**Python核心编程第二版-笔记**

# Python对象

## Python对象

Python对象拥有三个特性：

身份（id()）、类型（type()）、值

对象属性：

Python用句点（.）标记法来访问属性。最常用的属性是函数和方法。

含有数据属性的对象包括（但不限于）：类、类实例、模块、复数和文件。

## 标准类型（基本数据类型）

* 数字（分为几个子类型，其中有三个是整型）
  + Integer 整型

>>> type(123)

<type 'int'>

* + Boolean 布尔型

>>> type(True)

<type 'bool'>

* + Long Integer 长整型

>>> type(12345678901234215)

<type 'long'>

>>> type(12L)

<type 'long'>

* + Floating point real number 浮点型

>>> type(1.23)

<type 'float'>

* + Complex number 复数型

>>> type(1+2j)

<type 'complex'>

* String 字符串

>>> type('abc')

<type 'str'>

* List 列表

>>> type([1,2,3])

<type 'list'>

* Tuple 元组

>>> type((1,2,3))

<type 'tuple'>

* Dictionary 字典

>>> type({1:'a', 2:'b'})

<type 'dict'>

## 其他内建类型

* 类型
* Null对象（None）
* 文件
* 集合/固定集合
* 函数/方法
* 模块
* 类

### 类型对象和type类型对象

所有类型对象的类型都是type

类就是类型，实例是对应类型的对象

#############

>>> type(10)

<type 'int'>

>>> type(type(10))

<type 'type'>

>>>

#############

### None，Python的Null对象

Null对象或NoneType，一个特殊的类型，只有一个值：None。

它不支持任何运算，没有任何内建方法，布尔值是False

* 布尔值是False的对象：None、False（布尔类型）、所有的值为零的数

## 内部类型

代码、帧、跟踪记录、切片、省略、Xrange

### 代码对象

是编译过的Python源代码片段，是可执行对象。

通过调用内建函数compile()可以得到代码对象。

代码对象可以被exec命令或eval()内建函数来执行。

###################################

>>> code = "for i in range(3): print i"

>>> cmpcode = compile(code, '', 'exec')

>>> exec cmpcode

0

1

2

>>> cmpcode

<code object <module> at 02511380, file "", line 1>

>>> str = "3 \* 4 + 5"

>>> a = compile(str, '', 'eval')

>>> eval(a)

17

>>>

###################################

### 帧对象

表示Python的执行栈帧。

包含Python解释器在运行时所需要知道的所有信息。

它的属性包括：

1. 指向上一帧的链接
2. 正在被执行的代码对象
3. 本地及全局名字空间字典
4. 当前指令

每次函数调用产生一个新的帧，每个帧对象都会相应创建一个C栈帧。

用到帧对象的一个地方是跟踪记录对象

### 跟踪记录对象

当代码运行出错，Python就会引发一个异常，当异常发生时，一个包含针对异常的栈跟踪信息的跟踪记录对象就会被创建。

#################################

>>> b

Traceback (most recent call last):

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

NameError: name 'b' is not defined

>>>

#################################

### 切片对象

当使用切片语法时，就会创建切片对象。

切片方式有：步进切片、多维切片、省略切片。

#############步进切片##############

>>> str

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2

2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4

2, 43, 44, 45, 46, 47, 48, 49]

>>> str[1:10:2]

[1, 3, 5, 7, 9]

>>> str[10:1:-2]

[10, 8, 6, 4, 2]

>>> str[:20:4]

[0, 4, 8, 12, 16]

>>> str[:20:-4]

[49, 45, 41, 37, 33, 29, 25, 21]

>>> str[::5]

[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]

>>> str[::-5]

[49, 44, 39, 34, 29, 24, 19, 14, 9, 4]

>>> str[:5]

[0, 1, 2, 3, 4]

>>>

###################################

#############多维切片##############

>>> str

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2

2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4

2, 43, 44, 45, 46, 47, 48, 49]

>>> str2 = numpy.array(str).reshape(5,10)

>>> str2

array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],

[30, 31, 32, 33, 34, 35, 36, 37, 38, 39],

[40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])

>>> str2[0,2:4]

array([2, 3])

>>> str2[2:4,2:4]

array([[22, 23],

[32, 33]])

>>> str2[:, 3]

array([ 3, 13, 23, 33, 43])

>>> str2[::2,::2]

array([[ 0, 2, 4, 6, 8],

[20, 22, 24, 26, 28],

[40, 42, 44, 46, 48]])

>>>

##############################################

#############省略切片##############

>>> str2

array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],

[30, 31, 32, 33, 34, 35, 36, 37, 38, 39],

[40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])

>>> str2[...,2:4]

array([[ 2, 3],

[12, 13],

[22, 23],

[32, 33],

[42, 43]])

###################################

### 省略对象

用于切片语法中，起记号作用。

省略对象有一个唯一的名字Ellipsis，其布尔值始终为True

###################

>>> bool(Ellipsis)

True

>>> type(Ellipsis)

<type 'ellipsis'>

>>>

###################

#####################

>>> a=[1,2]

>>> a.append(a)

>>> print a

[1, 2, [...]]

#####################

>>> str2

array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

[20, 21, 22, 23, 24, 25, 26, 27, 28, 29],

[30, 31, 32, 33, 34, 35, 36, 37, 38, 39],

[40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])

>>> print str2[...,0]

[ 0 10 20 30 40]

>>> print str2[0,...,0]

0

>>> #str2[0,...,0] = str2[0][0]

...

>>>

##################################

### Xrange对象

调用内建函数xrange()会生成一个Xrange对象。

xrange()用于需要节省内存使用或range()无法完成的超大数据集场合。

###########################

>>> xrange(10)

xrange(10)

>>> type(xrange(10))

<type 'xrange'>

>>>

###########################

## 标准类型运算符

### 对象值的比较

比较运算符

==、<=、>=、<、>、!=

#########################

>>> 'abv' == 'xef'

False

>>> 'abv' >= 'xef'

False

>>> 'abv' <= 'xef'

True

>>> [2,4] == [4,2]

False

>>> [2,4] == [2,4]

True

>>> 3<4<6

True

>>> 3<4==4

True

>>> 2!=4

True

>>>

#################################

### 对象身份比较

is、is not

######################

>>> a=4.3

>>> b=1.3+3.0

>>> a is b

False

>>> id(a)

43448120

>>> id(b)

43448072

>>> a == b

True

>>> a is not b

True

>>>

##########################

* 如果创建对象为整数或字符串对象时，会指向相同的对象，因为Python对简单整数已经做了缓存。（不推荐使用）

######################

>>> a=2

>>> b=1+1

>>> a is b

True

>>> c=4

>>> d=1+3

>>> c is d

True

>>> id(c) == id(d)

True

>>> a ='a'

>>> b = 'a'

>>> a is b

True

>>> id(a)

30331752

>>> id(b)

30331752

>>>

#######################

### 布尔类型

布尔逻辑运算符：

and、or、not

##########################

>>> 1<3 and 3<5

True

>>> 1>3 or 3<5

True

>>> x,y =23,45

>>> x is y

False

>>> not(x is y)

True

>>> not x>y

True

>>>

############################

## 标准类型内建函数

cmp()、repr()、str()、type()、等同于repr()函数的单反引号（``）运算符

### type()

用法：type(object)

接受一个对象作为参数，并返回它的类型。它的返回值是一个类型对象。

####################################

>>> type(123)

<type 'int'>

>>> type('python')

<type 'str'>

>>> type(type(123))

<type 'type'>

>>>

####################################

对于不容易显示的对象，Python会以一个相对标准的格式表示：<object\_something\_or\_another>。通常会提供对象类别、ID或位置，或其它合适的信息。

### cmp()

用于比较两个对象的大小。

cmp(obj1, obj2)

obj1<obj2, return -1

obj1>obj2, return 1

obj1==obj2, return 0

########################

>>> a=1

>>> b=2

>>> cmp(a,b)

-1

>>> cmp(b,a)

1

>>> b=1

>>> cmp(a,b)

0

>>> a='a'

>>> b='b'

>>> cmp(a,b)

-1

>>> cmp(b,a)

1

>>> b='a'

>>> cmp(a,b)

0

>>>

#########################

### str()和repr() (及``运算符)

str()函数得到的字符串可读性好，一般用于输出

repr()函数得到的字符串通常可以用来重新获得该对象。obj == eval(repr(obj))

####################

>>> str(123)

'123'

>>> str(2e5)

'200000.0'

>>> str([1,2,3])

'[1, 2, 3]'

>>> repr([1,2,3])

'[1, 2, 3]'

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

'[1, 2, 3]'

>>>

####################

str()输出对人比较友好，repr()输出对Python比较友好。

* 为什么有了repe()还需要``？

因为某些场合函数会比运算符更适合使用。

例：当处理类似函数这样的可执行对象或根据不同的数据项调用不同的函数处理时，函数比运算符用起来方便。

如：x \*\* y 和 pow(x, y) 执行的都是x的y次方。

### type() 和 isinstance()

检查类型：

##############################################

#encoding=utf-8

def displayNumType(num):

if isinstance(num, (int, long, float, complex)):

print num, 'is a number of type: ', type(num).\_\_name\_\_

else:

print num, 'is not a number at all!'

displayNumType(123)

displayNumType(-123)

displayNumType(123.123)

displayNumType(123L)

displayNumType(1+2j)

displayNumType('abc')

############################################

输出：

E:\个人\Python\test>python typechk.py

123 is a number of type: int

-123 is a number of type: int

123.123 is a number of type: float

123 is a number of type: long

(1+2j) is a number of type: complex

abc is not a number at all!

##############################################

### Python类型运算符和内建函数总结

反单引号：``

内建函数：cmp(obj1, obj2)、repr(obj)、str(obj)、type(obj)

值比较：>、<、>=、<=、==、!=、<>

对象比较：is、is not

布尔比较：not、and、or

## 类型工厂函数

* int()、long()、float()、complex()
* str()、unicode()、basestring()
* list()、tuple()
* type()
* dict()
* bool()
* set()、frozenset()
* object()
* classmethod()
* staticmethod()
* super()
* property()
* file()

## 标准类型的分类

### 存储模型

按存储模型分类：

* 原子或标量存储：能保存单个字面对象的类型
* 容器存储：可容纳多个对象的类型

|  |  |
| --- | --- |
| 分类 | python 类型 |
| 标量/原子类型 | 数值（所有的数值类型）、字符串（全部是文字） |
| 容器类型 | 列表、元组、字典 |

### 更新模型

对象创建成功之后，它的值是否可以进行更新？

按更新模型分类：

* 可变对象：允许值被更新
* 不可变对象：不允许值被更改

|  |  |
| --- | --- |
| 分类 | python 类型 |
| 可变类型 | 列表、字典 |
| 不可变类型 | 数字、字符串、元组 |

### 访问模型

按访问模型分类：

* 直接存取
* 顺序
* 映射

|  |  |
| --- | --- |
| 分类 | python 类型 |
| 直接访问 | 数字 |
| 顺序访问 | 字符串、列表、元组 |
| 映射访问 | 字典 |

### 标准类型分类：

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | 存储模型 | 更新模型 | 访问模型 |
| 数字 | 标量/原子 | 不可更改 | 直接访问 |
| 字符串 | 标量/原子 | 不可更改 | 顺序访问 |
| 列表 | 容器 | 可更改 | 顺序访问 |
| 元组 | 容器 | 不可更改 | 顺序访问 |
| 字典 | 容器 | 可更改 | 映射访问 |

## 不支持的类型

* char、byte
* 指针
* short
* double

## 练习

1. Python对象。与所有Python对象有关的三个属性是什么？请简单的描述一下。

身份、值、类型

1. 类型。不可更改（immutable）指的是什么？Python的哪些类型是可更改的（mutable），哪些不是？

不可更改指一个对象创建成功后，对象的身份不可更改。

可更改类型：列表、字典

1. 类型。哪些Python类型是按照顺序访问的，它们和映射类型的不同是什么？

按照顺序访问的类型：字符串、列表、元组

序列类型是元素按从0开始的索引顺序存放，使用顺序的数字偏移量取值；

映射类型是键值对的集合，元素无序存放，通过唯一的key来访问取值。

1. type()。内建函数type()做什么？type()返回的对象是什么？

type()用来检查对象的类型。

type()返回的对象是类型对象。

1. str()和repr()。内建函数str()与repr()之间的不同是什么？哪一个等价于反引号（``）运算符？

str()：可读性好，一般用于直接输出

repr()：可以重新获得该对象，一般用eval()函数来得到原来的对象。obj = eval(repr(obj))

repr()等价于反引号（``）

1. 对象相等。您认为type(a) == type(b)和type(a) is type(b)之间的不同是什么？

type(a) == type(b)：对象值的比较

type(a) is type(b)：对象类型的比较

1. 内建函数dir()。在第二章的几个练习中，我们用内建函数dir()做了几个实验，它接受一个对象，然后给出相应的属性。请对types模块做相同的实验。记下您熟悉的类型，包括您对这些类型的认识，然后记下您还不熟悉的类型。在学习Python的过程中，您要逐步将“不熟悉”的类型变得“熟悉”起来。

#####################################

>>> import types

>>> dir(types)

['BooleanType', 'BufferType', 'BuiltinFunctionType', 'BuiltinMethodType', 'Class

Type', 'CodeType', 'ComplexType', 'DictProxyType', 'DictType', 'DictionaryType',

'EllipsisType', 'FileType', 'FloatType', 'FrameType', 'FunctionType', 'Generato

rType', 'GetSetDescriptorType', 'InstanceType', 'IntType', 'LambdaType', 'ListTy

pe', 'LongType', 'MemberDescriptorType', 'MethodType', 'ModuleType', 'NoneType',

'NotImplementedType', 'ObjectType', 'SliceType', 'StringType', 'StringTypes', '

TracebackType', 'TupleType', 'TypeType', 'UnboundMethodType', 'UnicodeType', 'XR

angeType', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', '\_\_pack

age\_\_']

#####################################

1. 列表和元组。列表和元组的相同点是什么？不同点是什么？

相同点：都是容器类型，都是顺序访问

不同点：列表可更改，元组不可更改

1. 练习，给定以下赋值：

a = 10

b = 10

c = 100

d = 100

e = 10.0

f = 10.0

请问下面各表达式的输出是什么？为什么？

1. a is b

输出：True

原因：Python对简单的数字有缓存。指向的是相同的对象。id(a) == id(b)

1. c is d

输出：True

原因：Python对简单的数字有缓存。指向的是相同的对象。id(c) == id(d)

1. e is f

输出：False

原因：e和f指向的是不同的对象。id(e) != id(f)

# 数字

## 数字类型

### 如何创建数值对象并用其赋值（数字对象）

anInt = 123

aLong = 9999999999999999999999999L

aFloat = 1.23

aComplex = 1+2j

### 如何更新数字对象

对于不可变类型来说，更新数字对象是，创建一个新的对象并把它赋给变量。

#######################

>>> anInt = 1

>>> id(anInt)

32177896

>>> anInt = 2

>>> id(anInt)

32177884

#######################

### 如何删除数字对象

删除对象的引用 del

########################

>>> anInt = 2

>>> id(anInt)

32177884

>>> del anInt

>>> id(anInt)

Traceback (most recent call last):

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

NameError: name 'anInt' is not defined

#########################

## 整型

### 用于数字运算的内建函数：

abs(num)：返回num的绝对值

####################

>>> abs(-13)

13

>>> abs(10.)

10.0

>>> abs(-34812.34534)

34812.34534

>>> abs(0.23-0.78)

0.55

>>> abs(1+2j)

2.23606797749979

####################

coerce(num1,num2)：将num1和num2转换为同一类型，然后以一个元组的形式返回

###################

>>> coerce(1,2.0)

(1.0, 2.0)

>>> coerce(1.2,12L)

(1.2, 12.0)

>>> coerce(1+2j,12l)

((1+2j), (12+0j))

>>> coerce(1+2j,12L)

((1+2j), (12+0j))

>>>

###################

divmod(num1,num2)：返回一个元组（num1/num2, num1%num2）,对浮点型和复数的商进行下舍入（复数仅取实数部分的商）

####################

>>> divmod(10,3)

(3, 1)

>>> divmod(3,10)

(0, 3)

>>> divmod(10,2.5)

(4.0, 0.0)

>>> divmod(2.5,10)

(0.0, 2.5)

>>> divmod(2+1j,0.5-1j)

((-0+0j), (2+1j))

>>>

####################

pow(num1,num2,mod=1)：取num1得num2次方，如果提供mod参数，则计算结果再对mod进行取余运算

####################

>>> pow(2,3)

8

>>> pow(5,3)

125

>>> pow(-5,2)

25

>>> pow(-5,3)

-125

>>> pow(-5.235234,3)

-143.48559220290582

>>> pow(1+2j,2)

(-3+4j)

>>> pow(3,2,2)

1

####################

round(flt,ndig=1)：接受一个浮点型flt并对其四舍五入，保存ndig位小数。若不提供ndig参数，则默认小数点后0位

####################

>>> round(3)

3.0

>>> round(3.45)

3.0

>>> round(3.4999,1)

3.5

>>> round(3.4999,2)

3.5

>>> round(321.4999,-2)

300.0

>>> round(321.4999,-1)

320.0

>>> round(326.4999,-1)

330.0

>>> round(-326.4999,3)

-326.5

>>> round(-326.4999,1)

-326.5

>>> round(-326.4678,3)

-326.468

>>>

####################

#### int()、round()、math.floor()区别

int()：直接截去小数部分（返回值为整数）

round()：得到最接近原数的整型（返回值为浮点型）

math.floor()：得到最接近原数但小于原数的整型（返回值为浮点型）

###########################

>>> int(0.2)

0

>>> round(0.2)

0.0

>>> import math

>>> math.floor(0.2)

0.0

>>> int(.7)

0

>>> round(.7)

1.0

>>> math.floor(.7)

0.0

>>> int(1.2)

1

>>> round(1.2)

1.0

>>> math.floor(1.2)

1.0

>>> int(1.7)

1

>>> round(1.7)

2.0

>>> math.floor(1.7)

1.0

>>> int(-1.2)

-1

>>> round(-1.2)

-1.0

>>> math.floor(-1.2)

-2.0

>>> int(-1.7)

-1

>>> round(-1.7)

-2.0

>>> math.floor(-1.7)

-2.0

>>>

###########################

#### 仅用于整型的内建函数

#### 进制转换函数

hex(num)：将数字转换成十六进制数并以字符串形式返回

oct(num)：将数字转换成八进制数并以字符串形式返回

##########################

>>> hex(123)

'0x7b'

>>> hex(12)

'0xc'

>>> type(hex(12))

<type 'str'>

>>> oct(123)

'0173'

>>> oct(12)

'014'

>>> type(oct(12))

<type 'str'>

>>>

##########################

#### ASCLL转换函数

chr(num)：将ASCLL值的数字转换成ASCLL字符，范围：1 <= num <=255

ord(chr)：接受一个ASCLL或Unicode字符（长度为1的字符串），返回相应的ASCLL值或Unicode值

unichr(num)：接受Unicode码值，返回其对应的Unicode字符

#############################

>>> chr(97)

'a'

>>> chr(65)

'A'

>>> ord('a')

97

>>> ord('A')

65

#############################

### random模块：

randint()：来年改革整型参数，返回二者之间的随机整型

############################

>>> import random

>>> random.randint(1,5)

2

>>> random.randint(1,5)

2

>>> random.randint(1,5)

3

>>> random.randint(1,5)

5

>>> random.randint(1,5)

4

>>> random.randint(1,5)

1

>>> random.randint(1,5)

5

############################

randrange()：接受和range()一样的参数，随机返回range([start,]stop[,step])结果的一项

###########################

>>> random.randrange(1,10,2)

3

>>> random.randrange(1,10,2)

5

>>> random.randrange(1,10,2)

7

>>> random.randrange(1,10,2)

7

>>> random.randrange(1,10,2)

9

>>> random.randrange(1,10,2)

7

>>> random.randrange(1,10,2)

3

>>> random.randrange(1,10,2)

1

###########################

uniform()：几乎和randint()一样，不过它返回的是二者之间的一个浮点型（不包括范围上限）

##########################

>>> random.uniform(1,4)

1.9253509535844897

>>> random.uniform(1,4)

3.0647894353949034

##########################

random()：类似于uniform()，只不过下限恒等于0.0，上限恒等于1.0

##########################

>>> random.random()

0.18326735664530858

>>> random.random()

0.49401230258565443

>>> random.random()

0.3537790093765273

##########################

choice()：随机返回给定序列的一个元素

##########################

>>> random.choice('abc')

'a'

>>> random.choice('abc')

'b'

>>> random.choice('abc')

'a'

>>> random.choice([1,2,3,4])

1

>>> random.choice([1,2,3,4])

3

>>> random.choice((1,2,3,4))

2

>>> random.choice((1,2,3,4))

3

##########################

## 练习

1.写一个函数，计算并返回两个数的乘积

调用这个函数，并显示它的结果

#########################

#encoding=utf-8

def product(x,y):

return x\*y

print product(2,3)

#########################

2.标准类型操作符。写一段脚本，输入一个检验成绩，根据下面的标准，输出他的评分成绩（A-F）。

A：90-100

B：80-89

C：70-79

D：60-69

F：<60

#########################

#encoding=utf-8

while True:

try:

score = raw\_input("Please enter a score(q:quit): ")

if score == 'q':

print 'Exit!'

break

s = int(score)

print 'Score:',

if s >= 90 and s <= 100:

print 'A'

elif s >= 80 and s < 90:

print 'B'

elif s >= 70 and s < 80:

print 'C'

elif s >= 60 and s < 70:

print 'D'

elif s >= 0 and s< 60:

print 'F'

else:

print 'input error!'

except Exception,e:

print 'input error!'

#########################

3.取余。判断给定年份是否是闰年。使用下面的公式。

一个闰年就是值它可以被4整除，但不能被100整除，或者它既可以被4整除又可以被100整除。比如1992年、1996年和2000年是闰年，但1967年和1900年则不是闰年。下一个是闰年的整世纪是2400年。

#########################

#encoding=utf-8

while True:

try:

year = raw\_input("请出入一个年份(输入“退出”则退出程序)：".decode("utf-8").encode("gbk"))

if year == "退出".decode("utf-8").encode("gbk"):

print u"退出程序！"

break

y = int(year)

if (((y % 4 == 0) and (y % 100 != 0)) or (y % 400 == 0)) and (y > 0):

print year + u"年是闰年。"

elif y > 0:

print year + u"年不是闰年。"

else:

print u"输入有误。"

except Exception,e:

print u"输入有误。"

#########################

1. 取一个任意小于1美元的金额，然后计算可以换成最少多少枚硬币。硬币有1美分、5美分、10美分、25美分4种。1美元等于100美分。举例来说，0.76美元换算结果应该是3枚25美分，1枚1美分。类似76枚1美分，2枚25美分+2枚10美分+1枚5美分+1枚1美分这样的结果都是不符合要求的。

########################

#encoding=utf-8

import random

n = round(random.random(),2)

print n,u"美元",

num=int(n\*100)

num25=num/25

num-=num25\*25

num10=num/10

num-=num10\*10

num5=num/5

num-=num5\*5

num1=num

print u"= %d枚25美分 + %d枚10美分 + %d枚5美分 + %d枚1美分" % (num25,num10,num5,num1)

########################

1. 几何。计算面积和体积。

a 正方形和立方体

b 园和球

#######################

#encoding=utf-8

try:

a=raw\_input('请输入边长/半径：'.decode('utf-8').encode('gbk'))

print u'此正方形面积为：%.2f' % float(a)\*\*2

print u'此立方体体积为：%.2f' % float(a)\*\*3

print u'此圆的面积为：%.2f' % ((float(a)\*\*2)\*3.14)

print u'此球的体积为：%.2f' % (4.0/3\*3.14\*(float(a)\*\*3))

except Exception, e:

print u'输入有误！'

#######################

1. 取余。

a 使用循环和算术运算，求出0-20之间的所有偶数。

#######################

for i in range(21):

if i % 2 ==0:

print i,

#######################

b 同上，不过这次输出所有的奇数。

#######################

for i in range(21):

if i % 2 !=0:

print i,

#######################

c 输入两个整数，判断两者是否有整除关系，根据判断结果分别返回True和False。

######################

try:

num1 = int(raw\_input('请输入一个数：'.decode('utf-8').encode('gbk')))

num2 = int(raw\_input('请输入另一个数：'.decode('utf-8').encode('gbk')))

if num1 == 0 and num2 == 0:

print 'false'

elif num2 != 0 and num1 % num2 == 0:

print 'True'

elif num1 != 0 and num2 % num1 == 0:

print 'True'

else:

print 'False'

except Exception, e:

print u'输入有误！'

######################

# 序列

## 字符串

### 字符串的创建和赋值

----------------------------------------

>>> s = 'hello world'

>>> s1 = "hello python"

>>> print s1

hello python

>>> s

'hello world'

>>> s2 = str(range(4))

>>> s2

'[0, 1, 2, 3]'

>>>

-----------------------------------------

### 遍历字符串

--------------------------------------------

>>> s='hello world'

>>> s[1]

'e'

>>> s[1:4]

'ell'

>>> s[:3]

'hel'

>>> s[::-1]

'dlrow olleh'

>>> s[:6:-1]

'dlro'

>>> s[-1:-5:-1]

'dlro'

>>> s[4:]

'o world'

>>>

-------------------------------------------------

### 修改字符串

---------------------------------------------------

>>> s='hello world'

>>> s+s

'hello worldhello world'

>>> s=s +' hello python'

>>> s

'hello world hello python'

>>> s='python'

>>> s

'python'

>>>

-----------------------------------------------------

### 删除字符和字符串

-----------------------------------------------------

>>> s='hello world'

>>> s=s[:2]+s[5:7]

>>> s

'he w'

>>> del s

>>> s

Traceback (most recent call last):

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

NameError: name 's' is not defined

>>>

------------------------------------------------

### 字符串和操作符

#### string模块：

# ascii\_letters：大写字母+小写字母

--------------------------

string.ascii\_letters

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

------------------------------

# ascii\_lowercase：小写字母

--------------------------------

string.ascii\_lowercase

'abcdefghijklmnopqrstuvwxyz'

----------------------------------

# ascii\_uppercase：大写字母

--------------------------------

string.ascii\_uppercase

'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

--------------------------------

# atof：全是数字的字符串，返回其float型数字

--------------------------------

string.atof('123')

123.0

string.atof('123.123')

123.123

----------------------------------

# capitalize：首字母大写

-----------------------------------

>>> string.capitalize('fafdf')

'Fafdf'

------------------------------------

# capwords：单词首字母大写

----------------------------------

>>> string.capwords('hello world hello python')

'Hello World Hello Python'

>>> string.capwords('hello world,hello python!hi,nihao!')

'Hello World,hello Python!hi,nihao!'

>>> string.capwords('hello world,hello python!hi,nihao!',',')

'Hello world,Hello python!hi,Nihao!'

--------------------------------------

# center：以字符为中心，给定字符串长度，不足在两边补给的字符

--------------------------------------

>>> string.center('a',9,'\*')

'\*\*\*\*a\*\*\*\*'

--------------------------------------

# count查找字符/字符串在字符串中出现次数，可在限定的字符串中查找

---------------------------------------

>>> 'aaaabbbcccdd'.count('a')

4

>>> 'aaaabbbcccdd'.count('a',2,5)

2

>>> 'aaaabbbcccdd'.count('e',2,5)

0

>>> 'aaaabbbcccdd'.count('aa',0,5)

2

-----------------------------------------

# digits：0-9的数字

----------------------------------------

>>> string.digits

'0123456789'

--------------------------------------

# find：查找字符/字符串是否在字符串中，如果在返回索引，否则返回-1

--------------------------------------

>>> 'abc'.find('a')

0

>>> 'abc'.find('d')

-1

>>> 'abc'.find('c')

2

>>> 'abcdefg'.find('f',3)

5

>>> 'abcdefg'.find('f',3,5)

-1

----------------------------------

# hexdigits：八进制字符0-9a-fA-F

------------------------------

>>> string.hexdigits

'0123456789abcdefABCDEF'

-------------------------------

# index：返回字符/字符串是否在字符串中的索引，若不存在，则报错

---------------------------------

>>> 'abcdefg'.index('cd')

2

>>> 'abcdefg'.index('cdf')

Traceback (most recent call last):

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

ValueError: substring not found

>>> 'abcdefg'.index('cd',1,6)

2

>>> 'abcdefg'.index('cd',1,9)

2

------------------------------------------

# join：连接元组/列表中的字符串，合并为一个新字符串

--------------------------------------------

>>> ''.join(('a','b','c'))

'abc'

>>> ''.join(['a','b','c'])

'abc'

>>> '\*'.join(['a','b','c'])

'a\*b\*c'

>>> '#'.join(('a','b','c'))

'a#b#c'

-------------------------------------------------

# letters：小写字母+大写字母

-------------------------------------------------

>>> string.letters

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

-----------------------------------------------------

# ljust：字符串左对齐，使用给定字符填充至指定长度

--------------------------------------------------

>>> string.ljust('abc',10,'\*')

'abc\*\*\*\*\*\*\*'

-----------------------------------------------------

# lower：将字符串中所有大写字母转为小写

--------------------------------------------------

>>> string.lower('AFJASEFjroesirs3452AESFF')

'afjasefjroesirs3452aesff'

-------------------------------------------------

# lowercase：所有小写字母

-------------------------------------------

>>> string.lowercase

'abcdefghijklmnopqrstuvwxyz'

-----------------------------------------------

# lstrip：去除字符串左侧空格或指定字符

--------------------------------------------

>>> ' a b c '.lstrip()

'a b c '

>>> 'aaavs a b c aaaa'.lstrip('a')

'vs a b c aaaa'

------------------------------------------

# octdigits：所有八进制字符

----------------------------------------

>>> string.octdigits

'01234567'

--------------------------------------

# replace：替换字符串中的指定字符/字符串为指定的字符/字符串，若次数指定，则替换不超过指定次数

---------------------------------------

>>> 'ahuahuahjihjahuabkaaagr'.replace('a','\*\*\*')

'\*\*\*hu\*\*\*hu\*\*\*hjihj\*\*\*hu\*\*\*bk\*\*\*\*\*\*\*\*\*gr'

>>> 'ahuahuahjihjahuabkaaagr'.replace('a','\*\*\*',2)

'\*\*\*hu\*\*\*huahjihjahuabkaaagr'

------------------------------------------

# rfind：从字符串右边开始查找

---------------------------------

>>> 'abcdefd'.rfind('d')

6

>>> 'abcdefd'.rfind('g')

-1

-------------------------------

# rindex：从右边开始返回字符/字符串中的索引，不存在则报错

---------------------------------

>>> 'abcdefd'.rindex('g')

Traceback (most recent call last):

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

ValueError: substring not found

>>> 'abcdefd'.rindex('d')

6

----------------------------------

# rjust：字符串右对齐，使用给定字符填充至指定长度

-------------------------------------

>>> 'abc'.rjust(10)

' abc'

>>> 'abc'.rjust(10,'+')

'+++++++abc'

-------------------------------------

# rsplit：从右侧开始，以给定字符切片指定次数

--------------------------------------

>>> 'uhaujajhioahuaoijaoahoiaj'.rsplit('a',3)

['uhaujajhioahuaoij', 'o', 'hoi', 'j']

>>> 'uhaujajhioahuaoijaoahoiaj'.rsplit('a')

['uh', 'uj', 'jhio', 'hu', 'oij', 'o', 'hoi', 'j']

------------------------------------------

# rstrip：去除字符串右侧空格，或指定字符

-------------------------------------------

>>> ' uhaujaj '.rstrip()

' uhaujaj'

>>> '++++uhaujaj+++++'.rstrip('+')

'++++uhaujaj'

--------------------------------------------

# split：从左侧开始，以给定字符切片指定次数

-------------------------------------------

>>> 'uhaujajhioahuaoijaoahoiaj'.split('a')

['uh', 'uj', 'jhio', 'hu', 'oij', 'o', 'hoi', 'j']

>>> 'uhaujajhioahuaoijaoahoiaj'.split('a',3)

['uh', 'uj', 'jhio', 'huaoijaoahoiaj']

-------------------------------------------

# strip：去除首尾空格或指定字符

---------------------------------------------

>>> '++++uhaujaj+++++'.strip('+')

'uhaujaj'

>>> '++++uhaujaj+++++'.strip('+,')

'uhaujaj'

>>> '+,+++uhaujaj+++,,++'.strip('+,')

'uhaujaj'

--------------------------------------------------

# swapcase：翻转字符串中的大小写

--------------------------------------------

>>> string.swapcase('fasdfJOIFJEfje')

'FASDFjoifjeFJE'

>>> string.swapcase('fasdfJOI234FJEfje')

'FASDFjoi234fjeFJE'

----------------------------------------------

# upper：将所有字母大写

------------------------------------------

>>> string.upper('fdsfEFWS324fdsf')

'FDSFEFWS324FDSF'

--------------------------------------------

# uppercase：所有大写字母

--------------------------------------------

>>> string.uppercase

'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

--------------------------------------------

# whitespace：所有空白符

-------------------------------------------

>>> string.whitespace

'\t\n\x0b\x0c\r '

-------------------------------

# zfill：左侧补零至指定长度

-------------------------------

>>> string.zfill('agbc',10)

'000000agbc'

>>> 'ab'.zfill(10)

'00000000ab'

---------------------------------

转移字符：

\t TAB

\0 空字符

\b 退格

\a 响铃

\n 换行

\v 纵向制表符

\f 换页

\r 回车

\\ 反斜扛

\’ 单引号

\” 双引号

-----------------------------------------

>>> print 'aaa\taaa'

aaa aaa

>>> print 'aaa\0aaa'

aaa aaa

>>> print 'aaa\baaa'

aaaaa

>>> print 'aaa\naaa'

aaa

aaa

>>> print 'aaa\aaaa'

aaaaaa

>>> print 'aaa\vaaa'

aaa  
aaa

>>> print 'aaa\faaa'

aaaaaa

>>> print 'aaa\raaa'

aaa

----------------------------------

三引号：所见即所得

UTF-8编码：可以用1-4个字节来表示其他语言的字符。

UTF-16编码：用2个字节来存储。

unicode()函数：接受一个string字符串做参数，返回一个Unicode字符串

decode()：解码

encode()：编码

Unicode字符串是basestring的子类

混合类型字符串操作需要把普通字符串转换成Unicode对象。

%s：（字符串格式化操作符），是把Python字符串中的Unicode对象执行了str(u)操作,输出是u.encode(默认编码)

## 列表

不仅可以包含python的标准类型，而且可以用用户定义的对象作为自己的元素。列表可以包含不同类型的对象。

### 新建列表:

------------------------------------------------

>>> lista=[]

>>> lista

[]

>>> listb=[12,'34',5.6,['abc',]]

>>> listb=[12,'34',5.6,['abc','helll'],(1,2,3),7+9j,{1:2,3:4}]

>>> listab

Traceback (most recent call last):

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

NameError: name 'listab' is not defined

>>> listb

[12, '34', 5.6, ['abc', 'helll'], (1, 2, 3), (7+9j), {1: 2, 3: 4}]

>>> range(10)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list('123abc')

['1', '2', '3', 'a', 'b', 'c']

>>>

-----------------------------------------------------

### 访问列表中的值：

切片：

-----------------------------------------------------

>>> lista

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> lista[:]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> lista[:5]

[0, 1, 2, 3, 4]

>>> lista[3:5]

[3, 4]

>>> lista[::-1]

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

>>> lista[:7:2]

[0, 2, 4, 6]

>>> lista[2:7:2]

[2, 4, 6]

>>> lista[2::2]

[2, 4, 6, 8]

>>>

--------------------------------------------------

### 更新列表：

append，insert，extend

----------------------------------------------------

>>> lista

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> lista.append(10)

>>> lista

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> lista.insert(0,'a')

>>> lista

['a', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> lista.extend(['a',])

>>> lista

['a', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a']

>>> lista.extend(['a','b','c'])

>>> lista

['a', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'a', 'b', 'c']

>>> lista[0]=-1

>>> lista

[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'a', 'b', 'c']

>>>

-------------------------------------------------------

### 删除列表元素或本身

del，remove，pop

------------------------------------------------------

>>> lista

[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'a', 'b', 'c']

>>> del lista[:5]

>>> lista

[4, 5, 6, 7, 8, 9, 10, 'a', 'a', 'b', 'c']

>>> lista.remove(10)

>>> lista

[4, 5, 6, 7, 8, 9, 'a', 'a', 'b', 'c']

>>> del lista

>>> lista

Traceback (most recent call last):

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

NameError: name 'lista' is not defined

>>> listb

[12, '34', 5.6, ['abc', 'helll'], (1, 2, 3), (7+9j), {1: 2, 3: 4}]

>>> listb.pop(5)

(7+9j)

>>> listb

[12, '34', 5.6, ['abc', 'helll'], (1, 2, 3), {1: 2, 3: 4}]

>>> listb.pop()

{1: 2, 3: 4}

>>> listb

[12, '34', 5.6, ['abc', 'helll'], (1, 2, 3)]

----------------------------------------------------------------

### 列表解析：

----------------------------------------------------------

>>> [x\*\*2 for x in range(5)]

[0, 1, 4, 9, 16]

>>> [x\*\*2 for x in range(10) if x%2==0]

[0, 4, 16, 36, 64]

-------------------------------------------------------------

### 标准类型函数：

cmp():

比较规则：

1. 对两个列表的元素进行比较.

2. 如果比较的元素是同类型的,则比较其值,返回结果.

3. 如果两个元素不是同一种类型,则检查它们是否是数字.

a. 如果是数字,执行必要的数字强制类型转换,然后比较.

b. 如果有一方的元素是数字,则另一方的元素"大"(数字是"最小的")

c. 否则,通过类型名字的字母顺序进行比较.

4. 如果有一个列表首先到达末尾,则另一个长一点的列表"大".

5. 如果我们用尽了两个列表的元素而且所有元素都是相等的,那么结果就是个平局,就是说返回一个0.

### 序列类型函数：

len():

返回列表的元素个数

--------------------------

>>> lista=[1,2,3]

>>> len(lista)

3

>>> len(lista\*2)

6

--------------------------

max()、min()

sort()、reverse()

sorted()、reversed()

enumerate()、zip()

------------------------

>>> lista

[1, 2, 3]

>>> listb

[12, '34', 5.6, ['abc', 'helll'], (1, 2, 3), {1: 2, 3: 4}]

>>> zip(lista,listb)

[(1, 12), (2, '34'), (3, 5.6)]

>>> zip(lista)

[(1,), (2,), (3,)]

----------------------------

sum()

list()、tuple()

## 元组

### 新建元组：

只有一个元素的元组需要在元素后面加一个逗号，用以防止跟普通的分组操作符混淆

-------------------------------------------------

>>> a=(12,1.2,'1233ve',[1,2,3],1+3j,{1:2,3:4})

>>> a

(12, 1.2, '1233ve', [1, 2, 3], (1+3j), {1: 2, 3: 4})

>>> b=(1,)

>>> b

(1,)

>>> c=(None,)

>>> c

(None,)

>>> tuple('abc')

('a', 'b', 'c')

---------------------------------------

### 访问元组中的值：

切片

-----------------------------------------

>>> a=(12,1.2,'1233ve',[1,2,3],1+3j,{1:2,3:4})

>>> a[1:4]

(1.2, '1233ve', [1, 2, 3])

>>> a[3][1]

2

>>> a[:]

(12, 1.2, '1233ve', [1, 2, 3], (1+3j), {1: 2, 3: 4})

>>> a[::-1]

({1: 2, 3: 4}, (1+3j), [1, 2, 3], '1233ve', 1.2, 12)

>>> a[::3]

(12, [1, 2, 3])

>>> a[2]

'1233ve'

-----------------------------------------

### 更新元组：

元组是不可变类型，只能通过现有的片段构造一个新的元组。

-----------------------------------------------

>>> aa=(12,1.2,'1233ve',[1,2,3],1+3j,{1:2,3:4})

>>> aa=aa[0],aa[1],aa[-1]

>>> aa

(12, 1.2, {1: 2, 3: 4})

>>> b=(1,2)

>>> c=(3,4)

>>> d=b+c

>>> d

(1, 2, 3, 4)

------------------------------------------------------

### 删除元组：

只能删除一整个元组

-------------------------------------------------

>>> del a

----------------------------------------------------

### 元组的操作符和内建函数

#### 标准类型操作符，序列类型操作符和内建函数

跟列表的完全一样

##### 创建，重复，连接操作

----------------------------------------------

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

>>> t

(1, 2, 3)

>>> t\*2

(1, 2, 3, 1, 2, 3)

>>> t+(4,5,6)

(1, 2, 3, 4, 5, 6)

------------------------------------------

##### 成员关系操作，切片操作

--------------------------------------------

>>> t

(1, 2, 3)

>>> 1 in t

True

>>> 4 in t

False

>>> t[2]

3

>>> t[1:]

(2, 3)

>>> t[:]

(1, 2, 3)

>>> t[::-1]

(3, 2, 1)

---------------------------------------------------

##### 内建函数

-------------------------------------------------

>>> t

(1, 2, 3)

>>> str(t)

'(1, 2, 3)'

>>> list(t)

[1, 2, 3]

>>> len(t)

3

>>> max(t)

3

>>> min(t)

1

>>> cmp(t,(3,4,5))

-1

-----------------------------------------

##### 操作符

------------------------------------------

>>> (1,2)<(2,3)

True

>>> (1,2)>(2,3)

False

>>> (1,2)==(2,3)

False

---------------------------------------------

#### 元组类型操作符和内建函数，内建方法

跟列表一样，元组也没有自己的运算符和内建函数，因为元组是不可变的

### 元组的特殊性

#### 默认集合类型

所有的多个对象的，逗号分隔的，没有明确用符号定义的，默认的类型都是元组。

-------------------------------------

>>> 1,2,3,'agc'

(1, 2, 3, 'agc')

>>> x,y,z=1,2,3

>>> x,y,x

(1, 2, 1)

>>> x,y,z

(1, 2, 3)

-------------------------------------

所有函数返回的多对象（不包括有符号封装的）都是元组类型。

------------------------------

>>> def fun():

... return 1,2,3

...

>>> fun()

(1, 2, 3)

>>> 1,2>3,4

(1, False, 4)

-------------------------------

## 列表VS元组

使用不可变类型变量的一个情况是：若你在维护一些敏感的数据，并且需要把这些数据传递给一个并不了解的函数（调用该函数可能会修改你的数据），这时用不可变类型变量就会安全许多。

需要可变类型参数的例子：如果你在管理动态数据集合时，需要先把它们创建出来，逐渐地或者不定期的添加它们，或者有时还要移除一些单个元素，这时必须使用可变类型参数。

通过内建的list()和tuple()转换函数，可以轻松的在两者之间进行转换。

# 映射和集合类型

## 映射类型：字典

字典是Python语言中唯一的映射类型。

### 创建字典和给字典赋值

----------------------------------------------

>>> dict1={}

>>> dict1

{}

>>> dict2={'name':'python','age':18}

>>> dict2

{'age': 18, 'name': 'python'}

>>> dict3=dict((['a',1],['b',2]))

>>> dict3

{'a': 1, 'b': 2}

>>> dict4={}.fromkeys(('x','y'),-1)

>>> dict4

{'y': -1, 'x': -1}

>>> dict5={}.fromkeys(('x','y'))

>>> dict5

{'y': None, 'x': None}

----------------------------------------------

>>> dict1

{}

>>> dict1['a']=1

>>> dict1[1]='ada'

>>> dict1[1.23]='adcd'

>>> dict1

{'a': 1, 1: 'ada', 1.23: 'adcd'}

--------------------------------------------------

### 访问字典中的值

-------------------------------------------------

>>> dict2

{'age': 18, 'name': 'python'}

>>> for key in dict2.keys():

... print key,dict2[key]

...

age 18

name python

>>> for key in dict2:

... print key,dict2[key]

...

age 18

name python

>>> dict2['name']

'python'

>>> for key,value in dict2.items():

... print key,value

...

age 18

name python

>>> dict2

{'age': 18, 'name': 'python'}

>>> print 'my name is %(name)s, i am %(age)d.' %dict2

my name is python, i am 18.

--------------------------------------------------

### 更新字典

---------------------------------------------------

>>> dict2

{'age': 18, 'name': 'python'}

>>> dict2['age']=20

>>> dict2['id']=20161202

>>> dict2

{'age': 20, 'name': 'python', 'id': 20161202}

--------------------------------------------------

### 删除字典元素和字典

-----------------------------------------------------

>>> dict2

{'age': 20, 'name': 'python', 'id': 20161202}

>>> dict3

{'a': 1, 'b': 2}

>>> **del dict3['a']**

>>> dict3

{'b': 2}

>>> **del dict3**

>>> dict3

Traceback (most recent call last):

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

NameError: name 'dict3' is not defined

>>> **dict2.pop**('id')

20161202

>>> dict2

{'age': 20, 'name': 'python'}

>>> **dict2.clear()**

>>> dict2

{}

-------------------------------------------------------

## 映射类型操作符

-------------------------------------------

>>> dict2

{'age': 18, 'name': 'python'}

>>> dict1

{'a': 1, 1: 'ada', 1.23: 'adcd'}

>>> dict1**<**dict2

False

>>> dict1**>**dict2

True

>>> dict1**==**dict2

False

----------------------------------------------------

>>> dict2

{'age': 18, 'name': 'python'}

>>> 'name' **in** dict2

True

>>> 'name' **not in** dict2

False

>>> 'id' in dict2

False

----------------------------------------------------

## 映射类型的内建函数和工厂函数

### 标准类型函数：type()、str()、cmp()

cmp()：首先是比较字典的大小（键值对个数），然后是键，最后是值。

------------------------------------------

>>> dict1

{'a': 1, 1: 'ada', 1.23: 'adcd'}

>>> dict2

{'age': 18, 'name': 'python'}

>>> type(dict2)

<type 'dict'>

>>> str(dict2)

"{'age': 18, 'name': 'python'}"

>>> cmp(dict2,dict1)

-1

>>> cmp(dict1,dict2)

1

>>> cmp(dict1,dict1)

0

-------------------------------------------

字典比较的算法：



### 工厂函数：dict()

----------------------------------------

>>> dict()

{}

>>> dict(zip(('x','y'),(1,2)))

{'y': 2, 'x': 1}

>>> dict([['x',1],['y',2]])

{'y': 2, 'x': 1}

>>> dict([('xy'[i-1],i) for i in range(1,3)])

{'y': 2, 'x': 1}

>>> dict(x=1,y=2)

{'y': 2, 'x': 1}

>>> dict1=dict(x=1,y=2)

>>> dict2=dict(\*\*dict1)

>>> dict2

{'y': 2, 'x': 1}

>>> dict2=dict1.copy()

>>> dict2

{'y': 2, 'x': 1}

------------------------------------

### 内建函数：len()、hash()

hash()：返回对象的哈希值，可以用来判断某个对象是否可以做一个字典的键。

------------------------------------------

>>> len(dict1)

2

>>> hash('')

0

>>> hash('abc')

-1600925533

>>> hash({})

Traceback (most recent call last):

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

TypeError: unhashable type: 'dict'

>>> hash([])

Traceback (most recent call last):

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

TypeError: unhashable type: 'list'

----------------------------------------------

## 映射类型内建方法

---------------------------------------------------------

>>> dict1

{'y': 2, 'x': 1}

>>> **dict1.clear()**

>>> dict1

{}

>>> dict1=**dict2.copy()**

>>> dict1

{'y': 2, 'x': 1}

>>> **dict.fromkeys**([1,2,3])

{1: None, 2: None, 3: None}

>>> dict.fromkeys([1,2,3],'a')

{1: 'a', 2: 'a', 3: 'a'}

>>> dict2**.get**('x')

1

>>> dict2.get('z')

>>> dict2.get('z',3)

3

>>> dict2

{'y': 2, 'x': 1}

>>> dict2**.has\_key**('x')

True

>>> dict2.has\_key('z')

False

>>> dict2**.items**()

[('y', 2), ('x', 1)]

>>> dict2**.iteritems**()

<dictionary-itemiterator object at 0x004F42D0>

>>> dict2**.iterkeys**()

<dictionary-keyiterator object at 0x020E9BD0>

>>> dict2**.keys**()

['y', 'x']

>>> dict2**.pop**('x')

1

>>> dict2

{'y': 2}

>>> dict2.**setdefault**('z',3)

3

>>> dict2

{'y': 2, 'z': 3}

>>> dict2.**setdefault**('z',5)

3

>>> dict2

{'y': 2, 'z': 3}

>>> dict1**.update**(dict2)

>>> dict1

{'y': 2, 'x': 1, 'z': 3}

>>> dict2.**values**()

[2, 3]

>>> dict2**.itervalues**()

<dictionary-valueiterator object at 0x004F42D0>

----------------------------------------------------------

>>> dict1

{'y': 2, 'x': 1, 'z': 3}

>>> for key in **sorted(dict1)**:

... print key,dict1[key]

...

x 1

y 2

z 3

-----------------------------------------------------

## 字典的键

### 不允许一个键对应多个值，当有键发生冲突时，取最后的赋值

--------------------------------

>>> dict1={'x':1,'x':2}

>>> dict1

{'x': 2}

>>> dict1['x']=3

>>> dict1

{'x': 3}

---------------------------------------

### 键必须是可哈希的

所有不可变的类型都是可哈希的。

也有一些可变对象（很少）是可哈希的，如：一个实现了\_\_hash\_()特殊方法的类。因为\_\_hash\_\_()方法返回一个整数，所以仍然是用不可变的值（做字典的键）。

## 集合类型

集合（sets）有两种不同的类型，可变集合（set）和不可变集合（frozenset）。

可变集合可以添加和删除元素，不可变集合则不可以；

可变集合不是可哈希的，既不能做字典的键也不能做其他集合中的元素；

不可变集合是可哈希的，能被用做字典的键也可以做其他集合中的元素。

### 创建集合类型和给集合赋值

集合被创建的唯一方法：用集合的工厂方法：set()和frozenset()。

--------------------------------------

>>> s = set('nihao python')

>>> s

set(['a', ' ', 'i', 'h', 'o', 'n', 'p', 't', 'y'])

>>> t = frozenset('helloworld')

>>> t

frozenset(['e', 'd', 'h', 'l', 'o', 'r', 'w'])

>>> type(s)

<type 'set'>

>>> type(t)

<type 'frozenset'>

>>> len(s)

9

>>> len(t)

7

>>> 'h' in s

True

>>> 't' in t

False

>>> for i in s:

... print i

...

a

i

h

o

n

p

t

y

-------------------------------------

### 更新集合

--------------------------------------

>>> s

set(['a', ' ', 'i', 'h', 'o', 'n', 'p', 't', 'y'])

>>> s.add('a')

>>> s

set(['a', ' ', 'i', 'h', 'o', 'n', 'p', 't', 'y'])

>>> s.add('z')

>>> s

set(['a', ' ', 'i', 'h', 'o', 'n', 'p', 't', 'y', 'z'])

>>> s.remove(' ')

>>> s

set(['a', 'i', 'h', 'o', 'n', 'p', 't', 'y', 'z'])

>>> s -= set('papi')

>>> s

set(['h', 'o', 'n', 't', 'y', 'z'])

--------------------------------------------------

删除集合中的成员和集合

------------------------------------------------

>>> s=set([1,2,3])

>>> s

set([1, 2, 3])

>>> s.remove(1)

>>> s

set([2, 3])

>>> del s

>>>

---------------------------------------

## 集合类型操作符

### 标准类型操作符（所有的集合类型）

**成员关系（in、not in）**

-------------------------------------------

>>> s=set([1,2,3])

>>> 1 in s

True

>>> 4 not in s

True

>>> 4 in s

False

--------------------------------------------

**集合等价/不等价（==、!=）**

----------------------------------------------

>>> s

set([1, 2, 3])

>>> t

frozenset([1, 2, 3])

>>> s == t

True

>>> s != t

False

>>> set('posh')==set('shop')

True

------------------------------------------------

**子集/超集（<、<=、>、>=）**

-----------------------------------------

>>> set('pop') < set('opse')

True

>>> set('pop') >= set('opse')

False

-------------------------------------------

### 集合类型操作符（所有的集合类型）

#### 联合（|）

等价于 or

两个集合的联合是一个新集合，该集合中的每个元素至少是其中一个集合的成员

等价的方法：union()

--------------------------------------------

>>> s1

set([1, 2, 3])

>>> s2

frozenset([2, 3, 4])

>>> s1 | s2

set([1, 2, 3, 4])

-------------------------------------------------

#### 交集（&）

等价于 and

两个集合的交集是一个新集合，该集合中的每一个元素同时是两个集合中的成员

等价的方法：intersection()

--------------------------------------------------

>>> s1

set([1, 2, 3])

>>> s2

frozenset([2, 3, 4])

>>> s1 & s2

set([2, 3])

--------------------------------------------------

#### 差补/相对补集（-）

两个集合（s1和s2）的差补或相对补集是指一个集合C，该集合中的元素，只属于集合s1，而不属于集合s2。

等价的方法：difference()

-------------------------------------------------------

>>> s1

set([1, 2, 3])

>>> s2

frozenset([2, 3, 4])

>>> s1 - s2

set([1])

---------------------------------------------------------

#### 对称差分（^）

对称差分是集合的XOR（又称“异或”，exclusive disjunction）

两个集合（s1和s2）的对称差分是指另外一个集合C，该集合中的元素，只能是属于集合s1或者集合s2的成员，不能同时属于两个集合。

等价的方法：symmetric\_difference()

---------------------------------------------------------

>>> s1

set([1, 2, 3])

>>> s2

frozenset([2, 3, 4])

>>> s1 ^ s2

set([1, 4])

------------------------------------------------------------

#### 混合集合类型操作

可变集合和不可变集合混合操作，若左右两个操作数的类型相同，则结果类型是相同的；若左右两个操作数的类型不同，则结果类型与左操作数类型相同。注：加号不是集合类型的操作符：

--------------------------------------------------------------

>>> s1 + s2

Traceback (most recent call last):

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

TypeError: unsupported operand type(s) for +: 'set' and 'frozenset'

>>> s1 | s2

set([1, 2, 3, 4])

>>> s2 | s1

frozenset([1, 2, 3, 4])

>>> s1 & s2

set([2, 3])

>>> s2 & s1

frozenset([2, 3])

>>> s1 - s2

set([1])

>>> s2 - s1

frozenset([4])

>>> s1 ^ s2

set([1, 4])

>>> s2 ^ s1

frozenset([1, 4])

-----------------------------------------------------------

### 集合类型操作符（仅适用于可变集合）

#### （Union）Update（|=）

从已存在的集合中添加（可能多个）成员

此方法与update()等价

--------------------------------------------

>>> s1

set([1, 2, 3])

>>> s1 |= set('haha')

>>> s1

set(['a', 1, 2, 3, 'h'])

>>> s1 |= frozenset('56')

>>> s1

set(['a', 1, 2, 3, '5', '6', 'h'])

-------------------------------------------------

#### Retention/Intersection Update（&=）

保留（或交集更新）操作保留与其他集合的共有成员

此方法与intersection\_update()等价

--------------------------------------------------

>>> s1

set(['a', 1, 2, 3, 'h', '5', '6'])

>>> s1 &= set('a2h')

>>> s1

set(['a', 'h'])

>>> s1 &= frozenset('a')

>>> s1

set(['a'])

---------------------------------------------------

#### Difference Update（-=）

对集合s1和s2进行差更新操作s1-=s2,差更新操作会返回一个集合，该集合中的成员是集合s1去除掉集合s2中元素后剩余的元素

此方法与difference\_update()等价

-----------------------------------------------------------

>>> s1

set(['a', 1, 2, 3, 'h', '5', '6'])

>>> s2

frozenset([2, 3, 4])

>>> s1 -= s2

>>> s1

set(['a', 1, 'h', '5', '6'])

-----------------------------------------------------------

Symmetric Difference Update（^=）

对集合s1和集合s2进行对称差分更新操作（s1^=s2），对称差分更新操作返回一个集合，该集合中的成员仅是原集合s1或仅是另一集合s2中的成员

此方法与symmetric\_difference\_update()等价

------------------------------------------------------------

>>> s1

set([1, 2, 3])

>>> s2

frozenset([2, 3, 4])

>>> s1 ^= s2

>>> s1

set([1, 4])

--------------------------------------------------------------

## 内建函数

### 标准类型函数

len()

----------------------------------

>>> s1

set([1, 4])

>>> len(s1)

2

>>> s2

frozenset([2, 3, 4])

>>> len(s2)

3

-------------------------------------

### 集合类型工厂函数

set()和frozenset()，分别用来生成可变和不可变集合，参数必须是可迭代的，即一个序列，或迭代器，或支持迭代的一个对象

--------------------------------------

>>> set('abc')

set(['a', 'c', 'b'])

>>> set([1,2,3])

set([1, 2, 3])

>>> frozenset((1,2,3))

frozenset([1, 2, 3])

>>> set()

set([])

>>> frozenset()

frozenset([])

--------------------------------------------

## 集合类型内建方法

### 方法（所有的集合方法）

s.issubset(t)：若s是t的子集，返回True，否则返回False

s.issuperset(t)：若s是t的超级，返回True，否则返回False

s.union(t)：返回一个新集合，该集合是s和t的并集

s.intersection(t)：返回一个新集合，该集合是s和t的交集

s.difference(t)：返回一个新集合，该集合是s的成员，不是t的成员

s.symmetric\_difference(t)：返回一个新集合，该集合是s或t的成员，不是s和t共有的成员

s.copy()：返回一个新集合，它是集合s的浅复制

---------------------------------------------

>>> s=set([1,2,3])

>>> t=set([2,3,4])

>>> s.issubset(t)

False

>>> s.issuperset(t)

False

>>> s.union(t)

set([1, 2, 3, 4])

>>> s.intersection(t)

set([2, 3])

>>> s.difference(t)

set([1])

>>> s.symmetric\_difference(t)

set([1, 4])

>>> s.copy()

set([1, 2, 3])

-----------------------------------------------

### 方法（仅适用于可变集合）

s.update(t)：用t中的元素修改s，即，s现在包含s或t的成员

s.intersection\_update(t)：s中的成员是共同属于s和t的元素

s.difference\_update(t)：s中的成员是属于s但不包含在t中的元素

s.symmetric\_difference\_update(t)：s中的成员更新为那些包含在s或t中，但不是s和t共有的元素

s.add(obj)：在集合s中添加对象obj

s.removeobj)：从集合s中删除对象obj；若obj不是集合s中的元素，则引发KeyError错误

s.discard(obj)：若obj是集合s中的元素，则从集合s中删除对象obj

s.pop()：删除集合s中的任意一个对象，并返回它

s.clear()：删除集合s中的所有元素

-------------------------------------------------

set([1, 2, 3])

>>> s.update(t)

>>> s

set([1, 2, 3, 4])

>>> s=set([1,2,3])

>>> s.intersection\_update(t)

>>> s

set([2, 3])

>>> s=set([1,2,3])

>>> s.difference\_update(t)

>>> s

set([1])

>>> s=set([1,2,3])

>>> s.symmetric\_difference\_update(t)

>>> s

set([1, 4])

>>> s.add(2)

>>> s

set([1, 2, 4])

>>> s.add(3)

>>> s

set([1, 2, 3, 4])

>>> s.remove(4)

>>> s

set([1, 2, 3])

>>> s.remove(4)

Traceback (most recent call last):

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

KeyError: 4

>>> s.discard(3)

>>> s.discard(3)

>>> s

set([1, 2])

>>> s.pop()

1

>>> s.clear()

>>> s

set([])

------------------------------------------------------

# 条件和循环

## if语句

使用映射对象的一个最大好处就是它的搜索操作比类似if-elif-else语句或是for循环这样的序列查询要快很多。

### 条件表达式

x if c else y

-------------------------------

>>> 4 if 2>3 else 5

5

>>> 4 if 2<3 else 5

4

---------------------------------

## while语句

很有可能被设置为无限循环

## for语句

可以遍历序列成员，可以用在列表解析和生成器表达式中，会自动地调用迭代器的next()方法，捕获StopIteration异常并结束循环。

迭代序列有三种基本方法：

1. 通过序列项迭代

------------------------------

>>> for i in 'abcde':

... print i

...

a

b

c

d

e

-------------------------------

1. 通过序列索引迭代

----------------------------------

>>> s=[1,2,3,4]

>>> for i in range(len(s)):

... print s[i]

...

1

2

3

4

-----------------------------------

1. 使用项和索引迭代

------------------------------------

>>> s=[1,2,3,4]

>>> for i,j in enumerate(s):

... print i,j

...

0 1

1 2

2 3

3 4

--------------------------------------

sorted()、zip()：返回一个序列（列表）

reversed()、enumerate()：返回迭代器（类似序列）

## continue

当遇到continue时，首先程序会终止当前循环，并忽略剩余的语句，然后回到循环的顶端，在开始下一次迭代前，如果是条件循环，将验证条件表达式，如果是迭代循环，将验证是否还有元素可以迭代，只有在验证成功的情况下，才会开始下一次迭代。

## pass语句

不做任何事情，NOP（No Operation）无操作，可作为开发中的小技巧，可以先把结构定下来，标记以后需要完成的代码。

如：

def func():

pass

或

if x == ‘abc’:

pass

else

pass

另外，在异常处理中也被经常用到。

pass在任何需要语句块的地方都可以使用。

## else语句

else语句可以在while和for循环中使用。

在循环中使用时，else子句只在正常循环完成后执行，break语句也会跳过else块。

---------------------------------

>>> n = 5

>>> sum = 0

>>> while n>0:

... sum+=n

... n-=1

... else:

... print sum

...

15

---------------------------------

>>> for i in range(5):

... if i ==3:

... break

... else:

... print i

... else:

... print 'Done'

...

0

1

2

----------------------------------

## 迭代器和iter()函数

迭代器就是有一个next()方法的对象，而不是通过索引来计数。

使用迭代可以提升性能，可以创建更简洁可读的代码。

### 使用迭代器

1. 序列

------------------------------------------

>>> t=(12,'abc',12,34)

>>> i = iter(t)

>>> i.next()

12

>>> i.next()

'abc'

>>> i.next()

12

>>> i.next()

34

>>> i.next()

Traceback (most recent call last):

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

StopIteration

-----------------------------------------------

1. 字典和文件是另外两个可迭代的Python数据类型

字典的迭代器会遍历它的键

----------------------------------------------------------

>>> d={'a':1,'b':2}

>>> for i in d:

... print i

...

a

b

>>> for k in d.iterkeys():

... print k

...

a

b

>>> for v in d.itervalues():

... print v

...

1

2

>>> for k,v in d.iteritems():

... print k,v

...

a 1

b 2

-----------------------------------------------------------

文件对象生成的迭代器会自动调用readline()方法

-----------------------------------------------------------

>>> fp = open("d:\\test1.txt")

>>> for eachLine in fp:

... print eachLine,

...

123

abc

456

def

>>> fp.close()

----------------------------------------------------------------

### 创建迭代器

对一个对象调用iter()就可以得到它的迭代器：iter(obj)、iter(func, sentinel)

iter(func, sentinel)：重复地调用func，直到迭代器的下个值等于sentinel

## 列表解析

语法：

[expr for iter\_var in iterable]

-----------------------------------------

>>> [x for x in range(10)]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> [x for x in range(10) if x % 2 ==0]

[0, 2, 4, 6, 8]

---------------------------------------------

## 生成器表达式

语法：

(expr for iter\_var in iterable)

--------------------------------------------

>>> (x+1 for x in range(5))

<generator object <genexpr> at 0x0251D418>

>>> for i in (x+1 for x in range(5)):

... print i

...

1

2

3

4

5

---------------------------------------------

# 文件和输入输出

### 文件内建函数[open()和file()]

#### open()

open()函数成功打开文件后返回一个文件对象，否则报错(IOError)。

语法：

file\_boject = open(file\_name,access\_mode=’r’,buffering=-1)

file\_name：可以是相对路径，也可以是绝对路径

access\_mode：文件的打开模式（’r’、’w’、’a’、’U’）。使用’r’或’U’打开的文件必须是已经存在的；使用’w’打开的文件，若存在则清空；用’a’打开的文件是在末尾追加数据，若文件不存在，会自动创建

buffering：用于指示访问文件所采用的缓冲方式。0表示不缓冲，1表示只缓冲一行数据，任何大于1的值代表使用给定值作为缓冲区大小，不提供参数或者为负值代表使用系统默认缓冲机制。

文件对象的访问模式：

r：只读，w：写（清空内容），a：追加，r+：读写，w+：读写，a+：读写，rb：二进制读，wb：二进制写，ab：二进制追加，rb+：二进制读写，wb+：二进制读写，ab+二进制读写

rU或Ua：以读方式打开，同时提供通用换行符支持（PEP 278）

#### file()

file()和open()函数具有相同的功能，可以任意替换

##### 通用换行符支持（UNS）

使用’U’标志打开文件的时候，所有的行分隔符（或行结束符，无论它原来是什么）通过Python的输入方法返回时都会被替换为换行符NEWLINE(\n)。

UNS只用于读取文本文件。在编译Python的时候，UNS默认是打开的。

## 文件内建方法

### 输入

read([size])：读取字节到字符串中，最多读取给定数目个字节，若没有给定size参数或size为负，文件将被读取至末尾。

readline([size])：读取打开文件的一行，然后整行，包括行结束符，作为字符串返回。size默认为-1，代表读至行结束符，若提供该参数，若超过size个字节后会返回不完整的行。

readlines()：读取所有行，然后把它们作为一个字符串列表返回。

### 输出

write()：把含有文本数据或二进制数据块的字符串写入到文件中去。

writelines()：接受一个字符串列表作为参数，将它们写入文件，行结束符并不会被自动加入，若需要的话，必须在调用writelines()前给每行结尾加上行结束符。

### 文件内移动

seek(offset,0/1/2)：可以再文件中移动游标到不同的位置。offset表示相对某个位置偏移量，第二个参数，0代表从文件开头算起，1代表从当前位置算起，2代表从文件末尾算起。

tell()：是对seek()的补充，返回当前文件游标所在位置。

### 文件迭代

一行一行访问文件：

for eachLine in f：

...

eachLine表示文本文件的一行（包括末尾的行结束符）

文件迭代更为高效，而且写（和读）这样的Python代码更容易。

### 其它

close()：关闭文件，若不显示地关闭文件，有可能丢失输出缓冲区的数据

fileno()：返回打开文件的描述符，一个整数，可以用在如os模块（os.read()）的一些底层操作上。

flush()：直接把内部缓冲区中的数据立刻写入文件，而不是被动地等待输出缓冲区被写入。

isatty()：当文件是一个leitty设备时返回True，否则返回False

truncate([size])：将文件截取到当前文件游标位置或者到给定的size，以字节为单位。

### 文件方法杂项

#### 行分隔符和其它文件系统的差异：

操作系统间的差异之一是它们所支持的行分隔符不同。在POSIX（Unix系列或Mac OS X）系统上，行分隔符是（\n），在旧的MacOS下是（\r），而DOS和Wind32系统下是（\r\n）。

另一个不同是路径分隔符。POSIX使用“/”，DOS和Windows使用“\”，旧版本的MacOS使用“:”，用来分隔文件路径名，标记当前目录和父目录。

#### 有助于跨平台开发的os模块属性

os.linesep：用于在文件中分隔行的字符串

os.sep：用来分隔文件路径名的字符串

os.pathsep：用于分隔文件路径的字符串

os.curdir：当前工作目录的字符串名称

os.pardir：（当前工作目录的）父目录字符串名称

注：print语句默认在输出内容末尾后加一个换行符，而在语句后加一个逗号就可以避免这个行为。

## 文件内建属性

file.closed：True表示文件已经关闭，否则为False

file.encoding：文件所使用的编码若file.encoding为None时使用系统默认编码

file.mode：文件打开时使用的访问模式

file.name：文件名

file.newlines：未读取到行分隔符时为None，只有一种行分隔符时为一个字符串，当文件有多种类型的行结束符时，则为一个包含所有当前所遇到的行结束符的列表

file.softspace：为0表示在输出一数据后，要加上一个空格符，1表示不加。

## 标准文件

只要程序一执行，就可以访问三个标准文件。分别是：标准输入（一般是键盘），标准输出（到显示器的缓冲输出）和标准错误（到屏幕的非缓冲输出）。这里的“缓冲”和“非缓冲”是指open()函数的第三个参数。这些文件沿用的是c语言中的命名，分别为：stdin、stdout、stderr。

print语句通常是输出到sys.stdout；而内建raw\_input()则通常从sys.stdin接受输入。

## 命令行参数

sys.argv：是命令行参数的列表（argument vector）

len(sys.argv)：是命令行参数的个数（也就是argc）（argument count）

脚本：

print sys.argv

print len(sys.argv)

输出：

C:\Users\liwang>python E:\ practice20161224.py 123 test haha python

['E:\\ practice20161224.py', '123', 'test', 'haha', 'python']

5

------------------------------------

## 文件系统

文件系统的访问大多通过Python的os模块实现。该模块是python访问操作系统功能的主要接口。

os模块提供一些常见文件或目录操作函数；os.path模块可以完成一些针对路径名的操作。

这两个模块提供了与平台和操作系统无关的统一的文件系统访问方法。

### os模块：

文件处理：

os.remove()：删除文件

----------------------------------------

>>> os.getcwd()

'C:\\Users\\liwang'

>>> os.chdir('d:\\test')

>>> os.getcwd()

'd:\\test'

>>> path = os.getcwd()

>>> os.listdir(path)

['ha.txt', 'haha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log']

>>> os.**remove**("haha.txt")

>>> os.listdir(path)

['ha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log']

-------------------------------------------------

os.rename()：重命名文件

--------------------------------------------------

>>> os.listdir(path)

['ha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log']

>>> os.rename("ha.txt","hahaha.txt")

>>> os.listdir(path)

['hahaha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log']

---------------------------------------------------

os.stat()：返回文件信息

---------------------------------------

>>> import os

>>> os.**stat**("d:")

nt.stat\_result(st\_mode=16895, st\_ino=0L, st\_dev=0L, st\_nlink=0, st\_uid=0, st\_gid

=0, st\_size=28672L, st\_atime=1484036957L, st\_mtime=1484036957L, st\_ctime=1438738

835L)

>>> os.**stat**("d:\\test\\test.log")

nt.stat\_result(st\_mode=33206, st\_ino=0L, st\_dev=0L, st\_nlink=0, st\_uid=0, st\_gid

=0, st\_size=9L, st\_atime=1482216734L, st\_mtime=1482736342L, st\_ctime=1482216734L

)

---------------------------------------------

os.utime()：更新时间戳，更新文件访问时间和修改时间

os.utime(path,(newatime,newmtime))：设置文件访问时间和修改时间为指定时间戳

os.utime(path,None)：设置文件访问时间和修改时间为当前时间的时间戳

------------------------------------------------

>>> os.stat("d:\\test\\test.log")

nt.stat\_result(st\_mode=33206, st\_ino=0L, st\_dev=0L, st\_nlink=0, st\_uid=0, st\_gid

=0, st\_size=9L, st\_atime=1482216734L, st\_mtime=1482736342L, st\_ctime=1482216734L

)

>>> os.utime("d:\\test\\test.log",(1400000000,1450000000))

>>> os.stat("d:\\test\\test.log")

nt.stat\_result(st\_mode=33206, st\_ino=0L, st\_dev=0L, st\_nlink=0, st\_uid=0, st\_gid

=0, st\_size=9L, st\_atime=1400000000L, st\_mtime=1450000000L, st\_ctime=1482216734L

)

>>> os.utime("d:\\test\\test.log",None)

>>> os.stat("d:\\test\\test.log")

nt.stat\_result(st\_mode=33206, st\_ino=0L, st\_dev=0L, st\_nlink=0, st\_uid=0, st\_gid

=0, st\_size=9L, st\_atime=1484534539L, st\_mtime=1484534539L, st\_ctime=1482216734L

)

--------------------------------------------------------------

os.tmpfile()：创建并打开（’wb’）一个新的临时文件

----------------------------------------------

>>> tmpfile = os.tmpfile()

>>> tmpfile.write("haha\n")

>>> tmpfile.write("123\n")

>>> tmpfile.seek(0,0)

>>> print tmpfile.read()

haha

123

>>> tmpfile.close()

------------------------------------------------

os.walk()：生成一个目录树下的所有文件名

---------------------------------------------------

>>> for root,dirs,files in os.walk('d:\\test'):

... print root

...

d:\test

d:\test\log

d:\test\log\2017-01-12

d:\test\log\2017-01-13

d:\test\log\2017-01-14

>>> for root,dirs,files in os.walk('d:\\test'):

... print root

... print dirs

... print files

...

d:\test

['log']

['hahaha.txt', 'ip.txt', 'myCal.html', 'ping.txt', 'test.log']

d:\test\log

['2017-01-12', '2017-01-13', '2017-01-14']

[]

d:\test\log\2017-01-12

[]

['012.log']

d:\test\log\2017-01-13

[]

['013.log']

d:\test\log\2017-01-14

[]

['014.log']

------------------------------------------------------------

目录/文件夹：

os.chdir()：改变当前工作目录

os.getcwd()：返回当前工作目录

os.listdir()：列出指定目录下的内容

--------------------------------------------------------

>>> os.getcwd()

'E:\\'

>>> os.chdir("d:\\test")

>>> os.getcwd()

'd:\\test'

>>> os.listdir(os.getcwd())

['hahaha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log']

----------------------------------------

os.mkdir()：创建目录

os.makedirs()：创建多层目录，中间没有的目录也会创建

-----------------------------------------------------------------

>>> os.mkdir("d:\\test\\123")

>>> os.makedirs("d:\\test\\test1\\test11")

>>> os.listdir(os.getcwd())

['123.txt', 'hahaha.txt', 'ip.txt', 'log', 'myCal.html', 'ping.txt', 'test.log',

'test1']

---------------------------------------------------------------

os.rmdir()：删除目录

os.removedirs()：删除多层目录，从最下级开始遍历，只要目录为空就删除，直到目录不为空

-------------------------------------------------------------

>>> os.rmdir("d:\\test\\123")

>>> os.removedirs("d:\\test\\test1\\test11\\test111")

---------------------------------------------------------------

### os.path模块：

分隔：

os.path.basename()：去掉目录路径，返回文件名

-----------------------------------------

*>>> os.path.basename('d:\\test\\test.log')*

*'test.log'*

---------------------------------------------

os.path.dirname()：去掉文件名，返回目录路径

------------------------------------------------

*>>> os.path.dirname("d:\\test\\tset.log")*

*'d:\\test'*

------------------------------------------------------

os.path.join()：将分离的各部分组合成一个路径名

-----------------------------------------------------

*>>> os.path.join("d:\\test","123.txt")*

*'d:\\test\\123.txt'*

----------------------------------------------------

os.path.split()：返回(dirname(), basename())元组

-----------------------------------------------------

*>>> os.path.split("d:\\test\\123.txt")*

*('d:\\test', '123.txt')*

-------------------------------------------------------

os.path.splitdrive()：返回(drivename, pathname)元组，可以用来获取盘符

--------------------------------------------------------

*>>> os.path.splitdrive("d:\\test\\123.txt")*

*('d:', '\\test\\123.txt')*

-----------------------------------------------------------

os.path.splitext()：返回(filename, extension)元组，可以用来获取文件后缀名

---------------------------------------------------------------

>>> os.path.splitext("d:\\test\\123.txt")

('d:\\test\\123', '.txt')

----------------------------------------------------------------

信息：

os.path.getatime()：返回最近访问时间

os.path.getctime()：返回文件创建时间

os.path.getmtime()：返回最近文件修改时间

os.path.getsize()：返回文件大小（以字节为单位）

----------------------------------------------------------------------

*>>> filepath = "d:\\test\\test.log"*

*>>> os.path.getatime(filepath)*

*1484534539.548*

*>>> os.path.getctime(filepath)*

*1482216734.0950682*

*>>> os.path.getmtime(filepath)*

*1484534539.548*

*>>> os.path.getsize(filepath)*

*9L*

-----------------------------------------------------------------

查询：

os.path.exists()：指定路径（文件或目录）是否存在

-----------------------------------------------

*>>> os.path.exists("d:\\test")*

*True*

*>>> os.path.exists("d:\\test\\test.log")*

*True*

*>>> os.path.exists("d:\\test\\test12.log")*

*False*

*>>> os.path.exists("d:\\test123")*

*False*

----------------------------------------------------

os.path.isabs()：指定路径是否为绝对路径

-----------------------------------------------------

*>>> os.path.isabs("d:\\test\\tse")*

*True*

*>>> os.path.isabs("test\\tse")*

*False*

*>>> os.path.isabs("\\test\\tse")*

*True*

------------------------------------------------------

os.path.isdir()：指定路径是否存在且为一个目录

-------------------------------------------------------

*>>> os.path.isdir("d:\\test")*

*True*

*>>> os.path.isdir("d:\\test123")*

*False*

*>>> os.path.isdir("d:\\test\\test.log")*

*False*

----------------------------------------------------------

os.path.isfile()：指定路径是否存在且为一个文件

---------------------------------------------------------

>>> os.path.isfile("d:\\test\\test.log")

True

>>> os.path.isfile("d:\\test\\test123.log")

False

>>> os.path.isfile("d:\\test")

False

------------------------------------------------------------

## 永久存储模块

可以把用户的数据归档保存起来供以后使用，大部分呢永久性存储模块是用来存储字符串数据的。

### pickle和marshal模块

用来转换并储存Python对象，为保存和传输提供方便

marshal只能处理简单的Python对象（数字、序列、映射，以及代码对象），而pickle还可以处理递归对象，被不同地方多次引用的对象，以及用户定义的类和实例，pickle还有一个增强的版本cPickle，使用C实现了相关的功能。

cPickle模块中的两个主要函数：dump()和load()

dump()接受一个文件句柄和一个数据对象作为参数，把数据对象以特定的格式保存到给定文件中；

load()从文件中取出已保存的对象。

-------------------------------------------

*>>> import cPickle*

*>>> content = ["abc","python",123]*

*>>> f = file("d:\\test\\test.data",'w')*

*>>> cPickle.****dump****(content,f)*

*>>> f.close()*

*>>> f = file("d:\\test\\test.data")*

*>>> cc = cPickle.****load****(f)*

*>>> print cc*

*['abc', 'python', 123]*

--------------------------------------------------

## 相关模块

文件相关模块：

base64：提供二进制字符串和文本字符串间的编码、解码操作

binascii：提供二进制和ASCII编码的二进制字符串间的编码、解码操作

bz2：访问BZ2格式的压缩文件

csv：访问csv文件（逗号分隔文件）

filecmp：用于比较目录和文件

fileinput：提供多个文本文件的行迭代器

tarfile：读写TAR归档文件，支持压缩文件

zipfile：用于读取ZIP归档文件的工具

gzip/zlib：读写GNU zip（gzip）文件（压缩需zlib模块）

shutil：提供高级的文件访问功能，包括复制文件，复制文件的访问权限，递归地目录树复制等

# 错误和异常

## 什么是异常

### 错误

错误是语法或逻辑上的，语法错误导致不能被解释器解释或编译器无法编译，这些错误必须在程序执行前纠正。逻辑错误是由于不完整或不合法的输入所致。

当python检测到一个错误时，解释器就会指出的当前流已经无法继续执行下去，这时候就出现了异常。

### 异常

因程序出现了错误而在正常控制流以外采取的行为。这个行为分为两个阶段：引起异常发生的错误；检测（和采取可能的措施）阶段。

引发和处理异常，可以再错误发生时更直接地控制它们，有了运行时管理错误的能力，应用程序的健壮性会有很大的提高。

## Python中的异常

NameError：尝试访问一个未定义的变量

------------------------------------------------

*>>> abc*

*Traceback (most recent call last):*

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

*NameError: name 'abc' is not defined*

----------------------------------------------------

ZeroDivisionError：除数为零

--------------------------------------------

*>>> 1/0*

*Traceback (most recent call last):*

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

*ZeroDivisionError: integer division or modulo by zero*

-------------------------------------------------

SyntaxError：语法错误

是唯一不是在运行时发生的异常，而是在编译时发生的。

-----------------------------------------------

*>>> for*

*File "<stdin>", line 1*

*for*

*^*

*SyntaxError: invalid syntax*

---------------------------------------------------

IndexError：索引超出序列范围

----------------------------------------------------

*>>> a=[1,2]*

*>>> a[3]*

*Traceback (most recent call last):*

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

*IndexError: list index out of range*

---------------------------------------------------------

KeyError：请求一个不存在的字典关键字

------------------------------------------------------------

*>>> d={}*

*>>> d[1]*

*Traceback (most recent call last):*

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

*KeyError: 1*

----------------------------------------------------------

IOError：输入/输出错误

----------------------------------------------------------

*>>> f = open("d:\\123")*

*Traceback (most recent call last):*

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

*IOError: [Errno 2] No such file or directory: 'd:\\123'*

--------------------------------------------------------------

AttributeError：尝试访问未知的对象属性

-------------------------------------------------------------

*>>> import os*

*>>> os.hhh()*

*Traceback (most recent call last):*

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

*AttributeError: 'module' object has no attribute 'hhh'*

------------------------------------------------------------------

TypeError：参数类型错误

------------------------------------------------------------

*>>> float([])*

*Traceback (most recent call last):*

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

*TypeError: float() argument must be a string or a number*

---------------------------------------------------------------

## 检测和处理异常

异常可以通过try语句来检测。任何在try语句块里的代码都会被监测，检查有无异常发生。

try语句有两种主要形式：try-except和try-finally。一个try语句可以对应一个或多个except子句，但只能对应一个finally子句，或是一个try-except-finally复合语句。

try-except语句可以检测和处理异常，也可以添加一个else子句处理没有检测到异常的时候指定的代码；而try-finally只检测异常并做一些必要的清除工作（无论有没有异常），没有任何异常处理设施。

## try-except语句

最常见的try-except语句语法：

try:

try\_suite #监控这里的异常

except Exception[, reason]:

except\_suite #异常处理代码

----------------------------------------

*>>> try:*

*... 1/0*

*... except ZeroDivisionError,e:*

*... print e*

*...*

*integer division or modulo by zero*

--------------------------------------------

### 带有多个except的try语句

语法：

except Exception1[, reason1]:

suite\_for\_exception\_Exception1

except Exception2[, reason2]:

suite\_for\_exception\_Exception1

.

.

若检测到指定异常，则跳转到指定的异常处理代码块。

----------------------------------------

*>>> def safe\_float(obj):*

*... try:*

*... retval = float(obj)*

*... except ValueError:*

*... retval = "ValueError!"*

*... except TypeError:*

*... retval = "TypeError!"*

*... return retval*

*...*

*>>> safe\_float('a')*

*'ValueError!'*

*>>> safe\_float([])*

*'TypeError!'*

-------------------------------------------

### 处理多个异常的except语句

一个except子句里处理多个异常，要求异常被放在一个元组里：

except (Exc1[, Exc2[, ... ExcN]])[, reason]:

suite\_for\_Exc1\_to\_ExcN

--------------------------------------------

*>>> def safe\_float(obj):*

*... try:*

*... retval = float(obj)*

*... except (ValueError, TypeError):*

*... retval = "argument must be a number or numeric string"*

*... return retval*

*...*

*>>> safe\_float('a')*

*'argument must be a number or numeric string'*

*>>> safe\_float([])*

*'argument must be a number or numeric string'*

*>>> safe\_float('123')*

*123.0*

*>>> safe\_float(123)*

*123.0*

---------------------------------------------------

### 捕获所有异常

语法一：（其中e是Exception异常类的实例）

try:

...

except Exception,e:

...

语法二：(不推荐，因为这种方法不会给我们任何关于可能发生的错误信息)

try:

....

except:

....

### else子句

在try范围中没有异常被检测到时，执行else子句

try:

...

except:

...

else:

...

### finally子句

finally子句无论异常是否发生，是否捕捉都会执行finally子句。

try:

...

except:

...

else:

…

finally:

...

### try-finally语句

try:

try\_suite

finally:

finally\_suite #无论如何都执行

当try范围中发生一个异常时，会立即跳转到finally语句段，当finally中的所有代码都执行完毕后，会继续向上一层引发异常。

-------------------------------------------------

*try:*

*fp = open("d:\\test\\test.log",'r')*

*content = fp.readlines()*

*except IOError:*

*print "IOError!"*

*finally:*

*# 检测某个变量是否有定义的三种方法：*

*if vars().has\_key('fp'):*

*# if 'fp' in dir():*

*# if 'fp' in locals().keys():*

*print "finally!"*

*fp.close()*

---------------------------------------------

# 检测某个变量是否有定义的三种方法：

1. 使用内置函数locals()

XXX in locals().keys()

1. 使用内置函数dir()

XXX in dir()

1. 使用内置函数vars()

vars().has\_key(XXX)

## 上下文管理

### with语句

with语句是用来简化代码的

\_\_enter\_\_()方法，完成with语句块执行前的所有准备工作。

\_\_exit\_\_()方法，当with语句块执行结束，无论是正常的还是异常的，都会调用\_\_exit\_\_()方法，\_\_exit\_\_()有三个参数，如果with语句块正常结束，三个参数全部是None，如果发生异常，三个参数的值分别等于调用sys.exc\_info()函数返回的三个值：类型（异常类）、值（异常实例）、和回溯（traceback），响应的回溯对象

上下文管理器主要作用于共享资源，因此\_\_enter\_\_()和\_\_exit\_\_()方法基本是干的需要分配和释放资源的低层次工作，比如：数据库连接，锁分配，信号量加减，状态管理，打开/关闭文件，异常处理，等等。

## 触发异常

raise语句

一般用法：raise [SomeException[, args[, traceback]]]

SomeException：是触发异常的名字，可以是类、实例、或一个字符串

args：参数，值，来传给异常。异常的参数总是作为一个元组传入。

traceback：当异常触发时新生成的一个用于异常-正常化（exception-normally）的追踪（traceback）对象，很少用到

## 断言

断言是一句必须等价于布尔真的判定，如果断言成功不采取任何措施，否则触发AssertionError（断言错误）的异常。

断言语句：assert语句

语法：assert expression[, arguments]

---------------------------------

*>>> assert 1 == 1*

*>>> assert 1 == 2*

*Traceback (most recent call last):*

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

*AssertionError*

*>>> assert 1 == 2, 'one does not equal two !'*

*Traceback (most recent call last):*

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

*AssertionError: one does not equal two !*

----------------------------------------

*>>> try:*

*... assert 1 == 2, 'one does not equal two !'*

*... except AssertionError,args:*

*... print '%s: %s' %(args.\_\_class\_\_.\_\_name\_\_, args)*

*...*

*AssertionError: one does not equal two !*

----------------------------------------------

## 标准异常

Python内建异常：

BaseException：所有异常的基类

SystemExit：python解释器请求退出

KeyboardInterrupt：用户中断执行（通常是输入^C）

Exception：常规错误的基类

StopIteration：迭代器没有更多的值

GeneratorExit：生成器（Generator）发生异常来通知退出

StandardError：所有的内建标准异常的基类

ArithmeticError：所有数值计算错误的基类

FloatingPointError：浮点计算错误

OverflowError：数值运算超出最大限制

ZeroDivisionError：除（或取模）零（所有数据类型）

AssertionError：断言语句失败

AttributeError：对象没有这个属性

EOFError：没有内建输入，到达EOF标记

EnvironmentError：操作系统错误的基类

IOError：输入/输出操作失败

OSError：操作系统错误

WindowsError：Windows系统调用失败

ImportError：导入模块/对象失败

KeyboardInterrupt：用户中断操作（通常是输入^C）

LookupError：无效数据查询的基类

IndexError：序列中没有此索引

KeyError：映射中没有这个键

MemoryError：内存溢出错误（对于Python解释器不是致命的）

NameError：未声明/初始化对象（没有属性）

UnboundLocalError：访问未初始化的本地变量

ReferenceError：弱引用（Weak reference）试图访问已经垃圾回收了的对象

RuntimeError：一般的运行时错误

NotImplementedError：尚未实现的方法

SyntaxError：Python语法错误（唯一的在编译时发生的异常）

IndentationError：缩进错误

TabError：Tab和空格混用

TypeError：对类型无效的操作

ValueError：传入无效的参数

UnicodeError：Unicode相关的错误

UnicodeDecodeError：Unicode解码时的错误

UnicodeEncodeError：Unicode编码时的错误

UnicodeTranslateError：Unicode转换时的错误

Warning：警告的基类

DeprecationWarning：关于被弃用的特征的警告

FutureWarning：关于构造将来语义会有改变的警告

OverflowWarning：旧的关于自动提升为长整形的警告

PendingDeprecationWarning：关于特性将会被废弃的警告

RuntimeWarning：可疑的运行时行为（runtime behavior）的警告

SyntaxWarning：可疑的语法的警告

UserWarning：用户代码生成的警告

SystemExit，KeyboardInterrupt和Exception是直接从BaseException派生的异常子类。其他所有的内建异常都是Exception的子类。

# 函数和函数式编程

## 什么是函数

函数就是将重复代码隔离成易于管理的小块，可以复用。

### 关键字参数

让调用者通过函数调用中的参数名字来区分参数。这样规范允许参数缺失或者不按顺序。

例：

def foo(x,y):

pass

标准调用：foo(1, 2)

关键字调用：foo(x=1, y=2)、foo(y=2, x=1)

### 参数组

把元组或字典作为参数组传递给函数：

func(\*tuple\_grp\_noknow\_args, \*\*dict\_grp\_kw\_args)

tuple\_grp\_noknow\_args是以元组形式体现的非关键字参数组；

dict\_grp\_kw\_args是装有关键字参数的字典。

## 创建函数

语法：

**def** function\_name(args):

“function\_documentation\_string” #文档字符串

function\_body\_suite

查看文档字符串的方式：

1、help(function\_name)

2、function\_name.\_\_doc\_\_

### 内部/内嵌函数

在函数体内创建的另外一个函数（对象）。

最明显的创造内部函数的方法是在外部函数的定义体内定义函数（用def 关键字）；

另外一个函数体内创建函数对象的方式是使用lambda语句。

### 函数装饰器

语法：

以@开头，接着是装饰器函数的名字和可选的参数。紧跟着装饰器声明的是被修饰的函数和装饰函数的可选参数。如：

@decorator(dec\_opt\_args)

def func2Bdecorated(func\_opt\_args):

.

.

静态方法@staticmethod和类方法@classmethod，通过类名来调用：

class MyClass(object):

@staticmethod

def staticFoo():

…

@classmethod

def classFoo(cls):

…

#### 有参数和无参数的装饰器

@deco

def foo(): pass

等价于

foo = deco(foo)

@decomaker(deco\_args)

def foo(): pass

等价于

foo = decomaker(deco\_args)(foo)

含有多个装饰器的例子：

@deco1(deco\_args)

@deco2

def func(): pass

等价于

func = deco1(deco\_args)(deco2(func))

#### 什么是装饰器

装饰器实际就是函数。

可以在装饰器中置入通用功能的代码来降低程序复杂度。

例如，可以用装饰器来：

* 引入日志
* 增加计时逻辑来检测性能
* 给函数加入事务的能力

一个使用函数装饰器的例子：

-----------------------------------------------------

import time

def tsfunc(func):

def wrappedFunc():

print "[%s] %s() called" % (time.ctime(), func.\_\_name\_\_)

return func()

return wrappedFunc

@tsfunc

def foo(): pass

#等价于foo = tsfunc(foo)

foo()

time.sleep(4)

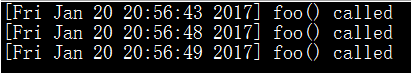
for i in range(2):

time.sleep(1)

foo()

------------------------------------------------

输出：



## 传递函数

在python中所有的对象都是通过引用来传递的，函数也不例外。函数也可以作为参数传递。

-----------------------------------------------------

>>> def foo():

... print 'in foo()'

...

>>> bar = foo

>>> bar()

in foo()

>>>

------------------------------------------------

上面例子中，当把foo赋值给bar时，bar和foo引用了同一个函数对象，“foo”（函数对象的引用），“foo()”（函数对象的调用）。

以下是函数作为参数传递的例子：

---------------------------------------------------

>>> def convert(func,seq):

... return [func(eachNum) for eachNum in seq]

...

>>> myseq = [12,1.23,123L,-5.6e8]

>>> print convert(int,myseq)

[12, 1, 123, -560000000]

>>> print convert(float,myseq)

[12.0, 1.23, 123.0, -560000000.0]

>>> print convert(abs,myseq)

[12, 1.23, 123L, 560000000.0]

>>> print convert(long,myseq)

[12L, 1L, 123L, -560000000L]

------------------------------------------------------

## 函数式编程

### 匿名函数 lambda

----------------------------------------------------------

>>> lambda :True

<function <lambda> at 0x02490D30>

>>> true = lambda :True

>>> true()

True

>>> a = lambda x,y=2:x+y

>>> a(2)

4

>>> a(2,5)

7

>>> b = lambda \*z:z

>>> b(1)

(1,)

>>> b(1,'abc')

(1, 'abc')

------------------------------------------------------

### 内建函数filter()、map()、reduce()

**filter(func, seq)**：调用一个布尔函数func来迭代遍历每个seq中的元素；返回一个使func返回值为True的元素的序列

--------------------------------------------

>>> filter(lambda x:True if x>50 else False,[20,30,60,70])

[60, 70]

-----------------------------------------------

**map(func, seq1[, seq2......])**：将函数func作用于给定序列seq的每个元素，并用一个列表来提供返回值；如果func为None，func表现为一个身份函数，返回一个含有每个序列中元素集合的n个元组的列表

--------------------------------------------------

>>> map(None,[1,2,3],[4,5,6],[7,8,9])

[(1, 4, 7), (2, 5, 8), (3, 6, 9)]

>>> map(lambda x,y,z:x+y+z,[1,2,3],[4,5,6],[7,8,9])

[12, 15, 18]

----------------------------------------------------

**reduce(func, seq[, init])**：将二元函数（两个参数）func作用于seq序列的元素，每次携带一对（先前的结果以及下一个序列元素），连续的将现有的结果和下一个值作用在获得的随后的结果上，最后减少我们的序列为一个单一的返回值；如果初始值init给定，第一个比较会是init和第一个序列元素而不是序列的头两个元素。

------------------------------------------------------

>>> reduce(lambda x,y:x+y,range(1,10),10)

55

>>> reduce(lambda x,y:x\*y,range(1,10))

362880

--------------------------------------------------------

## 变量作用域

### 全局变量与局部变量

定义在函数内的变量是局部变量，作用域是函数体内；

一个模块的最高级别的变量有全局变量，作用域是模块内。

全局变量：存放在堆，一个特点是：除非被删除掉，否则它们存活到脚本运行结束，且对所有的函数，它们的值都是可以被访问的。

局部变量：存放在栈，只是暂时的存在，只依赖于定义它们的函数现阶段是否处于活动。

### global语句

在一个函数中使用全局变量时，需要使用global语句。如果不加global语句，是不允许对全局变量进行修改的。

global var1[, var2, ......]

--------------------------------------

>>> a = 3

>>> def foo():

... global a

... a += 1

... print a

...

>>> foo()

4

-----------------------------------------

### 变量作用域和命名空间

## 生成器

生成器不会把结果保存在一个序列中，而是保存生成器的状态，在每次进行迭代时返回一个值，直到遇到StopIteration异常结束。

语法：

生成器表达式：通列表解析语法，只不过把列表解析的[]换成()

----------------------------------

>>> gen = (x for x in range(3))

>>> gen.next()

0

>>> gen.next()

1

>>> gen.next()

2

>>> gen.next()

Traceback (most recent call last):

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

StopIteration

---------------------------------------

生成器函数：在函数中如果出现了yield关键字，那么该函数就不再是普通函数，而是生成器函数。

------------------------------------------

>>> def odd():

... n=1

... while True:

... yield n

... n += 2

...

>>> count = 0

>>> for i in odd\_num:

... if count >= 5:

... break

... print i

... count += 1

...

3

5

7

9

11

---------------------------------------------------

# 模块

import语句的模块顺序：

建议所有的模块在Python模块的开头部分导入。导入顺序最好按照：

Python标准库模块、Python第三方模块、应用程序自定义模块

from module import \* ：不是良好的编程风格，因为很可能覆盖当前命名空间中现有的名字。

globals()和locals()：分别返回调用者全局和局部命名空间的字典。

在一个函数内部，局部命名空间代表在函数执行时候定义的所有名字，locals()函数返回的就是包含这些名字的字典。globals()会返回函数可访问的全局名字。

------------------------------------------------

def foo():

print "calling foo()..."

aString = 'bar'

anInt = 123

print "foo()'s globals:",globals().keys()

print "foo()'s locals:",locals().keys()

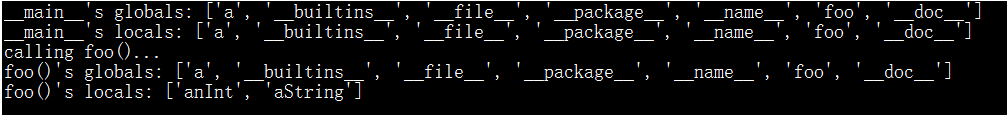
a = 100

print "\_\_main\_\_'s globals:",globals().keys()

print "\_\_main\_\_'s locals:",locals().keys()

foo()

输出：



reload()：可以重新导入一个已经导入的模块

## 包

一个有层次的文件目录结构

定义了一个由模块和子包组成的Python应用程序执行环境

# 面向对象编程

在Python中，面向对象编程主要有两个主题，就是类和类实例。

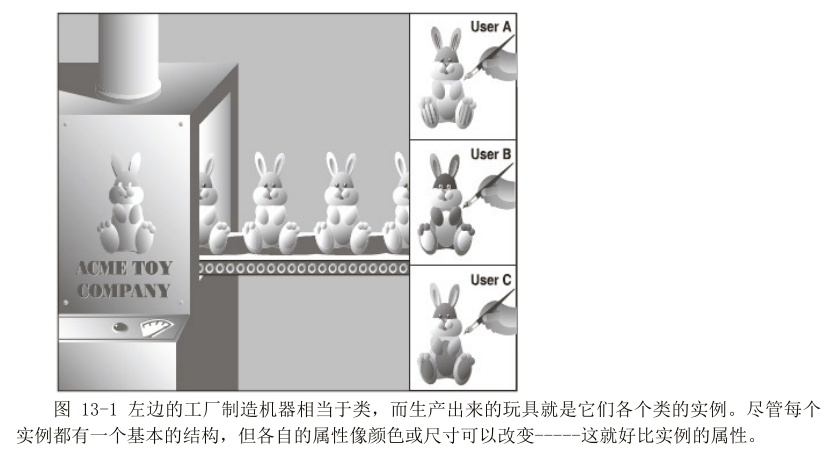
类是对象的定义；实例是“真正的实物”，它存放了类中所定义的对象的具体信息。

类名由大写字母打头。

class MyNewObjectType:

‘define MyNewObjectType classic class’

class\_suite



创建一个实例的过程称作实例化。

myFirstObject = MyNewObjectType()

创建一个Person类：

class Person(object):

def \_\_init\_\_(self,name,sex,age):

self.name = name

self.sex = sex

self.age = age

def getName(self):

return self.name

创建一个Student子类：

class Student(Person):

def \_\_init\_\_(self,name,sex,age,grade,id):

Person.\_\_init\_\_(name,sex,age)

self.grade = grade

self.id = id

def getId(self):

return self.id

### 类的属性

查看类的属性有两种方法：使用dir()内建函数；通过访问类的字典属性\_\_dict\_\_。

----------------------------------------------

>>> class Person(object):

... def \_\_init\_\_(self,name):

... self.name = name

... def getName(self):

... return self.name

...

>>> dir(Person)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribut

e\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_e

x\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_

\_weakref\_\_', 'getName']

>>> print Person.\_\_dict\_\_

{'\_\_module\_\_': '\_\_main\_\_', 'getName': <function getName at 0x026F29F0>, '\_\_dict\_

\_': <attribute '\_\_dict\_\_' of 'Person' objects>, '\_\_weakref\_\_': <attribute '\_\_wea

kref\_\_' of 'Person' objects>, '\_\_doc\_\_': None, '\_\_init\_\_': <function \_\_init\_\_ at

0x026F29B0>}

--------------------------------------------------

### 特殊的类属性

对任何类C，显示了类C的所有特殊属性：

C.\_\_name\_\_：类C的名字（字符串）

C.\_\_doc\_\_：类C的文档字符串

C.\_\_bases\_\_：类C的所有父类构成的元组

C.\_\_dict\_\_：类C的属性

C.\_\_module\_\_：类C定义所在的模块

C.\_\_class\_\_：实例C对应的类（仅新式类中）

------------------------------------------

>>> class Person(object):

... def \_\_init\_\_(self,name):

... self.name = name

... def getName(self):

... return self.name

...

>>> Person.\_\_name\_\_

'Person'

>>> Person.\_\_doc\_\_

>>> Person.\_\_bases\_\_

(<type 'object'>,)

>>> Person.\_\_dict\_\_

dict\_proxy({'\_\_module\_\_': '\_\_main\_\_', 'getName': <function getName at 0x026F29F0

>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'Person' objects>, '\_\_weakref\_\_': <attri

bute '\_\_weakref\_\_' of 'Person' objects>, '\_\_doc\_\_': None, '\_\_init\_\_': <function

\_\_init\_\_ at 0x026F29B0>})

>>> Person.\_\_module\_\_

'\_\_main\_\_'

>>> Person.\_\_class\_\_

<type 'type'>

>>> person = Person('123')

>>> person.\_\_class\_\_

<class '\_\_main\_\_.Person'>

------------------------------------------------

### 特殊的实例属性

实例仅有两个特殊属性。对于任意对象I：

I.\_\_class\_\_：实例化I的类

I.\_\_dict\_\_：I的属性

--------------------------------------------------

>>> class Person(object):

... def \_\_init\_\_(self):

... pass

...

>>> person = Person()

>>> person.\_\_class\_\_

<class '\_\_main\_\_.Person'>

>>> person.\_\_dict\_\_

{}

>>> person.foo = 1

>>> person.id = 123

>>> person.\_\_dict\_\_

{'foo': 1, 'id': 123}

------------------------------------------------

### 内建类型属性

可以使用dir()查看

如：复数

-------------------------------

>>> x=1+2j

>>> dir(x)

['\_\_abs\_\_', '\_\_add\_\_', '\_\_class\_\_', '\_\_coerce\_\_', '\_\_delattr\_\_', '\_\_div\_\_', '\_\_d

ivmod\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_float\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_

\_', '\_\_getattribute\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_i

nt\_\_', '\_\_le\_\_', '\_\_long\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_neg\_\_'

, '\_\_new\_\_', '\_\_nonzero\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rdiv\_\_', '\_\_rdi

vmod\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rmod\_\_',

'\_\_rmul\_\_', '\_\_rpow\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_'

, '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_', '\_\_truediv\_\_', 'conjugate', 'imag',

'real']

>>> x.\_\_class\_\_

<type 'complex'>

>>> x.imag

2.0

>>> x.real

1.0

----------------------------------

### 实例属性VS类属性

类属性仅是与类相关的数值，和实例无关，这些值像静态成员那样被引用，在多次实例化中调用类，它们的值保持不变。静态成员不会因为实例而改变他们的值，除非实例中显式改变它们的值。

类和实例都有各自的命名空间，类是类属性的命名空间，实例是实例属性的。

类属性可以通过类或实例来访问。

-----------------------------------

>>> class Person(object):

... version = 1.0

...

>>> c = Person()

>>> c.version #通过实例访问

1.0

>>> Person.version #通过类访问

1.0

>>> Person.version+=0.1 #只能通过类来更新

>>> Person.version #类访问

1.1

>>> c.version #实例访问

1.1

-------------------------------------

**从实例中访问类属性须谨慎**

任何对实例属性的赋值都会创建一个实例属性（若不存在的话）并对其赋值。如果类属性中存在同名的属性，则相互之间没有关联。

-----------------------------------------

>>> class Person(object):

... version = 1.0

...

>>> c = Person()

>>> c.version = 2

>>> c.version

2

>>> Person.version

1.0

------------------------------------------

### 继承

一个子类可以继承它的基类的任何属性，不管是数据属性还是方法。

### 通过继承覆盖方法

--------------------------------------------------

class P(object):

def foo(self):

print "P-foo()"

class C(P):

def foo(self):

# P.foo(self)

super(C,self).foo() #推荐

print "C-foo()"

c = C()

c.foo()

----------------------------------------------

重写\_\_init\_\_不会自动调用基类的\_\_init\_\_,如果在子类中覆盖了\_\_init\_\_()，子类被实例化时，基类的\_\_init\_\_()就不会被自动调用。

-----------------------------------------

class P(object):

def \_\_init\_\_(self):

print "P's constructor"

class C(P):

# pass

def \_\_init\_\_(self):

print "C's constructor"

c = C()

--------------------------------------------

可变类型的例子：

---------------------------------------

class SortedKeyDict(dict):

def keys(self):

return sorted(super(SortedKeyDict,self).keys())

d = SortedKeyDict((('a',1),('b',2),('c',3)))

print [key for key in d]

print d.keys()

输出：

['a', 'c', 'b']

['a', 'b', 'c']

----------------------------------------

### 多重继承

经典类，深度优先；

class A: pass

class B(A): pass

class C(A): pass

class D(B, C): pass

查找顺序：D→B→A→C→A

新式类，广度优先。

新式类中有个属性，可以知道查找顺序是怎样的：\_\_mro\_\_

class A(object): pass

class B(A): pass

class C(A): pass

class D(B, C): pass

查找顺序：D→B→C→A

print D.\_\_mro\_\_

输出：

(<class '\_\_main\_\_.D'>, <class '\_\_main\_\_.B'>, <class '\_\_main\_\_.C'>, <class '\_\_main\_\_.A'>, <type 'object'>)

--------------------------------------------