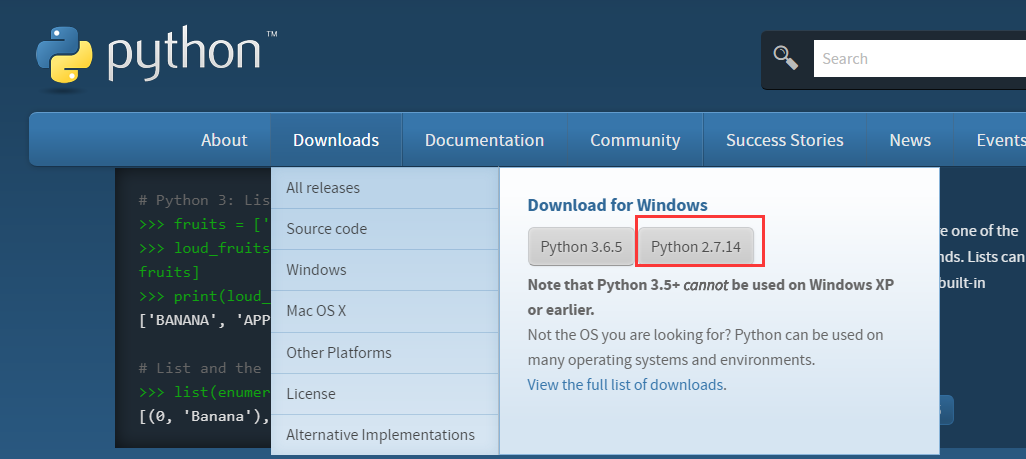
## 1 安装软件

### 1.1下载安装

官网下载 <https://www.python.org>



3.6版本和2.7版本语法不兼容，所以下载老 版本，这样很多插件就可以用了

下载完直接安装。

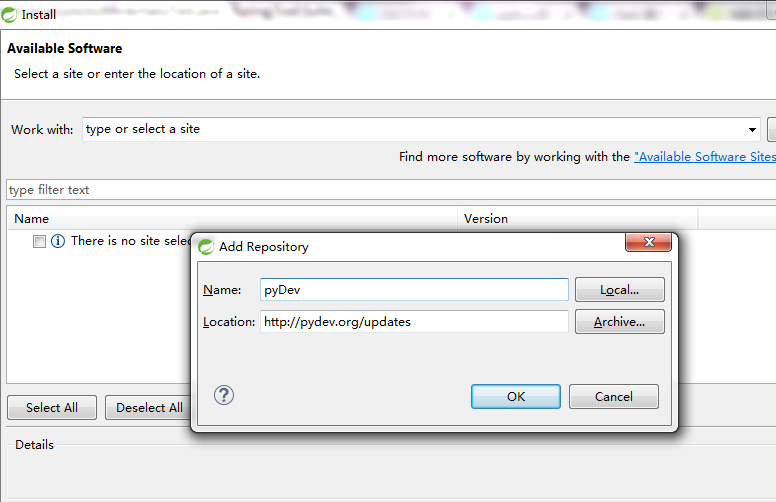
### 1.2 配置环境变量

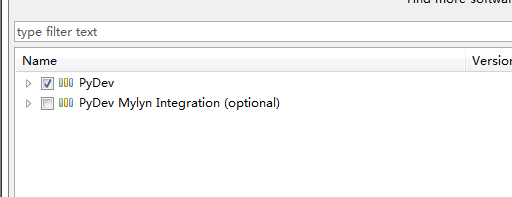
把安装目录配置到环境变量里的path里去。配置完成之后 验证：

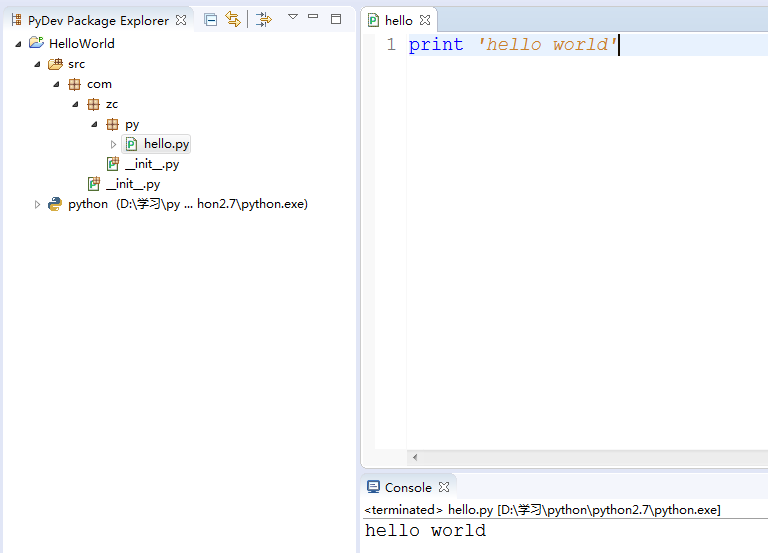


2 Eclipse配置python开发环境

http://pydev.org/updates







## 2 数据类型

### 2.1 简单数据类型

#### 2.1.1 整数

Python可以处理任意大小的整数，当然包括负整数，在Python程序中，整数的表示方法和数学上的写法一模一样，例如：1，100，-8080，0，等等。

计算机由于使用二进制，所以，有时候用十六进制表示整数比较方便，十六进制用0x前缀和0-9，a-f表示，例如：0xff00，0xa5b4c3d2，等等。

#### 2.1.2 浮点数

浮点数也就是小数，之所以称为浮点数，是因为按照科学记数法表示时，一个浮点数的小数点位置是可变的，比如，1.23x10^9和12.3x10^8是相等的。浮点数可以用数学写法，如1.23，3.14，-9.01，等等。但是对于很大或很小的浮点数，就必须用科学计数法表示，把10用e替代，1.23x10^9就是**1.23e9**，或者**12.3e8**，0.000012可以写成**1.2e-5**，等等。

整数和浮点数在计算机内部存储的方式是不同的，整数运算永远是精确的（除法难道也是精确的？是的！），而浮点数运算则可能会有四舍五入的误差。

#### 2.1.3 字符串

字符串是以''或""括起来的任意文本，比如**'abc'**，**"xyz"**等等。请注意，**''**或**""**本身只是一种表示方式，不是字符串的一部分，因此，字符串'abc'只有a，b，c这3个字符。

#### 2.1.4 布尔值

布尔值和布尔代数的表示完全一致，一个布尔值只有True、False两种值，要么是True，要么是False，在Python中，可以直接用True、False表示布尔值（请注意大小写），也可以通过布尔运算计算出来。

布尔值可以用and、or和not运算。

and运算是与运算，只有所有都为 True，and运算结果才是 True。

or运算是或运算，只要其中有一个为 True，or 运算结果就是 True。

not运算是非运算，它是一个单目运算符，把 True 变成 False，False 变成 True。

Python把0、空字符串''和None看成 False，其他数值和非空字符串都看成 True，所以：

True and 'a=T' 计算结果是 'a=T'

继续计算 'a=T' or 'a=F' 计算结果还是 'a=T'

**要解释上述结果，又涉及到 and 和 or 运算的一条重要法则：短路计算。**

1. 在计算 a and b 时，如果 a 是 False，则根据与运算法则，整个结果必定为 False，因此返回 a；如果 a 是 True，则整个计算结果必定取决与 b，因此返回 b。

2. 在计算 a or b 时，如果 a 是 True，则根据或运算法则，整个计算结果必定为 True，因此返回 a；如果 a 是 False，则整个计算结果必定取决于 b，因此返回 b。

所以Python解释器在做布尔运算时，只要能提前确定计算结果，它就不会往后算了，直接返回结果。

#### 2.1.5 空值

空值是Python里一个特殊的值，用None表示。None不能理解为0，因为0是有意义的，而None是一个特殊的空值

### 2.2 复杂数据类型

#### 2.2.1 list

Java里的数组

一种有序的集合，可以随时添加和删除其中的元素

由于Python是动态语言，所以list中包含的元素并不要求都必须是同一种数据类型，我们完全可以在list中包含各种数据：

>>> L = ['Michael', 100, True]

##### 2.2.1.1 获取元素

通过索引来获取list中的指定元素。索引从 0 开始

-1 这个索引来表示最后一个元素, 类似的，倒数第二用 -2 表示，倒数第三用 -3 表示

##### **2.2.1.2** 新增元素

**append()**总是把新的元素添加到 list 的尾部。

**L.insert(0, 'Paul')** 的意思是，'Paul'将被添加到索引为 0 的位置上（也就是第一个），而原来索引为 0 的Adam同学，以及后面的所有同学，都自动向后移动一位

##### 2.2.1.3 删除元素

**pop()**方法总是删掉list的最后一个元素，并且它还返回这个元素，所以我们执行 L.pop() 后，会打印出 'Paul'。

由于Paul的索引是2，因此，用 pop(2)把Paul删掉

##### 2.2.1.4 替换元素

L[2] = 'Paul'

#### 2.2.2 元组 tuple

* tuple一旦创建完毕，就不能修改了
* 创建tuple和创建list唯一不同之处是用( )替代了[ ]
* 正是因为用()定义**单元素**的tuple有歧义，所以 Python 规定，单元素 tuple 要多加一个逗号“,”，这样就避免了歧义, Python在打印单元素tuple时，也自动添加了一个“,”
* 可变

>>> t = ('a', 'b', ['A', 'B'])

**注意**到 t 有 3 个元素：**'a'，'b'**和一个list：**['A', 'B']**。list作为一个整体是tuple的第3个元素。list对象可以通过 t[2] 拿到：

>>> L = t[2

然后，我们把list的两个元素改一改：

>>> L[0] = 'X'

>>> L[1] = 'Y'

再看看tuple的内容：

>>> print t

('a', 'b', ['X', 'Y'])

#### 2.2.3 dict

Java 里的map

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

len(d) 获取长度

**key 的元素必须不可变**，Python的基本类型如字符串、整数、浮点数都是不可变的，都可以作为 key。但是list是可变的，就不能作为 key

##### 2.2.3.1 获取值

通过 key 访问 dict 的value，只要 key 存在，dict就返回对应的value。如果key不存在，会直接报错：KeyError

要避免 KeyError 发生，有两个办法：

**一是先判断一下 key 是否存在，用 in 操作符：**

if 'Paul' in d:

print d['Paul']

如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。

**二是使用dict本身提供的一个 get 方法，在Key不存在的时候，返回None：**

>>> print d.get('Bart')

59

>>> print d.get('Paul')

None

**遍历**

>>> d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

>>> for key in d:

... print key

##### 2.2.3.2 更新值

d['Paul'] = 72

直接赋值

#### 2.2.4 set

类似java 枚举

不能重复，无序

>>> s = set(['A', 'B', 'C', 'C'])

>>> print s

set(['A', 'C', 'B'])

>>> len(s)

3

输出语句Bart是该班的同学吗？

>>> 'Bart' in s

True

**set存储的元素和dict的key类似，必须是不变对象**

>>> s = set(['Adam', 'Lisa', 'Bart'])

>>> for name in s:

... print name

...

Lisa

Adam

Bart

##### 2.2.4.1 更新set

如果添加的元素已经存在于set中，add()不会报错，但是不会加进去了：

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

>>> s.add(3)

>>> print s

set([1, 2, 3])

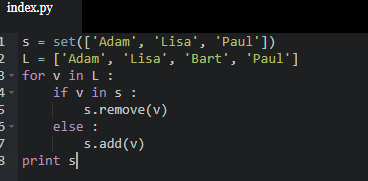
删除set中的元素时，用set的remove()方法：

>>> s = set([1, 2, 3, 4])

>>> s.remove(4)

>>> print s

set([1, 2, 3])



## 3. 输出语句

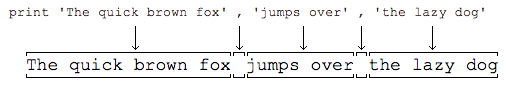
### 3.1 print

print语句也可以跟上多个字符串，用逗号“,”隔开，就可以连成一串输出：

>>> print 'The quick brown fox', 'jumps over', 'the lazy dog'

The quick brown fox jumps over the lazy dog

print会依次打印每个字符串，遇到逗号“,”会输出一个空格，因此，输出的字符串是这样拼起来的：

[](http://img.mukewang.com/54055502000179c205060086.jpg)

## 4 注释

### 4.1 单行注释 #

Python的注释以 # 开头，后面的文字直到行尾都算注释

# 这一行全部都是注释...

print 'hello' # 这也是注释

## 5 变量

变量是用一个变量名表示，变量名必须是大小写英文、数字和下划线（\_）的组合，且不能用数字开头

等号=是赋值语句，可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量

这种变量本身类型不固定的语言称之为动态语言

## 6 字符串

‘’ “” \ 转义。这些不用多说

如果一个字符串包含很多需要转义的字符，对每一个字符都进行转义会很麻烦。为了避免这种情况，我们可以在字符串前面加个前缀r ，表示这是一个 raw 字符串，里面的字符就不需要转义了。例如：

r'\(~\_~)/ \(~\_~)/'

但是r'...'表示法不能表示多行字符串，也不能表示包含'和 "的字符串（为什么？）

如果要表示多行字符串，可以用'''...'''表示：

'''Line 1

Line 2

Line 3'''

上面这个字符串的表示方法和下面的是完全一样的：

'Line 1\nLine 2\nLine 3'

还可以在多行字符串前面添加 r ，把这个多行字符串也变成一个raw字符串：

r'''Python is created by "Guido".

It is free and easy to learn.

Let's start learn Python in imooc!'''

## 7 unicode 字符串

如果中文字符串在Python环境下遇到 UnicodeDecodeError，这是因为.py文件保存的格式有问题。可以在第一行添加注释

# -\*- coding: utf-8 -\*-

目的是告诉Python解释器，用UTF-8编码读取源代码。然后用Notepad++ 另存为... 并选择UTF-8格式保存。

## 8 流程控制语句

### 8.1 if

age = 20

if age >= 18:

print 'your age is', age

print 'adult'

print 'END'

**注意:**Python代码的缩进规则。具有相同缩进的代码被视为代码块，上面的3，4行 print 语句就构成一个代码块（但不包括第5行的print）。如果 if 语句判断为 True，就会执行这个代码块。

缩进请严格按照Python的习惯写法：4个空格，不要使用Tab，更不要混合Tab和空格，否则很容易造成因为缩进引起的语法错误。

**注意**: if 语句后接表达式，然后用:表示代码块开始。

**if-else**

if age >= 18:

print 'adult'

else:

print 'teenager'

**if-elif-else**

if age >= 18:

print 'adult'

elif age >= 6:

print 'teenager'

elif age >= 3:

print 'kid'

else:

print 'baby'

**特别注意:**这一系列条件判断会从上到下依次判断，如果某个判断为 True，执行完对应的代码块，后面的条件判断就直接忽略，不再执行了。

### 8.2 for

L = ['Adam', 'Lisa', 'Bart']

for name in L:

print name

### 8.3 while

N = 10

x = 0

while x < N:

print x

x = x + 1

### 8.4 break continue

同java

## 9 函数

### 9.1 内置函数

内置函数文档地址：

http://docs.python.org/2/library/functions.html

#### 9.1.1 **abs()**函数

绝对值

>>> abs(-20)

20

#### 9.1.2 cmp(x, y)

比较函数 如果 **x<y**，返回**-1**，如果**x==y**，返回 **0**，如果 **x>y**，返回 **1**

#### 9.1.3 int()，str()

类型转换函数

#### 9.1.4 map()

是 Python 内置的高阶函数，它接收一个**函数 f** 和一个 **list**，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。

#### 9.1.5 reduce()

例如，编写一个f函数，接收x和y，返回x和y的和：

def f(x, y):

return x + y

调用 **reduce(f, [1, 3, 5, 7, 9])**时，reduce函数将做如下计算：

先计算头两个元素：f(1, 3)，结果为4；

再把结果和第3个元素计算：f(4, 5)，结果为9；

再把结果和第4个元素计算：f(9, 7)，结果为16；

再把结果和第5个元素计算：f(16, 9)，结果为25；

由于没有更多的元素了，计算结束，返回结果25。

上述计算实际上是对 list 的所有元素求和。虽然Python内置了求和函数sum()，但是，利用reduce()求和也很简单。

**reduce()还可以接收第3个可选参数，作为计算的初始值。**如果把初始值设为100，计算：

reduce(f, [1, 3, 5, 7, 9], 100)

结果将变为125，因为第一轮计算是：

计算初始值和第一个元素：**f(100, 1)**，结果为**101**。

#### 9.1.6 filter()

**filter()**函数是 Python 内置的另一个有用的高阶函数，filter()函数接收一个**函数 f**和一个**list**，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，**filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。**

例如，要从一个list [1, 4, 6, 7, 9, 12, 17]中删除偶数，保留奇数，首先，要编写一个判断奇数的函数：

def is\_odd(x):

return x % 2 == 1

然后，利用filter()过滤掉偶数：

filter(is\_odd, [1, 4, 6, 7, 9, 12, 17])

**结果：**[1, 7, 9, 17]

#### 9.1.7 **sorted()**

Python内置的 **sorted()**函数可对list进行排序：

>>>sorted([36, 5, 12, 9, 21])  
[5, 9, 12, 21, 36]

但 **sorted()**也是一个高阶函数，它可以接收一个比较函数来实现自定义排序，比较函数的定义是，传入两个待比较的元素 x, y，**如果 x 应该排在 y 的前面，返回 -1，如果 x 应该排在 y 的后面，返回 1。如果 x 和 y 相等，返回 0。**

因此，如果我们要实现倒序排序，只需要编写一个reversed\_cmp函数：

def reversed\_cmp(x, y):

if x > y:

return -1

if x < y:

return 1

return 0

这样，调用 sorted() 并传入 reversed\_cmp 就可以实现倒序排序：

>>> sorted([36, 5, 12, 9, 21], reversed\_cmp)

[36, 21, 12, 9, 5]

sorted()也可以对字符串进行排序，字符串默认按照ASCII大小来比较：

>>> sorted(['bob', 'about', 'Zoo', 'Credit'])

['Credit', 'Zoo', 'about', 'bob']

'Zoo'排在'about'之前是因为'Z'的ASCII码比'a'小。

### 9.2 自定义函数

我们以自定义一个求绝对值的 my\_abs 函数为例：

def my\_abs(x):

if x >= 0:

return x

else:

return -x

如果没有return语句，函数执行完毕后也会返回结果，只是结果为 None。

* 返回多值; 原来返回值是一个**tuple**！

return nx, ny

* 定义默认参数

由于函数的参数按从左到右的顺序匹配，所以**默认参数只能定义在必需参数的后面**

def power(x, n=2):

s = 1

while n > 0:

n = n - 1

s = s \* x

return

* 定义可变参数（参数个数不固定）

def fn(\*args):

print args

>>> fn('a', 'b')

('a', 'b')

### 9.3 高级函数

同js

### 9.4 匿名函数

匿名函数 lambda x: x \* x 实际上就是：

def f(x):

return x \* x

### 9.5 装饰器 @

类似java 的切面编程 aop

**带参数的装饰器**：

def log(prefix):

def log\_decorator(f):

def wrapper(\*args, \*\*kw):

print '[%s] %s()...' % (prefix, f.\_\_name\_\_)

return f(\*args, \*\*kw)

return wrapper

return log\_decorator

@log('DEBUG')

def test():

pass

print test()

由于decorator返回的新函数函数名已经不是'f2'，而是@log内部定义的'wrapper'。这对于那些依赖函数名的代码就会失效。decorator还改变了函数的\_\_doc\_\_等其它属性。如果要让调用者看不出一个函数经过了@decorator的“改造”，就需要把原函数的一些属性复制到新函数中

def log(f):

@functools.wraps(f)

def wrapper(x):

print 'call...'

return f(x)

return wrapper

### 9.6 偏函数

functools.partial就是帮助我们创建一个偏函数的，不需要我们自己定义int2()，可以直接使用下面的代码创建一个新的函数int2：

>>> import functools

>>> int2 = functools.partial(int, base=2)

>>> int2('1000000')

64

>>> int2('1010101')

85

## 10 切片

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

对应上面的问题，取前3个元素，用一行代码就可以完成切片：

>>> L[0:3]

['Adam', 'Lisa', 'Bart']

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

>>> L[:3]

['Adam', 'Lisa', 'Bart']

也可以从索引1开始，取出2个元素出来：

>>> L[1:3]

['Lisa', 'Bart']

切片操作还可以指定第三个参数：

>>> L[::2]

['Adam', 'Bart']

第三个参数表示每N个取一个，上面的 L[::2] 会每两个元素取出一个来，也就是隔一个取一个。

对于list，既然Python支持L[-1]取倒数第一个元素，那么它同样支持倒数切片，试试：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> L[-2:]

['Bart', 'Paul']

在很多编程语言中，针对字符串提供了很多各种截取函数，其实目的就是对字符串切片。Python没有针对字符串的截取函数，只需要切片一个操作就可以完成，非常简单。

>>> 'ABCDEFG'[:3]

'ABC'

>>> 'ABCDEFG'[-3:]

'EFG'

>>> 'ABCDEFG'[::2]

'ACEG'

## 11 迭代

### 11.1 迭代集合list

对于有序集合，元素确实是有索引的。有的时候，我们确实想在 for 循环中拿到索引，怎么办？

方法是使用 **enumerate() 函数**：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> for index, name in enumerate(L):

... print index, '-', name

...

0 - Adam

1 - Lisa

2 - Bart

3 - Paul

### 11.2 迭代dict

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

print d.values()

# [85, 95, 59]

for v in d.values():

print v

效果一样的

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

print d.itervalues()

# <dictionary-valueiterator object at 0x106adbb50>

for v in d.itervalues():

print v

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

... print key, ':', value

和 values() 有一个 itervalues() 类似， **items()**也有一个对应的 **iteritems()**，iteritems() 不把dict转换成list，而是在迭代过程中不断给出 tuple，所以， iteritems() 不占用额外的内存。

## 12 列表生成

要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]，我们可以用range(1, 11)：

但如果要生成[1x1, 2x2, 3x3, ..., 10x10]怎么做？方法一是循环：

>>> L = []

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

... L.append(x \* x)

...

>>> L

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

但是循环太繁琐，而列表生成式则可以用一行语句代替循环生成上面的list：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

**提示：**range(1, 100, 2) 可以生成list [1, 3, 5, 7, 9,...]

使用**for循环**的迭代不仅可以迭代普通的list，还可以迭代dict。

假设有如下的dict：

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

完全可以通过一个复杂的列表生成式把它变成一个 HTML 表格：

tds = ['<tr><td>%s</td><td>%s</td></tr>' % (name, score) for name, score in d.iteritems()]

print '<table>'

print '<tr><th>Name</th><th>Score</th><tr>'

print '\n'.join(tds)

print '</table>'

**条件过滤**

列表生成式的 **for 循环后面还可以加上 if 判断**。例如：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

如果我们只想要偶数的平方，不改动 range()的情况下，可以加上 if 来筛选：

>>> [x \* x for x in range(1, 11) if x % 2 == 0]

[4, 16, 36, 64, 100]

有了 if 条件，只有 if 判断为 True 的时候，才把循环的当前元素添加到列表中。

**多层表达式**

for循环可以嵌套，因此，在列表生成式中，也可以用多层 for 循环来生成列表。

对于字符串 'ABC' 和 '123'，可以使用两层循环，生成全排列：

>>> [m + n for m in 'ABC' for n in '123']

['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']

翻译成循环代码就像下面这样：

L = []

for m in 'ABC':

for n in '123':

L.append(m + n)

## 13 模块

一个py文件就是一个模块，文件名就是模块名

包就是文件夹

包和普通文件夹的区别：

包下面一定要有个 \_\_init\_\_.py文件，包的每一层都要有这个文件，即使它是空文件

import math 引入模块

from math import pow, sin, log

这样，可以直接引用 pow, sin, log 这3个函数，但math的其他函数没有导入进来：

>>> pow(2, 10)

1024.0

>>> sin(3.14)

0.0015926529164868282

利用ImportError错误，我们经常在Python中动态导入模块：

try:

from cStringIO import StringIO

except ImportError:

from StringIO import StringIO

### \_\_future\_\_

Python的新版本会引入新的功能，但是，实际上这些功能在上一个老版本中就已经存在了。要“试用”某一新的特性，就可以通过导入\_\_future\_\_模块的某些功能来实现。

例如，Python 2.7的整数除法运算结果仍是整数：

>>> 10 / 3

3

但是，Python 3.x已经改进了整数的除法运算，“**/**”除将得到浮点数，“**//**”除才仍是整数：

>>> 10 / 3

3.3333333333333335

>>> 10 // 3

3

要在Python 2.7中引入3.x的除法规则，导入**\_\_future\_\_**的**division**：

>>> from \_\_future\_\_ import division

>>> print 10 / 3

3.3333333333333335

### 安装第三方模块

pip install web.py

查看第三方模块网址：<https://pypi.python.org/>

## 14 面向对象

虽然可以通过**Person**类创建出**xiaoming、xiaohong**等实例，但是这些实例看上除了地址不同外，没有什么其他不同。在现实世界中，区分**xiaoming、xiaohong**要依靠他们各自的名字、性别、生日等属性。

如何让每个实例拥有各自不同的属性？由于Python是动态语言，对每一个实例，都可以直接给他们的属性赋值，例如，给**xiaoming**这个实例加上**name、gender**和**birth**属性：

xiaoming = Person()

xiaoming.name = 'Xiao Ming'

xiaoming.gender = 'Male'

xiaoming.birth = '1990-1-1'

给**xiaohong**加上的属性不一定要和**xiaoming**相同：

xiaohong = Person()

xiaohong.name = 'Xiao Hong'

xiaohong.school = 'No. 1 High School'

xiaohong.grade = 2

实例的属性可以像普通变量一样进行操作：

xiaohong.grade = xiaohong.grade + 1

### 14.1 构造函数

class Person(object):

def \_\_init\_\_(self, name, gender, birth):

self.name = name

self.gender = gender

self.birth = birth

### 14.2 访问控制

Python对属性权限的控制是通过属性名来实现的，如果一个属性由双下划线开头(\_\_)，该属性就无法被外部访问

class Person(object):

def \_\_init\_\_(self, name):

self.name = name

self.\_title = 'Mr'

self.\_\_job = 'Student'

p = Person('Bob')

print p.name

# => Bob

print p.\_title

# => Mr

print p.\_\_job

# => Error

Traceback (most recent call last):

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

AttributeError: 'Person' object has no attribute '\_\_job'

### 14.3 静态变量

class Person(object):

address = 'Earth'

def \_\_init\_\_(self, name):

self.name = name

修改静态变量 Person.address = 'China'

当实例属性和类属性重名时，实例属性优先级高

### 14.4 静态方法

要在class中定义类方法，需要这么写：@classmethod

class Person(object):

count = 0

@classmethod

def how\_many(cls):

return cls.count

def \_\_init\_\_(self, name):

self.name = name

Person.count = Person.count + 1

print Person.how\_many()

p1 = Person('Bob')

print Person.how\_many()

### 14.5 继承

如果已经定义了**Person**类，需要定义新的**Student**和**Teacher**类时，可以直接从Person类继承：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

定义**Student**类时，只需要把额外的属性加上，例如score：

class Student(Person):

def \_\_init\_\_(self, name, gender, score):

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

使用 **isinstance** 判断类型：

>>> isinstance(p, Person)

True # p是Person类型

>>> isinstance(p, Student)

False # p不是Student类型

>>> isinstance(p, Teacher)

False # p不是Teacher类型

14.6 多继承

class D(B, C):

def \_\_init\_\_(self, a):

super(D, self).\_\_init\_\_(a)

print 'init D...'