第一个例子：

Hello world：

#!/usr/bin/python

print "Hello, World!";

首先恭喜你，你已经成功掌握了第一个python程序！

## **Python 保留字符**

下面的列表显示了在Python中的保留字。这些保留字不能用作常数或变数，或任何其他标识符名称。

所有 Python 的关键字只包含小写字母。

|  |  |  |
| --- | --- | --- |
| and | exec | not |
| assert | finally | or |
| break | for | pass |
| class | from | print |
| continue | global | raise |
| def | if | return |
| del | import | try |
| elif | in | while |
| else | is | with |
| except | lambda | yield |

**行和缩进**

Python主要通过缩进来写模块。

这里要求：

参加群内的python代码规范文档和github上的wiki。

## **多行语句**

Python语句中一般以新行作为语句的结束符。

但是我们可以使用斜杠（ \）将一行的语句分为多行显示。

## **Python 引号**

Python 可以使用引号( **'** )、双引号( **"** )、三引号( **'''** 或 **"""** ) 来表示字符串，引号的开始与结束必须的相同类型的。

其中三引号可以由多行组成，编写多行文本的快捷语法，常用于文档字符串，在文件的特定地点，被当做注释。

## **Python注释**

python中单行注释采用 # 开头。

python 中多行注释使用三个单引号(''')或三个双引号(""")。

**多个变量赋值**

您也可以为多个对象指定多个变量。例如：

a, b, c = 1, 2, "john"

即a=1, b=2 , c=”john”

**Python 变量类型**

## **Python字符串**

字符串或串(String)是由数字、字母、下划线组成的一串字符。

一般记为 :

s="a1a2···an"(n>=0)

它是编程语言中表示文本的数据类型。

python的字串列表有2种取值顺序:

* 从左到右索引默认0开始的，最大范围是字符串长度少1
* 从右到左索引默认-1开始的，最大范围是字符串开头

## **Python列表**

List（列表） 是 Python 中使用最频繁的数据类型。

列表可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。

列表用 **[ ]** 标识，是 python 最通用的复合数据类型。

## **Python元组**

元组是另一个数据类型，类似于List（列表）。

元组用"()"标识。内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。

tuple = ( 'runoob', 786 , 2.23, 'john', 70.2 )

list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # 元组中是非法应用

list[2] = 1000 # 列表中是合法应用

## **Python 字典**

字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象集合，字典是无序的对象集合。

两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

**Python 运算符**

## **Python算术运算符**

以下假设变量： **a=10，b=20**：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| + | 加 - 两个对象相加 | a + b 输出结果 30 |
| - | 减 - 得到负数或是一个数减去另一个数 | a - b 输出结果 -10 |
| \* | 乘 - 两个数相乘或是返回一个被重复若干次的字符串 | a \* b 输出结果 200 |
| / | 除 - x除以y | b / a 输出结果 2 |
| % | 取模 - 返回除法的余数 | b % a 输出结果 0 |
| \*\* | 幂 - 返回x的y次幂 | a\*\*b 为10的20次方， 输出结果 100000000000000000000 |
| // | 取整除 - 返回商的整数部分（**向下取整**） | 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0 |

## **Python比较运算符**

以下假设变量a为10，变量b为20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| == | 等于 - 比较对象是否相等 | (a == b) 返回 False。 |
| != | 不等于 - 比较两个对象是否不相等 | (a != b) 返回 true. |
| <> | 不等于 - 比较两个对象是否不相等 | (a <> b) 返回 true。这个运算符类似 != 。 |
| > | 大于 - 返回x是否大于y | (a > b) 返回 False。 |
| < | 小于 - 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。 | (a < b) 返回 true。 |
| >= | 大于等于 - 返回x是否大于等于y。 | (a >= b) 返回 False。 |
| <= | 小于等于 - 返回x是否小于等于y。 | (a <= b) 返回 true。 |

## **Python赋值运算符**

以下假设变量a为10，变量b为20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| = | 简单的赋值运算符 | c = a + b 将 a + b 的运算结果赋值为 c |
| += | 加法赋值运算符 | c += a 等效于 c = c + a |
| -= | 减法赋值运算符 | c -= a 等效于 c = c - a |
| \*= | 乘法赋值运算符 | c \*= a 等效于 c = c \* a |
| /= | 除法赋值运算符 | c /= a 等效于 c = c / a |
| %= | 取模赋值运算符 | c %= a 等效于 c = c % a |
| \*\*= | 幂赋值运算符 | c \*\*= a 等效于 c = c \*\* a |
| //= | 取整除赋值运算符 | c //= a 等效于 c = c // a |

**Python 条件语句**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# 例：elif用法

num = 5

if num == 3: # 判断num的值

print 'boss'

elif num == 2:

print 'user'

elif num == 1:

print 'worker'

elif num < 0: # 值小于零时输出

print 'error'

else:

print 'roadman' # 条件均不成立时输出

python 复合布尔表达式计算采用短路规则，即如果通过前面的部分已经计算出整个表达式的值，则后面的部分不再计算。如下面的代码将正常执行不会报除零错误：

a=0

b=1if ( a > 0 ) and ( b / a > 2 ):

print "yes"else :

print "no"

而下面的代码就会报错：

a=0

b=1if ( a > 0 ) or ( b / a > 2 ):

print "yes"else :

print "no"

Python 没有 **switch/case** 语句，如果遇到很多中情况的时候，写很多的 **if/else** 不是很好维护，这时可以考虑用字典映射的方法替代：

#!/usr/bin/env python# -\*- coding: utf-8 -\*-

import osdef zero():

return "zero"

def one():

return "one"

def two():

return "two"

def num2Str(arg):

switcher={

0:zero,

1:one,

2:two,

3:lambda:"three"

}

func=switcher.get(arg,lambda:"nothing")

return func()

if \_\_name\_\_ == '\_\_main\_\_':

print num2Str(0)

八皇后问题 （循环递归法）

#\* queen problem with recurison

BOARD\_SIZE = 8

def under\_attack(col, queens):

left = right = col

for r, c in reversed(queens):

#左右有冲突的位置的列号

left, right = left - 1, right + 1

if c in (left, col, right):

return True

return False

def solve(n):

if n == 0:

return [[]]

smaller\_solutions = solve(n - 1)

return [solution+[(n,i+1)]

for i in xrange(BOARD\_SIZE)

for solution in smaller\_solutions

if not under\_attack(i+1, solution)]for answer in solve(BOARD\_SIZE):

print answer

## **循环使用 else 语句**

在 python 中，while … else 在循环条件为 false 时执行 else 语句块：

## 实例

#!/usr/bin/python

count = 0

while count < 5:

print count, " is less than 5"

count = count + 1

else:

print count, " is not less than 5"

以上实例输出结果为：

0 is less than 5

1 is less than 5

2 is less than 5

3 is less than 5

4 is less than 5

5 is not less than 5

摇筛子游戏

#!/usr/bin/env python3# -\*- coding: utf-8 -\*-

import randomimport sysimport time

result = []while True:

result.append(int(random.uniform(1,7)))

result.append(int(random.uniform(1,7)))

result.append(int(random.uniform(1,7)))

print result

count = 0

index = 2

pointStr = ""

while index >= 0:

currPoint = result[index]

count += currPoint

index -= 1

pointStr += " "

pointStr += str(currPoint)

if count <= 11:

sys.stdout.write(pointStr + " -> " + "小" + "\n")

time.sleep( 1 ) # 睡眠一秒

else:

sys.stdout.write(pointStr + " -> " + "大" + "\n")

time.sleep( 1 ) # 睡眠一秒

result = []

Pyhton 去除字符串首尾的空格：

def trim(s):

while s[:1] == ' ':

s = s[1:]

while s[-1:] == ' ':

s = s[:-1]

return s

str = ' Runoob 'print(trim(str))

使用循环嵌套来获取100以内的质数

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

num=[];

i=2for i in range(2,100):

j=2

for j in range(2,i):

if(i%j==0):

break

else:

num.append(i)print(num)

range()函数

>>> range(1,5) # 代表从1到5(不包含5)

[1, 2, 3, 4]

>>> range(1,5,2) # 代表从1到5，间隔2(不包含5)

[1, 3]

>>> range(5) # 代表从0到5(不包含5)

[0, 1, 2, 3, 4]

注意：默认情况下，range() 的起始值是 0。

>>> for i in range(5) :

... print(i)

...

0

1

2

3

4

格式化操作符辅助指令:

|  |  |
| --- | --- |
| **符号** | **功能** |
| \* | 定义宽度或者小数点精度 |
| - | 用做左对齐 |
| + | 在正数前面显示加号( + ) |
| <sp> | 在正数前面显示空格 |
| # | 在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X') |
| 0 | 显示的数字前面填充'0'而不是默认的空格 |
| % | '%%'输出一个单一的'%' |
| (var) | 映射变量(字典参数) |
| m.n. | m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话) |

Python2.6 开始，新增了一种格式化字符串的函数 [str.format()](http://www.runoob.com/python/att-string-format.html" \t "_blank)，它增强了字符串格式化的功能。

## **return 语句**

return语句[表达式]退出函数，选择性地向调用方返回一个表达式。不带参数值的return语句返回None。之前的例子都没有示范如何返回数值，下例便告诉你怎么做：

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- # 可写函数说明 def sum( arg1, arg2 ): # 返回2个参数的和." total = arg1 + arg2 print "函数内 : ", total return total; # 调用sum函数 total = sum( 10, 20 );

以上实例输出结果：

函数内 : 30

## **匿名函数**

python 使用 lambda 来创建匿名函数。

* lambda只是一个表达式，函数体比def简单很多。
* lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
* lambda函数拥有自己的命名空间，且不能访问自有参数列表之外或全局命名空间里的参数。
* 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

### **语法**

lambda函数的语法只包含一个语句，如下：

lambda [arg1 [,arg2,.....argn]]:expression

如下实例：

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- # 可写函数说明 sum = lambda arg1, arg2: arg1 + arg2; # 调用sum函数 print "相加后的值为 : ", sum( 10, 20 ) print "相加后的值为 : ", sum( 20, 20 )

以上实例输出结果：

相加后的值为 : 30相加后的值为 : 40

### **缺省参数**

调用函数时，缺省参数的值如果没有传入，则被认为是默认值。下例会打印默认的age，如果age没有被传入：

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- #可写函数说明 def printinfo( name, age = 35 ): "打印任何传入的字符串" print "Name: ", name; print "Age ", age; return; #调用printinfo函数 printinfo( age=50, name="miki" ); printinfo( name="miki" );

以上实例输出结果：

Name: mikiAge 50Name: mikiAge 35

### **不定长参数**

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述2种参数不同，声明时不会命名。基本语法如下：

def functionname([formal\_args,] \*var\_args\_tuple ): "函数\_文档字符串" function\_suite return [expression]

加了星号（\*）的变量名会存放所有未命名的变量参数。不定长参数实例如下：

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- # 可写函数说明 def printinfo( arg1, \*vartuple ): "打印任何传入的参数" print "输出: " print arg1 for var in vartuple: print var return; # 调用printinfo 函数 printinfo( 10 ); printinfo( 70, 60, 50 );

以上实例输出结果：

输出:10输出:706050

## **命名空间和作用域**

变量是拥有匹配对象的名字（标识符）。命名空间是一个包含了变量名称们（键）和它们各自相应的对象们（值）的字典。

一个 Python 表达式可以访问局部命名空间和全局命名空间里的变量。如果一个局部变量和一个全局变量重名，则局部变量会覆盖全局变量。

每个函数都有自己的命名空间。类的方法的作用域规则和通常函数的一样。

Python 会智能地猜测一个变量是局部的还是全局的，它假设任何在函数内赋值的变量都是局部的。

因此，如果要给函数内的全局变量赋值，必须使用 global 语句。

global VarName 的表达式会告诉 Python， VarName 是一个全局变量，这样 Python 就不会在局部命名空间里寻找这个变量了。

例如，我们在全局命名空间里定义一个变量 Money。我们再在函数内给变量 Money 赋值，然后 Python 会假定 Money 是一个局部变量。然而，我们并没有在访问前声明一个局部变量 Money，结果就是会出现一个 UnboundLocalError 的错误。取消 global 语句的注释就能解决这个问题。

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

Money = 2000def AddMoney():

# 想改正代码就取消以下注释:

# global Money

Money = Money + 1

print MoneyAddMoney()print Money

## **dir()函数**

dir() 函数一个排好序的字符串列表，内容是一个模块里定义过的名字。

返回的列表容纳了在一个模块里定义的所有模块，变量和函数。如下一个简单的实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 导入内置math模块import math

content = dir(math)

print content;

以上实例输出结果：

['\_\_doc\_\_', '\_\_file\_\_', '\_\_name\_\_', 'acos', 'asin', 'atan', 'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log','log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']

在这里，特殊字符串变量\_\_name\_\_指向模块的名字，\_\_file\_\_指向该模块的导入文件名。

## **test.py**

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

# 导入 Phone 包

from package\_runoob.runoob1 import runoob1

from package\_runoob.runoob2 import runoob2

runoob1()

runoob2()

以上实例输出结果：

package\_runoob 初始化

I'm in runoob1

I'm in runoob2

**Python 文件I/O**

### **raw\_input函数**

raw\_input([prompt]) 函数从标准输入读取一个行，并返回一个字符串（去掉结尾的换行符）：

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

str = raw\_input("请输入：")

print "你输入的内容是: ", str

## **打开和关闭文件**

现在，您已经可以向标准输入和输出进行读写。现在，来看看怎么读写实际的数据文件。

Python 提供了必要的函数和方法进行默认情况下的文件基本操作。你可以用 **file** 对象做大部分的文件操作。

### **open 函数**

你必须先用Python内置的open()函数打开一个文件，创建一个file对象，相关的方法才可以调用它进行读写。

语法：

file object = open(file\_name [, access\_mode][, buffering])

各个参数的细节如下：

* file\_name：file\_name变量是一个包含了你要访问的文件名称的字符串值。
* access\_mode：access\_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。
* buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。

不同模式打开文件的完全列表：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **模式** | **描述** | | | | | | | |
| r | 以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。 | | | | | | | |
| rb | 以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。一般用于非文本文件如图片等。 | | | | | | | |
| r+ | 打开一个文件用于读写。文件指针将会放在文件的开头。 | | | | | | | |
| rb+ | 以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。一般用于非文本文件如图片等。 | | | | | | | |
| w | 打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 | | | | | | | |
| wb | 以二进制格式打开一个文件只用于写入。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。 | | | | | | | |
| w+ | 打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。 | | | | | | | |
| wb+ | 以二进制格式打开一个文件用于读写。如果该文件已存在则打开文件，并从开头开始编辑，即原有内容会被删除。如果该文件不存在，创建新文件。一般用于非文本文件如图片等。 | | | | | | | |
| a | 打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 | | | | | | | |
| ab | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。 | | | | | | | |
| a+ | 打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。 | | | | | | | |
| ab+ | 以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。 | | | | | | | |
| **模式** | | **r** | **r+** | **w** | **w+** | **a** | **a+** |
| 读 | | + | + |  | + |  | + |
| 写 | |  | + | + | + | + | + |
| 创建 | |  |  | + | + | + | + |
| 覆盖 | |  |  | + | + |  |  |
| 指针在开始 | | + | + | + | + |  |  |
| 指针在结尾 | |  |  |  |  | + | + |

# **Python File(文件) 方法**

file 对象使用 open 函数来创建，下表列出了 file 对象常用的函数：

|  |  |
| --- | --- |
| **序号** | **方法及描述** |
| 1 | [file.close()](http://www.runoob.com/python/file-close.html)  关闭文件。关闭后文件不能再进行读写操作。 |
| 2 | [file.flush()](http://www.runoob.com/python/file-flush.html)  刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。 |
| 3 | [file.fileno()](http://www.runoob.com/python/file-fileno.html)  返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。 |
| 4 | [file.isatty()](http://www.runoob.com/python/file-isatty.html)  如果文件连接到一个终端设备返回 True，否则返回 False。 |
| 5 | [file.next()](http://www.runoob.com/python/file-next.html)  返回文件下一行。 |
| 6 | [file.read([size])](http://www.runoob.com/python/python-file-read.html)  从文件读取指定的字节数，如果未给定或为负则读取所有。 |
| 7 | [file.readline([size])](http://www.runoob.com/python/file-readline.html)  读取整行，包括 "\n" 字符。 |
| 8 | [file.readlines([sizehint])](http://www.runoob.com/python/file-readlines.html)  读取所有行并返回列表，若给定sizeint>0，则是设置一次读多少字节，这是为了减轻读取压力。 |
| 9 | [file.seek(offset[, whence])](http://www.runoob.com/python/file-seek.html)  设置文件当前位置 |
| 10 | [file.tell()](http://www.runoob.com/python/file-tell.html)  返回文件当前位置。 |
| 11 | [file.truncate([size])](http://www.runoob.com/python/file-truncate.html)  截取文件，截取的字节通过size指定，默认为当前文件位置。 |
| 12 | [file.write(str)](http://www.runoob.com/python/python-file-write.html)  将字符串写入文件，返回的是写入的字符长度。 |
| 13 | [file.writelines(sequence)](http://www.runoob.com/python/file-writelines.html)  向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。 |

**Python 面向对象**

## **面向对象技术简介**

* **类(Class):** 用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。

class ClassName:

'类的帮助信息' #类文档字符串

class\_suite #类体

* **类变量：**类变量在整个实例化的对象中是公用的。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
* **数据成员：**类变量或者实例变量, 用于处理类及其实例对象的相关的数据。
* **方法重写：**如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。
* **实例变量：**定义在方法中的变量，只作用于当前实例的类。
* **继承：**即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。例如，有这样一个设计：一个Dog类型的对象派生自Animal类，这是模拟"是一个（is-a）"关系（例图，Dog是一个Animal）。
* **实例化：**创建一个类的实例，类的具体对象。
* **方法：**类中定义的函数。
* **对象：**通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

## 实例

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

class Employee:

'所有员工的基类'

empCount = 0

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

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

"创建 Employee 类的第一个对象"

emp1 = Employee("Zara", 2000)

"创建 Employee 类的第二个对象"

emp2 = Employee("Manni", 5000)

emp1.displayEmployee()

emp2.displayEmployee()

print "Total Employee %d" % Employee.empCount

执行以上代码输出结果如下：

Name : Zara ,Salary: 2000Name : Manni ,Salary: 5000Total Employee 2

你可以添加，删除，修改类的属性，如下所示：

emp1.age = 7 # 添加一个 'age' 属性

emp1.age = 8 # 修改 'age' 属性del emp1.age # 删除 'age' 属性

## **Python内置类属性**

* \_\_dict\_\_ : 类的属性（包含一个字典，由类的数据属性组成）
* \_\_doc\_\_ :类的文档字符串
* \_\_name\_\_: 类名
* \_\_module\_\_: 类定义所在的模块（类的全名是'\_\_main\_\_.className'，如果类位于一个导入模块mymod中，那么className.\_\_module\_\_ 等于 mymod）
* \_\_bases\_\_ : 类的所有父类构成元素（包含了一个由所有父类组成的元组）

Python内置类属性调用实例如下：

## 实例

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

class Employee:

'所有员工的基类'

empCount = 0

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

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

print "Employee.\_\_doc\_\_:", Employee.\_\_doc\_\_

print "Employee.\_\_name\_\_:", Employee.\_\_name\_\_

print "Employee.\_\_module\_\_:", Employee.\_\_module\_\_

print "Employee.\_\_bases\_\_:", Employee.\_\_bases\_\_ print "Employee.\_\_dict\_\_:", Employee.\_\_dict\_\_

## 实例

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

class Parent: # 定义父类

parentAttr = 100

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

print "调用父类构造函数"

def parentMethod(self):

print '调用父类方法'

def setAttr(self, attr):

Parent.parentAttr = attr

def getAttr(self): print "父类属性 :", Parent.parentAttr class Child(Parent): # 定义子类 def \_\_init\_\_(self): print "调用子类构造方法" def childMethod(self): print '调用子类方法' c = Child() # 实例化子类 c.childMethod() # 调用子类的方法 c.parentMethod() # 调用父类方法 c.setAttr(200) # 再次调用父类的方法 - 设置属性值 c.getAttr() # 再次调用父类的方法 - 获取属性值

以上代码执行结果如下：

调用子类构造方法调用子类方法调用父类方法父类属性 : 200

Python 通过改变名称来包含类名:

122Traceback (most recent call last):

File "test.py", line 17, in <module>

print counter.\_\_secretCount # 报错，实例不能访问私有变量AttributeError: JustCounter instance has no attribute '\_\_secretCount'

Python不允许实例化的类访问私有数据，但你可以使用 **object.\_className\_\_attrName**（ **对象名.\_类名\_\_私有属性名** ）访问属性，参考以下实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

class Runoob:

\_\_site = "www.runoob.com"

runoob = Runoob()print runoob.\_Runoob\_\_site

执行以上代码，执行结果如下：

www.runoob.com

### **单下划线、双下划线、头尾双下划线说明：**

**\_\_foo\_\_**: 定义的是特殊方法，一般是系统定义名字 ，类似 **\_\_init\_\_()** 之类的。

**\_foo**: 以单下划线开头的表示的是 protected 类型的变量，即保护类型只能允许其本身与子类进行访问，不能用于 **from module import \***

**\_\_foo**: 双下划线的表示的是私有类型(private)的变量, 只能是允许这个类本身进行访问了。

# **Python 多线程**

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用thread模块中的start\_new\_thread()函数来产生新线程。语法如下:

thread.start\_new\_thread ( function, args[, kwargs] )

参数说明:

* function - 线程函数。
* args - 传递给线程函数的参数,他必须是个tuple类型。
* kwargs - 可选参数。

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- import thread import time # 为线程定义一个函数 def print\_time( threadName, delay): count = 0 while count < 5: time.sleep(delay) count += 1 print "%s: %s" % ( threadName, time.ctime(time.time()) ) # 创建两个线程 try: thread.start\_new\_thread( print\_time, ("Thread-1", 2, ) ) thread.start\_new\_thread( print\_time, ("Thread-2", 4, ) ) except: print "Error: unable to start thread" while 1: pass

## **使用Threading模块创建线程**

使用Threading模块创建线程，直接从threading.Thread继承，然后重写\_\_init\_\_方法和run方法：

## 实例(Python 2.0+)

#!/usr/bin/python # -\*- coding: UTF-8 -\*- import threading import time exitFlag = 0 class myThread (threading.Thread): #继承父类threading.Thread def \_\_init\_\_(self, threadID, name, counter): threading.Thread.\_\_init\_\_(self) self.threadID = threadID self.name = name self.counter = counter def run(self): #把要执行的代码写到run函数里面 线程在创建后会直接运行run函数 print "Starting " + self.name print\_time(self.name, self.counter, 5) print "Exiting " + self.name def print\_time(threadName, delay, counter): while counter: if exitFlag: (threading.Thread).exit() time.sleep(delay) print "%s: %s" % (threadName, time.ctime(time.time())) counter -= 1 # 创建新线程 thread1 = myThread(1, "Thread-1", 1) thread2 = myThread(2, "Thread-2", 2) # 开启线程 thread1.start() thread2.start() print "Exiting Main Thread"