>
```

search：方法。在整个字段中匹配，返回的也是 **math** 对象。

```
>>> re.search("ab","abc hellow")
<_sre.SRE_Match object at 0x00524F38>
>>> re.search("ab","hellow abc")
<_sre.SRE_Match object at 0x004BEA30>
```

finditer：方法。在整个字段中匹配，返回迭代器对象。

再对这个对象使用 **next** 方法，返回一个 **match** 对象。

那么就可以使用 **group** 方法查看返回的是什么。

```
>>> re.finditer("ab","abcd")
<callable-iterator object at 0x02361FF0>
>>> x=re.finditer("ab","abcd")
>>> g=x.next()
>>> g.group()
'ab'
```

Match 对象可以使用的方法。

group 方法。返回查找的对象是什么（正则是什么）

```
>>> re.finditer("ab","abcd")
<callable-iterator object at 0x02361FF0>
>>> x=re.finditer("ab","abcd")
>>> g=x.next()
>>> g.group()
'ab'
```

start 方法。返回查找字符串在 **math** 对象中出现的第一个位置。

end 方法。返回查找字符串在 **math** 对象中出现的最后一个位置。

span 方法。返回查找字符串在 **math** 对象中出现的第一个和最后一个位置，已元组的形式返回。

```
>>> x=re.search("abc","defabcabc")
>>> x
<_sre.SRE_Match object at 0x004BEA30>
>>> x.start()
3
>>> x.end()
6
>>> x.span()
(3, 6)
```

正则的 **split** 方法，为了使字符串可以使用正则匹配多个分隔符。

```
>>> re.split(r"[\+\\-\\*\\/]", "1+2-3*4/5")
['1', '2', '3', '4', '5']
```

正则的 **sub** 与 **subn** 方法，相当于字符串的 **replace** 方法。**Subn** 会显示匹配了多少次。

```
>>> re.sub(r"abc", "testabc", "abc def abc def abc")
'testabc def testabc def testabc'
>>> re.subn(r"abc", "testabc", "abc def abc def abc")
('testabc def testabc def testabc', 3)
```

12.4 查看某个类函数&变量的方法。

dir(re)：查看 **re** 这个类的方法以及变量。

```
>>> dir(re)
['DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE', 'S', 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X', '_MAXCACHE', '__all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__version__', '_alphanum', '_cache', '_cache_repl', '_compile', '_compile_repl', '_expand', '_locale', '_pattern_type', '_pickle', '_subx', 'compile', 'copy_reg', 'error', 'escape', 'findall', 'finditer', 'match', 'purge', 'search', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn', 'sys', 'template']
```

12.5 正则的其他编译属性&分组

S：使正则表达式，可以匹配换行符在内的所有字符

```
>>> r=r"yuanchun.yao"
>>> re.findall(r,"yuanchun.yao")
['yuanchun.yao']
>>> re.findall(r,"yuanchunsyao")
['yuanchunsyao']
>>> re.findall(r,"yuanchun\yao")
['yuanchun\yao']
>>> re.findall(r,"yuanchun\nyao")
[]
>>> re.findall(r,"yuanchun\nyao",re.S)
['yuanchun\nyao']
```

I：是正则表达式对大小写不敏感

```
>>> r=r"abc"
>>> re.findall(r,"abc ABC abC",re.I)
['abc', 'ABC', 'abC']
```

L：按照本地环境，匹配本地语法。

M:多行匹配，对[^],^{\$}有影响，可以匹配多行的行首，行尾。

```
>>> s="""
... hello csvt
... hello csvt hello
... csvt hehe
... """
>>> import re
>>> r=r"^csvt"
>>> re.findall(r,s)
[]
>>> re.findall(r,s,re.M)
['csvt']
```

但是当不是匹配行首行尾的时候，貌似没有什么影响。

```
>>> re.findall(r,s,re.M)
['csvt', 'csvt', 'csvt']
>>> re.findall(r,s)
['csvt', 'csvt', 'csvt']
```

X:可以定义一个多行的正则表达式，防止正则表达式过长。

```
['shabi ', 'neng', 'hao']
>>> tel="""
... \d{3,4}
... -?
... \d{8}
... """
>>> re.findall(tel,"010-44245100")
[]
>>> re.findall(tel,"010-44245100",re.X)
['010-44245100']
>>> tel
'\n\d{3,4}\n-?\n\d{8}\n'
```

分组：方便定义局部语句进行与或非等操作，并且在查询的时候可以只返回分组中的语句。

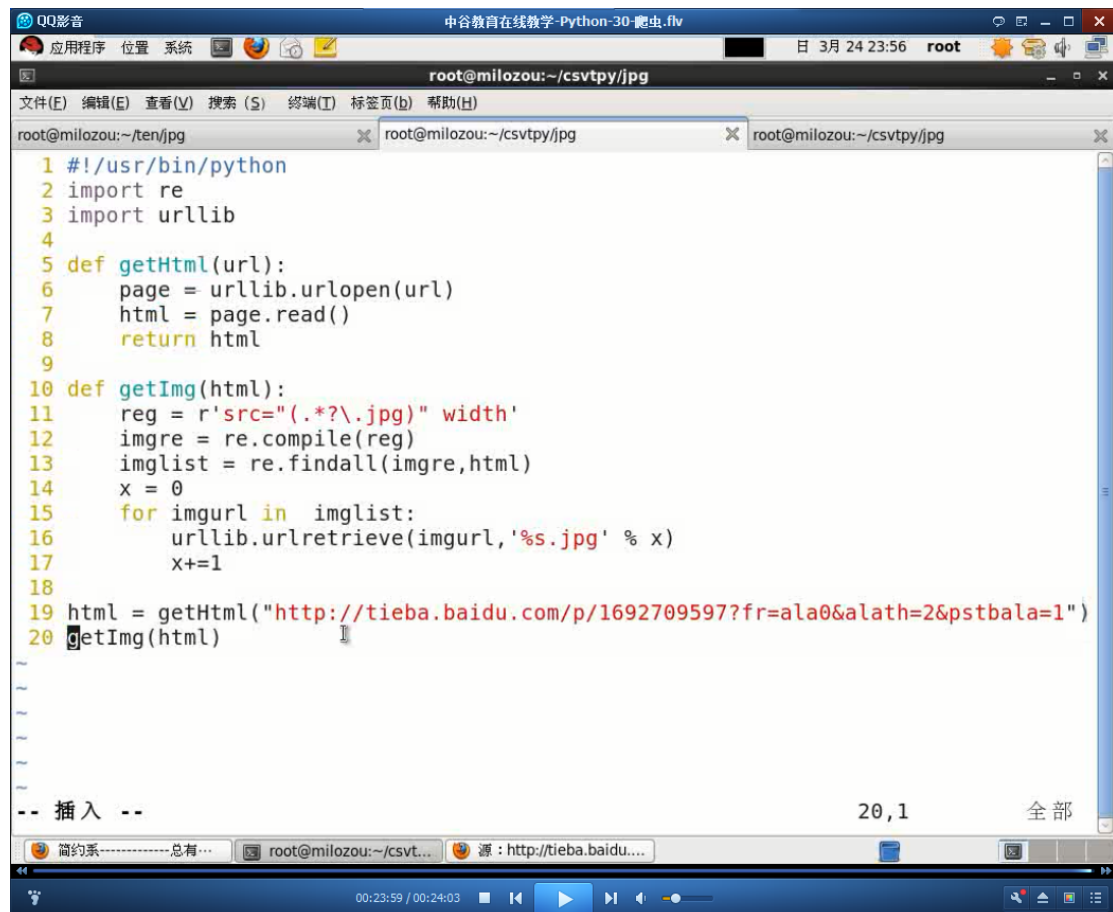
分组格式：()

```
>>> email=r"\w{3}@\w+(\.com|\.cn)"
>>> re.match(email,"www@csvt.com")
<_sre.SRE_Match object at 0x022938E0>
>>> re.findall(email,"www@csvt.com")
['.com']
>>> re.findall(email,"www@dlkajf.cn")
['.cn']
```

只返回分组的语句。

```
>>> s="""
... skdjf hello src=csvt yes jasdfkj
... klsdfj src=132 ksdfj lkjd
... src=234 yes
... lsdjf src=python yes ksa
... """
>>> r=r"hello src=.+ yes"
>>> re.findall(r,s)
['hello src=csvt yes']
>>> r=r"hello src=(.+) yes"
>>> re.findall(r,s)
['csvt']
```

12.7 爬虫下载网页图片



13.拷贝

13.1 浅拷贝

浅拷贝是对对象引用的拷贝。它保证对象整体的不变性，如果源对象内部增加了一个变量，那么 **copy** 出来的对象是不会跟着增加的。因为我 **copy** 的时候引用就指向这几个对象而已，不会跟着增加的。对于内部可变的部分，引用指向的是内部可变部分的整体，如果内部可变部分内容变了，**copy** 对象也会跟着变。对于内部不可变部分，源对象变了，**copy** 出来的对象不会跟着变，因为引用没有指向变动出来的新地址。依旧指向的老地址，所以不变。

```
>>> import copy
>>> a=[1,2,['a','b']]
>>> b=a
>>> b
[1, 2, ['a', 'b']]
>>> id(a)
36086848
>>> id(b)
36086848
>>> c=copy.copy(a)
>>> c
[1, 2, ['a', 'b']]
>>> id(c)
36072864
内部新增了一个元素。
>>> a.append("c")
>>> a
[1, 2, ['a', 'b'], 'c']
>>> b
[1, 2, ['a', 'b'], 'c']
>>> c
[1, 2, ['a', 'b']]
内部可变的变动。
>>> a[2].append("c")
>>> a
[1, 2, ['a', 'b', 'c'], 'c']
>>> b
[1, 2, ['a', 'b', 'c'], 'c']
>>> c
[1, 2, ['a', 'b', 'c']]
内部不可变的变动。
>>> a[1]=5
>>> a
[1, 5, ['a', 'b', 'c'], 'c']
```

```
>>> b
[1, 5, ['a', 'b', 'c'], 'c']
>>> c
[1, 2, ['a', 'b', 'c']]
```

13.2 深拷贝

深拷贝是对对象的完全拷贝完全独立出来，就连内部可变对象内部的引用也细分出来。

```
>>> a=[1,2,['a',"b"]]
>>> import copy
>>> d=copy.deepcopy(a)
>>> d
[1, 2, ['a', 'b']]
>>> a.append("c")
>>> a
[1, 2, ['a', 'b'], 'c']
>>> d
[1, 2, ['a', 'b']]
>>> a[2].append("c")
>>> a
[1, 2, ['a', 'b', 'c'], 'c']
>>> d
[1, 2, ['a', 'b']]
```

14.文件操作

打开文件可以使用 **open** 或者 **file** 函数。不带参数默认是只读模式。

使用文件对象可以调用 **read,readline,readlines,next&writte,wirtelines&seek**。

意思分别为 **read** 读全部, **readline** 读一行(到文件结尾时读取的位空), **readlines** 读多行(默认读全部), **next** 读一行(到文件结尾时读取

```
>>> fo.next()
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

StopIteration)。

同时他们都会移动文件指针到读取的当前位置。

writte 写一句话, **wirtelines** 写多句话(内部可以放列表)。

PS:read 方法和 **next** 不能混用。会报错。

seek 方法用来移动文件指针。 **fileObject.seek**(偏移量, 选项):选择为以下值时

0 : 文件头。 **2** : 文件尾。 **1** : 当前位置。

fileObject.flush : 将缓冲区文件写到硬盘。不用 **close**, 就可以改变文件状态。

fileObject.close:关闭文件。

文件打开模式。

r:只读 **r+**:读写 **w**:写入, 先删除原文件, 再重新写入, 如果没有这个文件会自动创建。

w+:读写, 先删除原文件, 再重新写入, 如果没有这个文件会自动创建。

a:追加写。在文件末尾, 重新写入, 如果没有这个文件会自动创建。

a+:追加读写。在文件末尾, 重新写入, 如果没有这个文件会自动创建。

b:打开二进制文件。

U:支持多种换行符。

```
>>> fo = file("canshu.py")
>>> fo
<open file 'canshu.py', mode 'r' at 0x0203D230>
>>> fo.read()
'def print1(x,y):\n\tprint x\n\tprint y\n\nd={"x":1,"y":2}\n\tprint1(**d)\n\ndef z
idian(x,*arg,**keyArg):\n\tprint x\n\tprint arg\n\tprint keyArg\n\n#zidian(1,2,3
,4,5,6,z=7,y=8)'
>>> fo.read()
"
>>> fo.seek(0,0)
>>> fo.readline()
'def print1(x,y):\n'
>>> fo.readlines()
['\tprint x\n', '\tprint y\n', 'd={"x":1,"y":2}\n\tprint1(**d)\n', '\n', 'de
f zidian(x,*arg,**keyArg):\n', '\tprint x\n', '\tprint arg\n', '\tprint keyArg\n
', '\n', '#zidian(1,2,3,4,5,6,z=7,y=8)']
>>> fo.seek(0,0)
```

例题：文件查找。

cat a.txt

hello haha

haha nihao hello nihao hello

hello wode hello nide tade nimende hello

while 写法：**PS**:记得给脚本加两个参数哦。一个是文件名称，一个是关键词

```
import sys
```

```
import os
```

```
import re
```

```
if len(sys.argv) == 1:
```

```
    print "no args"
```

```
    sys.exit(1)
```

```
fileName=sys.argv[1]
```

```

keyWord=sys.argv[2]
fileObj=open(fileName,"r+")
str=fileObj.readline()
count=0
while str != "":
    list = re.findall(keyWord,str)
    cc = len(list)
    if cc != 0:
        count = count + cc
    str=fileObj.readline()

```

```

fileObj.close()
print count
for 写法：
import re
fp = open("a.txt","r")
count = 0
for s in fp.readlines():
    li = re.findall("hello",s)
    if len(li) > 0:
        count = count + len(li)

```

```

print count
fp.close()

```

PS :必须使用 **for** 写法必须使用 **readlines** 方法, 因为 **readlines** 方法返回的是一个列表。
For 循环 会循环列表元素个数次。如果使用 **readline** 就只读一次, 而且它会把当前字符串作为一个序列来处理, 一次只读一个字节, 会循环字节个数次。**read** 方法和 **readline** 类似。

打开文件 **a**, 将其中的 **hello** 替换成 **csvt**。

```

fp1=open("a.txt","r")
fp2=open("a1.txt","w+")

for s in fp1.readlines():
    fp2.write(s.replace("hello","csvt"))

fp1.close()
fp2.close()

```

15.OS 模块

15.1 常用 os 函数

mkdir(path,[mode=0777]) :新建文件夹

makedirs(name,mode=511) :新建一系列文件夹

rmdir(path) :删除文件夹

removedirs(path) :删除一系列文件夹

listdir(path) :显示当前路径下文件和文件夹

getcwd() :显示当前路径

chdir(path) :移动路径到制定 path

walk(top,topdown=True,onerror=None) :以列表的形式显示当前路径, 当前路径下文件夹, 以及当前路径下文件。

```
>>> import os
>>> os.getcwd()
'C:\\Users\\yuanchun.yao'
>>> os.chdir("e:\\python")
>>> os.getcwd()
'e:\\python'
>>> os.mkdir("just_test_file")
>>> os.listdir(".")
['a.txt', 'a1.txt', 'add.py', 'canshu.py', 'compile.py', 'count_hello.py', 'download_picture.py', 'for.py', 'fun.py', 'haha', 'hello.py', 'jishu.py', 'just_test_file', 'module.py', 'print.py', 'replace.py', 'replace1.py', 'replace1.pyo', 'szys.py', 'test.py', 'test.txt', 'test1.py', 'test_haha', 'while.py', '__init__.py']
>>> os.makedirs("a1/b1/c1")
>>> os.listdir(".")
['a.txt', 'a1', 'a1.txt', 'add.py', 'canshu.py', 'compile.py', 'count_hello.py', 'download_picture.py', 'for.py', 'fun.py', 'haha', 'hello.py', 'jishu.py', 'just_test_file', 'module.py', 'print.py', 'replace.py', 'replace1.py', 'replace1.pyo', 'szys.py', 'test.py', 'test.txt', 'test1.py', 'test_haha', 'while.py', '__init__.py']
```

15.2 遍历目录

递归方法遍历目录

```
import os
def dirList(path,):
    print path
    os.chdir(path)
```

```

print os.getcwd()
filePath = os.listdir(path)

for p_path in filePath:
    fileName = os.path.join(path,p_path)
    if os.path.isfile(fileName):
        print len(path)*" "+p_path

for p_path1 in filePath:
    fileName = os.path.join(path,p_path1)
    if os.path.isdir(fileName):
        dirList(fileName)

dirList("E:\\eclipse j2SE")

```

walk()函数遍历目录，为了格式的好看，也使用了递归。实际上可以不使用。

```

import os
print ""
path="e:\\python"
print path
def walk_list(path):
    view=os.walk(path)
    listFile=view.next()
    for file in listFile[2]:
        print len(path)*" "+file+"          f"
    for dir in listFile[1]:
        print len(path)*" "+dir+"          d"
        rel_path=os.path.join(path,dir)
        walk_list(rel_path)

walk_list(path)

```

15.异常处理

捕获异常，做相应提示。严禁掩盖异常，最好不要在异常中处理。

```

#coding:utf8
import os
f_stat=0
try:
    fp = open ("asdf.py","r+")
    f_stat=1
    print nihao
except IOError,msg:

```



```

        print "IOError"
except NameError,info:
    print "Nameerror"
finally :
    if f_stat == 1 :
        fp.close()
        print "fp closed"

```

抛出异常

```

if "a" > 5:
    raise TypeError("Error: 'a' must be integer.")

```

执行效果

```

E:\python>python raise.py
Traceback (most recent call last):
  File "raise.py", line 2, in <module>
    raise TypeError("Error: 'a' must be integer.")
TypeError: Error: 'a' must be integer.

```

PS : python 对象大小的比较

1.任何两个对象都可以比较

2.相同类型的对象（实例），如果是数字型（**int/float/long/complex**），则按照简单的大小来比较；如果是非数字型，且类（型）中定义了__cmp__（含__gt__，__lt__等）则按照__cmp__来比较，否则按照地址（**id**）来比较

3.不同类型的对象（实例），如果其中一个比较对象是数字型（**int/float/long/complex**等），则数字型的对象<其它非数字型的对象 如果两个都是非数字型的对象，则按照类型名的顺序比较，如{} < "abc"（按照"dict" < "str"），而"abc" > [1,2], "abc" < (1,2)。

4.对于自定义的类（型）实例，如果继承自基本类型，则按照基本类型的规则比较（1-3）。否则，**old-style class** < **new-style class**, **new-style class** 之间按照类型名顺序比较，**old-style class** 之间按照地址进行比较

5.**bool** 类型是 **int** 的子类，且 **True=1**, **False=0**，比较时按照 1-4 来比较，如 **True > -1**, **True < 4.2**, **True < "abc"**等

上面的回答是针对 **Python2.x**, **3.x** 的有较大的变化，如 **str** 和 **int** 比较时会抛出异常等。

QQ影音 中谷教育在线教学-Python-36-异常处理.flv

CSV 中谷教育

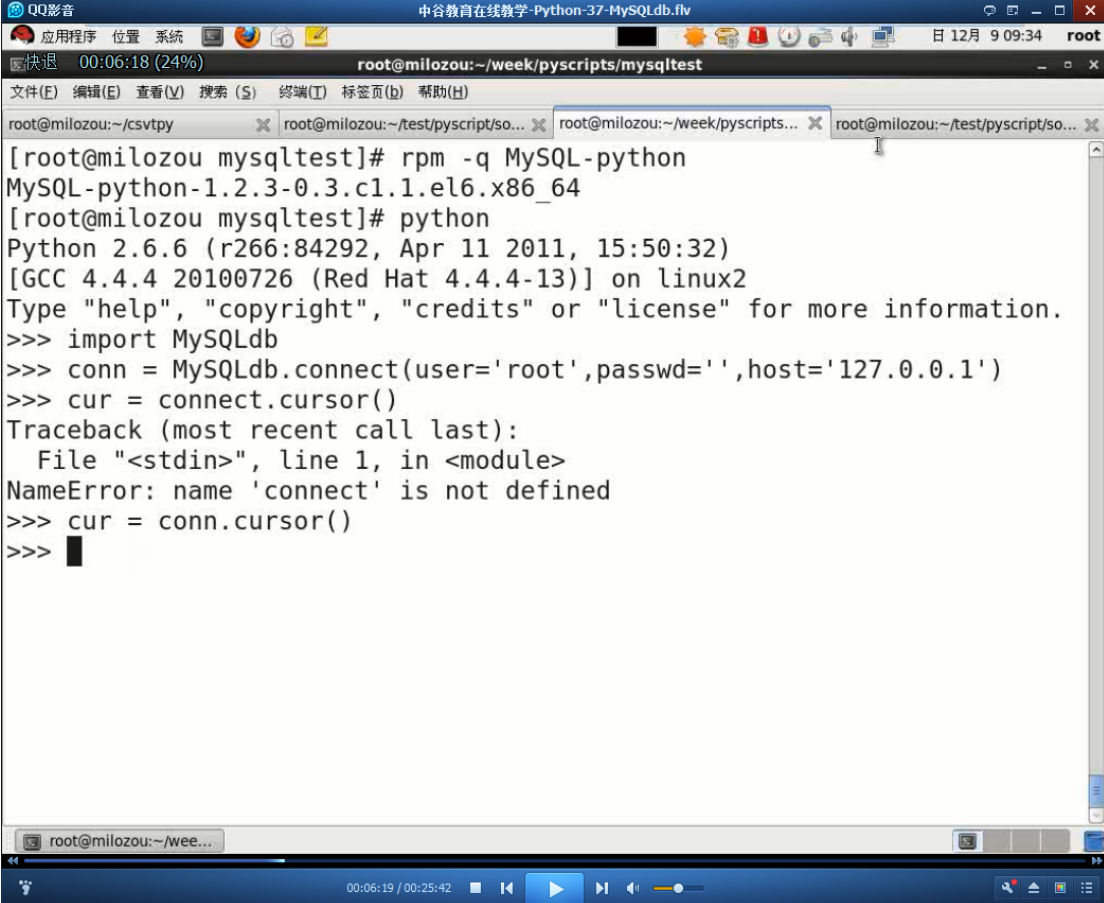
常见的Python异常

异常	描述
AssertionError	assert语句失败
AttributeError	试图访问一个对象没有的属性
IOError	输入输出异常，基本是无法打开文件
ImportError	无法引入模块或者包，基本是路径问题
IndentationError	语法错误，代码没有正确的对齐
IndexError	下标索引超出序列边界
KeyError	试图访问你字典里不存储的键
KeyboardInterrupt	Ctrl+C被按下
NameError	使用一个还未赋予对象的变量
SyntaxError	Python代码逻辑语法出错，不能执行
TypeError	传入的对象类型与要求不符
UnboundLocalError	试图访问一个还未设置的全局变量，基本上是由于另有一个同名的全局变量，导致你以为在访问
ValueError	传入一个不被期望的值，即使类型正确

www.csvt.net

00:22:14 / 00:25:36

16.MySqlLdb



The screenshot shows a terminal window titled "root@milozou:~/week/pyscripts/mysqltest". The terminal output is as follows:

```
[root@milozou mysqltest]# rpm -q MySQL-python
MySQL-python-1.2.3-0.3.c1.1.el6.x86_64
[root@milozou mysqltest]# python
Python 2.6.6 (r266:84292, Apr 11 2011, 15:50:32)
[GCC 4.4.4 20100726 (Red Hat 4.4.4-13)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import MySQLdb
>>> conn = MySQLdb.connect(user='root',passwd='',host='127.0.0.1')
>>> cur = conn.cursor()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'connect' is not defined
>>> cur = conn.cursor()
>>>
```

The error message "NameError: name 'connect' is not defined" indicates that the variable 'connect' is not defined in the current scope, despite the presence of the 'connect' method on the 'conn' object.

```
QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv
应用程序 位置 系统 00:08:42 (33%) root@milozou:~/week/pyscripts/mysqltest
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签页(b) 帮助(H)
root@milozou:~/csvtpty root@milozou:~/test/pyscript/so... root@milozou:~/week/pyscripts... root@milozou:~/test/pyscript/so...
[root@milozou mysqltest]# rpm -q MySQL-python
MySQL-python-1.2.3-0.3.c1.1.el6.x86_64
[root@milozou mysqltest]# python
Python 2.6.6 (r266:84292, Apr 11 2011, 15:50:32)
[GCC 4.4.4 20100726 (Red Hat 4.4.4-13)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import MySQLdb
>>> conn = MySQLdb.connect(user='root',passwd='',host='127.0.0.1')
>>> cur = connect.cursor()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'connect' is not defined
>>> cur = conn.cursor()
>>> conn.select_db(week)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'week' is not defined
>>> conn.select_db('week')
>>> █
```

```
00:08:43 / 00:25:42
QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv
应用程序 位置 系统 00:16:50 (65%) root@milozou:~/week/pyscripts/mysqltest
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签页(b) 帮助(H)
root@milozou:~/csvtpty root@milozou:~/test/pyscript/so... root@milozou:~/week/pyscripts... root@milozou:~/test/pyscript/so...
>>> cur.execute('insert into userinfo()')
>>> cur.execute('insert into userinfo(name,age,gender) value('milo',20,
'm')')
File "<stdin>", line 1
  cur.execute('insert into userinfo(name,age,gender) value('milo',20,
'm')')
SyntaxError: invalid syntax
>>> cur.execute("insert into userinfo(name,age,gender) value('milo',20,
'm')")
1L
>>> sqli = "insert into userinfo(name,age,gender) value(%,%,%s)"
>>> cur.execute(sqli,('csvt',5,'s'))
1L
>>> sqlim = "insert into userinfo(name,age,gender) values(%,%,%s)"
>>> cur.executemany(sqli,[('vt',5,'s'),('c',1,'s'),('s',2,'m')])
3L
>>> █

root@milozou:~/wee...
00:16:51 / 00:25:42
```

```
QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv 日 12月 9 09:50 root
root@milozou:~/week/pyscripts/mysqltest
>>> cur.execute('delete from userinfo where id=20')
>>> cur.execute('delete from userinfo where id=20')
1L
>>> cur.execute('update userinfo set name='boy' where id=16')
File "<stdin>", line 1
    cur.execute('update userinfo set name='boy' where id=16')
>>> cur.execute("update userinfo set name='boy' where id=16")
1L
>>> cur.execute("select * from userinfo")
18L
>>> cur.fetchone()
(1L, 'tom', 30, 'm')
>>> cur.fetchone()
(2L, 'jack', 20, 'm')
>>> cur.fetchone()
(3L, 'alice', 23, 'f')
>>> cur.fetchone()
(4L, 'robbin', 25, 'f')
>>>
```

```
>>> cur.fetchone()
(11L, 'aa', 30, 'm')
>>> cur.scroll(0, 'absolute') 移动指针
>>> cur.fetchone()
(1L, 'tom', 30, 'm')
>>> cur.fetchone()
(2L, 'jack', 20, 'm')
>>>
```

root@milozou:~/wee...


```
QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv 日 12月 9 09:53 root
root@milozou:~/week/pyscripts/mysqltest
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签页(b) 帮助(H)
root@milozou:~/csvt.py root@milozou:~/test/pyscript/so... root@milozou:~/week/pyscripts... root@milozou:~/test/pyscript/so...
>>> cur.scroll(0,'absolute')
>>> cur.fetchmeny()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Cursor' object has no attribute 'fetchmeny'
>>> cur.fetchmany()
((1L, 'tom', 30, 'm'),)
>>> cur.fetchmany(15)
((2L, 'jack', 20, 'm'), (3L, 'alice', 23, 'f'), (4L, 'robbin', 25, 'f'),
(5L, 'luck', 132, 's'), (6L, 'lia', 35, 'm'), (9L, 'luce', None, 's'),
(10L, 'python', 20, 'm'), (11L, 'aa', 30, 'm'), (12L, 'cc', 23, 'm'),
(13L, 'milo', 25, 'm'), (14L, 'qq', 99, 'm'), (18L, 'milo', 20, 'm'),
(16L, 'boy', 2, 'm'), (17L, 'c', 30, 's'), (19L, 'csvt', 5, 's'))
>>> cur.fetchmany(15)
((21L, 'c', 1, 's'), (22L, 's', 2, 'm'))
>>>
>>> cur.fetchmany(15)
()
>>> cur.scroll(0,'absolute')
>>>
```

```
00:23:55 / 00:25:42
QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv 日 12月 9 09:53 root
快速退 00:24:24 (94%) root@milozou:~/week/pyscripts/mysqltest
文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签页(b) 帮助(H)
root@milozou:~/csvt.py root@milozou:~/test/pyscript/so... root@milozou:~/week/pyscripts... root@milozou:~/test/pyscript/so...
>>> cur.fetchmany(18)
((1L, 'tom', 30, 'm'), (2L, 'jack', 20, 'm'), (3L, 'alice', 23, 'f'), (4L, 'robbin', 25, 'f'), (5L, 'luck', 132, 's'), (6L, 'lia', 35, 'm'), (9L, 'luce', None, 's'), (10L, 'python', 20, 'm'), (11L, 'aa', 30, 'm'), (12L, 'cc', 23, 'm'), (13L, 'milo', 25, 'm'), (14L, 'qq', 99, 'm'), (18L, 'milo', 20, 'm'), (16L, 'boy', 2, 'm'), (17L, 'c', 30, 's'), (19L, 'csvt', 5, 's'), (21L, 'c', 1, 's'), (22L, 's', 2, 'm'))
>>> cur.execute("select * from userinfo")
18L
>>> cur.fetchmany(18)
((1L, 'tom', 30, 'm'), (2L, 'jack', 20, 'm'), (3L, 'alice', 23, 'f'), (4L, 'robbin', 25, 'f'), (5L, 'luck', 132, 's'), (6L, 'lia', 35, 'm'), (9L, 'luce', None, 's'), (10L, 'python', 20, 'm'), (11L, 'aa', 30, 'm'), (12L, 'cc', 23, 'm'), (13L, 'milo', 25, 'm'), (14L, 'qq', 99, 'm'), (18L, 'milo', 20, 'm'), (16L, 'boy', 2, 'm'), (17L, 'c', 30, 's'), (19L, 'csvt', 5, 's'), (21L, 'c', 1, 's'), (22L, 's', 2, 'm'))
>>> cur.execute("select * from userinfo")
```

QQ影音 中谷教育在线教学-Python-37-MySQLdb.flv 日 12月 9 09:54 root

应用程序 位置 系统

快退 00:25:16 (98%) root@milozou:~/week/pyscripts/mysqltest

文件(E) 编辑(E) 查看(V) 搜索(S) 终端(T) 标签页(b) 帮助(H)

root@milozou:~/csvtpy root@milozou:~/test/pyscript/so... root@milozou:~/week/pyscripts... root@milozou:~/test/pyscript/so...

```
>>> cur.fetchmany(cur.execute("select * from userinfo"))
((1L, 'tom', 30, 'm'), (2L, 'jack', 20, 'm'), (3L, 'alice', 23, 'f'), (
4L, 'robbin', 25, 'f'), (5L, 'luck', 132, 's'), (6L, 'lia', 35, 'm'), (
9L, 'luce', None, 's'), (10L, 'python', 20, 'm'), (11L, 'aa', 30, 'm'),
(12L, 'cc', 23, 'm'), (13L, 'milo', 25, 'm'), (14L, 'qq', 99, 'm'), (1
8L, 'milo', 20, 'm'), (16L, 'boy', 2, 'm'), (17L, 'c', 30, 's'), (19L,
'csvt', 5, 's'), (21L, 'c', 1, 's'), (22L, 's', 2, 'm'))
>>> cur.close()
>>> conn.close()
>>>
```

先关游标, 再关链接

root@milozou:~/wee...

00:25:16 / 00:18:05