# Python语言的特点和应用领域

## Python语言的特点

Python（英国发音：/ˈpaɪθən/）, 是一种面向对象的解释型计算机程序设计语言，由荷兰人Guido van Rossum（吉多·范罗苏姆）于1989年发明，第一个公开发行版发行于1991年，已经有了28年的历史。

Python是纯粹的免费、开源软件。

Python语法简洁清晰，简单易学。

Python具有丰富和强大的库。

面向对象语言

胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起。

很多人工智能相关库使用Python写的，因此Python是人工智能首选语言。

## Python语言应用领域

目前使用Python语言的企业有很多

1. 著名的Google公司在其网络搜索系统中广泛应用Python语言。同时还聘用了Python之父 （Guido van Rossum）

2.国外知名的YouTube视频分享网站，一些重要的服务也几乎都是用Python编写的程序；

3.P2P文件分享系统Bittorrent是一个Python 程序。

1. Intel 、Cisco(思科) 、Hewlett-Packard(惠普) 、Seagate(希捷) 、Qualcomm(高通) 和IBM 也都使用Python 进行硬件测试
2. JPMorgan Chase(摩根大通集团) 、UBS(瑞士联合银行集团) 、Getco 和Citadel 使用Python，经济市场预测领域也能看到python的身影。
3. 高科技含量的领域也有Python语言的身影，像是NASA(美国国家航空航天局) 、Los Alamos(洛斯阿拉莫斯洛杉矶国家实验室) 、Fermilab(费米实验室) 、JPL(喷气推进实验室) 等使用Python 实现科学计算任务；
4. IRobot公司使用Python 开发了商业机器人真空吸尘器；
5. NSA(美国国家安全局) 在加密和智能分析中使用Python 。
6. IronPort 业界领先的互联网信息安全产品提供商，也在电子邮件服务器产品中使用了超过100 万行的Python 代码实现其工作；
7. Python在用户图形接口领域也很受欢迎

Python语言的简洁和快速的开发周期，让它十分适合开发GUI程序。玩蛇网Python内置的TKinter 的标准面向对象接口Tk GUI API ，使Python 程序可以生成可移植的本地观感的GUI

python语言对传统的数据库需求提供了对所有主流关系数据库系统的接口：像是Sybase(美国Sybase公司研制的一种关系型[数据库系统](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/_blank)) 、ODBC(开放数据库互连) 、MySQL 、PostgreSQL 、SQLite 、Oracle 、Informix ... Python 在金融领域的主要应用是什么?Python定义了一种通过Python 脚本存取，SQL 数据库系统的可移植的数据库API ，这个API 对于各种底层应用的数据库系统都是统一的。

Python编程语言应用领域实在是很广泛，以上为主要应用比较多的几大类，像是游戏、图像、人工智能、XML 、机器人等等还有很多。python在未来一定是会在计算机语言中占有很重要的地位，这也是我们关注喜欢使用python的原因之一。

## 从1997年到2017年从排名第27到全球第4



## Python相对于其他语言实现相同的功能，代码更少更简单



# Python语言的优势

## 1\_面向对象

从根本上讲，**Python 是一种面向对象的语言。它的类模块支持多态、操作符重载和多重继承等高级概念，并且以Python 特有的简洁的语法和类型，OOP 十分易于使用**。事实上，即使你不懂这些术语，仍会发现学习Python 比学习其他OOP 语言要容易得多。

除了作为一种强大的代码构建和重用手段以外，Python 的OOP 特性使它成为面向对象系统语言如C++ 和Java 的理想脚本工具。例如，通过适当的粘接代码，Python 程序可以对C++ 、Java 和C# 的类进行子类的定制。

**OOP 是Python 的一个选择而已，这一点非常重要。不必强迫自己立马成为一个面向对象高手，你同样可以继续深入学习。就像C++ 一样，Python 既支持面向对象编程也支持面向过程编程的模式**。如果条件允许的话，其面向对象的工具即刻生效。这对处于预先设计阶段的策略开发模式十分有用。

## 2\_免费

Python 的使用和分发是完全免费的。就像其他的开源软件一样，例如，Tcl 、Perl 、Linux 和Apache 。**你可以从Internet 上免费获得Python 系统的源代码。复制Python ，将其嵌入你的系统或者随产品一起发布都没有任何限制。实际上，如果你愿意的话，甚至可以销售它的源代码。**

但请别误会："免费"并不代表" 无支持"。恰恰相反，Python 的在线社区对用户需求的响应和商业软件一样快。而且，由于Python 完全开放源代码，提高了开发者的实力，并产生了一个很大的专家团队。尽管学习研究或改变一个程序语言的实现并不是对每一个人来说都那么有趣，但是当你知道还有源代码作为最终的帮助和无尽的文档资源是多么的令人欣慰。你不需要去依赖商业厂商。

Python 的开发是由社区驱动的，是Internet 大范围的协同合作努力的结果。这个团体包括Python 的创始者Guido van Rossum：Python 社区内公认的"终身的慈善独裁者"[Benevolent Dictator for Life (BDFL) ] 。Python 语言的改变必须遵循一套规范的有约束力的程序（称作PEP 流程），并需要经过规范的测试系统和BDFL 进行彻底检查。值得庆幸的是，正是这样使得Python 相对于其他语言可以保守地持续改进。

## 3\_可移植

**Python 的标准实现是由可移植的ANSI C 编写的，可以在目前所有的主流平台上编译和运行**。**例如，如今从PDA 到超级计算机，到处可以见到Python 在运行。Python 可以在下列平台上运行**（这里只是部分列表）：

Linux 和UNIX 系统。

微软Windows 和DOS （所有版本）。

Mac OS （包括OS X 和Classic ）。

BeOS 、OS/2 、VMS 和QNX 。

实时操作系统，例如，VxWorks 。

Cray 超级计算机和IBM 大型机。

运行Palm OS 、PocketPC 和Linux 的PDA 。

运行Windows Mobile 和Symbian OS 的移动电话。

游戏终端和iPod 。

还有更多。

除了语言解释器本身以外，Python 发行时自带的标准库和模块在实现上也都尽可能地考虑到了跨平台的移植性。此外，Python 程序自动编译成可移植的字节码，这些字节码在已安装兼容版本Python 的平台上运行的结果都是相同的。

这些意味着Python 程序的核心语言和标准库可以在Linux 、Windows 和其他带有Python 解释器的平台无差别的运行。大多数Python 外围接口都有平台相关的扩展（例如，COM 支持Windows ），但是核心语言和库在任何平台都一样。就像之前我们提到的那样，Python 还包含了一个叫做Tkinter 的Tk GUI 工具包，它可以使Python 程序实现功能完整的无需做任何修改即可在所有主流GUI 平台运行的用户图形界面。

## 4\_功能强大

从特性的观点来看，Python 是一个混合体。它丰富的工具集使它介于传统的脚本语言（例如，Tcl 、Scheme 和Perl ）和系统语言(例如，C、C++ 和Java) 之间。**Python 提供了所有脚本语言的简单和易用性，并且具有在编译语言中才能找到的高级软件工程工具。不像其他脚本语言，这种结合使Python 在长期大型的开发项目中十分有用**。下面是一些Python 工具箱中的工具简介。

**动态类型Python 在运行过程中随时跟踪对象的种类，不需要代码中关于复杂的类型和大小的声明**。事实上，Python 中没有类型或变量声明这回事。因为Python 代码不是约束数据的类型，它往往自动地应用了一种广义上的对象。

**自动内存管理Python 自动进行对象分配，当对象不再使用时将自动撤销对象（"垃圾回收"），当需要时自动扩展或收缩。Python 能够代替你进行底层的内存管理。**

**大型程序支持为了能够建立更大规模的系统，Python 包含了模块、类和异常等工具**。这些工具允许你组织系统为组件，使用OOP 重用并定制代码，并以一种优雅的方式处理事件和错误。

**内置对象类型Python 提供了常用的数据结构作为语言的基本组成部分。例如，列表（list ）、字典（dictionary ）、字符串（string ）**。我们将会看到，它们灵活并易于使用。例如，内置对象可以根据需求扩展或收缩，可以任意地组织复杂的信息等。

内置工具为了对以上对象类型进行处理，Python 自带了许多强大的标准操作，包括合并（concatenation ）、分片（slice ）、排序（sort ）和映射（mapping ）等。

**库工具为了完成更多特定的任务，Python 预置了许多预编码的库工具，从正则表达式匹配到网络都支持。Python 的库工具在很多应用级的操作中发挥作用**。

第三方工具由于 Python 是开放源代码的，它鼓励开发者提供Python 内置工具之外的预编码工具。从网络上，可以找到COM 、图像处理、CORBA ORB 、XML 、数据库等很多免费的支持工具。

除了这一系列的Python 工具外，**Python 保持了相当简洁的语法和设计。综合这一切得到的就是一个具有脚本语言所有可用性的强大编程工具**。

## 5\_可混合

**Python 程序可以以多种方式轻易地与其他语言编写的组件"粘接"在一起**。例如，Python 的C语言API 可以帮助Python 程序灵活地调用C程序。这意味着可以根据需要给Python 程序添加功能，或者在其他环境系统中使用Python 。例如，将Python 与C或者C++ 写成的库文件混合起来，使Python 成为一个前端语言和定制工具。就像之前我们所提到过的那样，这使Python 成为一个很好的快速原型工具；首先出于开发速度的考虑，系统可以先使用Python 实现，之后转移至C，根据不同时期性能的需要逐步实现系统。

## 6\_使用简单

**运行Python 程序，只需要简单地键入Python 程序并运行就可以了。不需要其他语言（例如，C或C++ ）所必须的编译和链接等中间步骤。Python 可立即执行程序，这形成了一种交互式编程体验和不同情况下快速调整的能力，往往在修改代码后能立即看到程序改变后的效果。**

**当然，开发周期短仅仅是Python 易用性的一方面的体现。Python 提供了简洁的语法和强大的内置工具**。实际上，Python 曾有种说法叫做"可执行的伪代码"。**由于它减少了其他工具常见的复杂性，当实现相同的功能时，用Python 程序比采用C、C++ 和Java 编写的程序更为简单、小巧，也更灵活。**

## 7\_简单易学

这一部分引出了本书的重点：**相对于其他编程语言，Python 语言的核心是惊人的简单易学。实际上，你可以在几天内（如果你是有经验的程序员，或许只需要几个小时）写出不错的Python 代码**。这对于那些想学习语言可以在工作中应用的专业人员来说是一个好消息，同样对于那些使用Python 进行定制或控制系统的终端用户来说也是一个好消息。如今，许多系统依赖于终端用户可以很快地学会Python 以便定制其代码的外围工具，从而提供较少的支持甚至不提供支持。尽管Python 还是有很多高级编程工具，但不论对初学者还是行家高手来说，Python 的核心语言仍是相当简单的。

名字来源于 Monty Python

Python 名字的来源这不算是一项技术，但是，这似乎是令人很惊讶、保护得很好的秘密，而我们希望把它全盘托出。尽管Python 世界中都是蟒蛇的图标，但事实是，Python 创立者Guido van Rossum 是以BBC 喜剧Monty Python's Flying Circus 来命名的。他是Monty Python 的大影迷，而很多软件开发人员也是（事实上，这两个领域似乎有种对称）。

这给 Python 代码的例子加入一种幽默的特质。比如，一般来说，传统常规的变量名为"foo "和"bar "，在Python 的世界中变成了"spam "和"eggs "。而有时出现的"Brian "、"ni "、"shrubbery "等也是这样来的。这种方式甚至很大程度上影响了Python 社区：Python 会议上的演讲往往叫做"The Spanish Inquisition "。

当然，如果你熟悉这个幽默剧的话，所有这些你都会觉得很有趣，否则就没那么有意思了。你没有必要为了理解引自 Monty Python （也许本书中你就会找到）的例子而刻意去熟悉这一串剧情，但是至少你现在应该知道它们的来源。

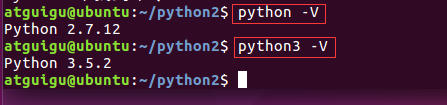
## 8\_现成为人工智能首选语言

很多人工智能相关库使用Python写的，因此Python是人工智能首选语言。

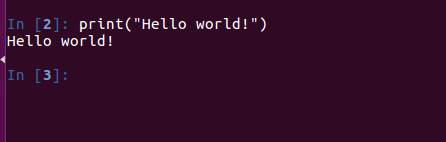
# Python语言结构

## 1\_搭建Python开发环境及入门

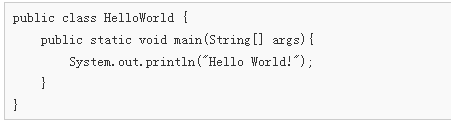
### 查看Python使用版本



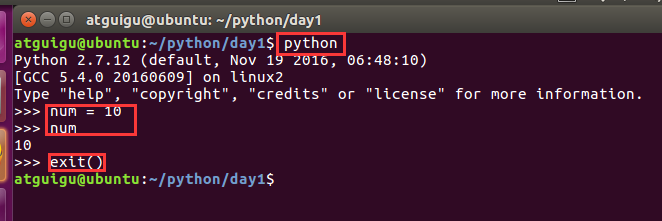
### Python在屏幕上打印“Hello world!”



### Java打印“Hello world!”



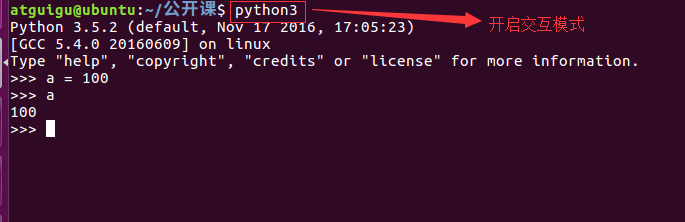
### Python2的交换模式



#### 开启python3交互模式

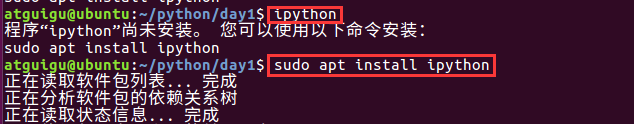
进入命令：python3

退出命令：exit()或者ctr+d

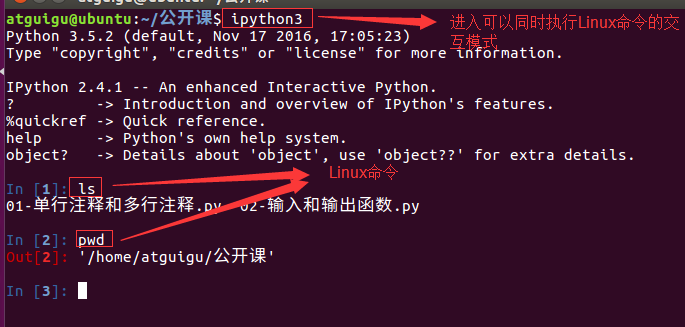


#### 开启可以执行Linux命令的交互模式

先安装



进入命令：ipython3



## 2\_标识符

以后变量和函数的命名

### 标识符组成

**标示符由字母、下划线和数字组成，且数字不能开头**

**python中的标识符是区分大小写的**

思考：下面的标示符哪些是正确的，哪些不正确为什么

fromNo12

from#12

my\_Boolean

my-Boolean

Obj2

2ndObj

myInt

test1

Mike2jack

My\_tExt

\_test

test!32

haha(da)tt

int

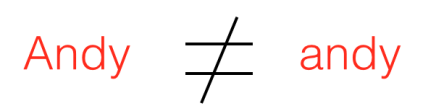
jack\_rose

jack&rose

GUI

G.U.I

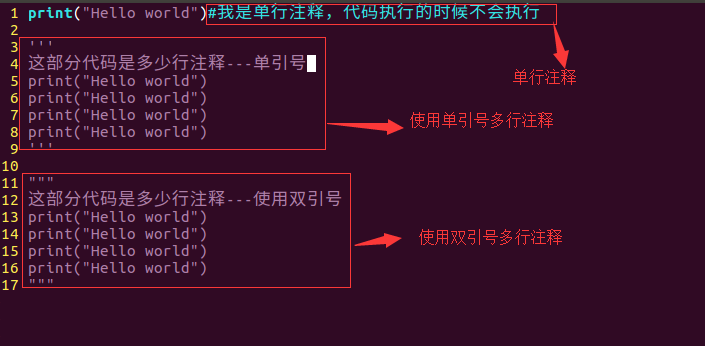
### python中的标识符是区分大小写的



## 3\_注释

作用一：注释可以起到一个**备注的作用**，团队合作的时候，个人编写的代码经常会被多人调用，**为了让别人能更容易理解代码的通途，使用注释是非常有效的**。

作用二：把部分代码不让其执行



## 4\_行与缩进-Pyhton代码风格(和其他语言的区别)

python最具特色的就是使用缩进来表示代码块，不需要使用大括号({})。

缩进的空格数是可变的，但是同一**个代码块**的语句必须包含**相同的缩进空格数**。实例如下：

if True:

print ("True")else:

else :

print ("False")

以下代码最后一行语句缩进数的空格数不一致，会导致运行错误：

if True:

print ("Answer")

print ("True")else:

else:

print ("Answer")

print ("False") # 缩进不一致，会导致运行错误

以上程序由于缩进不一致，执行后会出现类似以下错误：

File "test.py", line 6

print ("False") # 缩进不一致，会导致运行错误

^IndentationError: unindent does not match any outer indentation level

## 5\_字符串

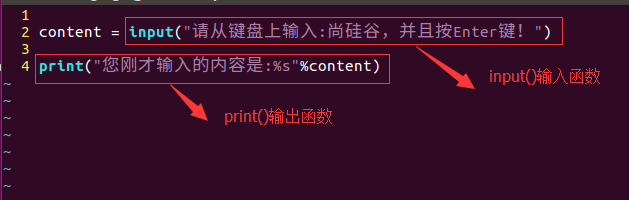
* python中单引号和双引号使用完全相同。
* 使用三引号('''或""")可以指定一个多行字符串。
* 转义符 '\'
* 字符串是不可变的。
* 按字面意义级联字符串，如"this " "is " "string"会被自动转换为this is string。

例如 name = “atguigu” 就是字符串，可以根据下标取值



## 6\_用户输入和输出

执行下面的程序在按回车键后就会等待用户输入：



运行结果：



## 7\_import 与 from...import

### import xxx

在 python 用 import 或者 from...import 来导入相应的模块。

将整个模块(random)导入，格式为： import random

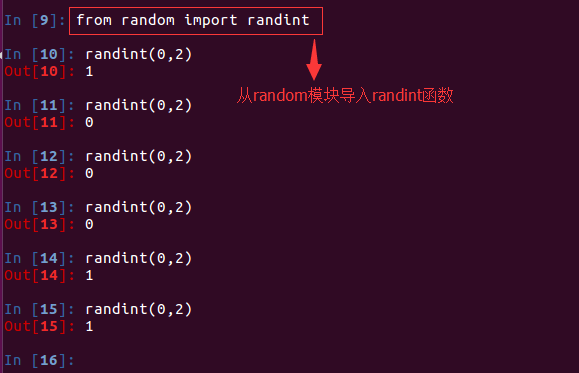


### from xxx import xxx的方法

从某个模块中导入某个函数,格式为： from random import randint

从某个模块中导入多个函数,格式为： from random import randint,shuffle

将某个模块中的全部函数导入，格式为： from random import \*



## 8\_Python3 中有六个标准的数据类型

Number（数字）

String（字符串）

List（列表）

Tuple（元组）

Sets（集合）

Dictionary（字典）

### Number（数字）

Python3 支持 **int、float、bool、complex（复数）**。

在Python 3里，只有一种整数类型 int，表示为长整型，没有 python2 中的 Long。

像大多数语言一样，数值类型的赋值和计算都是很直观的。

内置的 type() 函数可以用来查询变量所指的对象类型。

>>> a, b, c, d = 20, 5.5, True, 4+3j

>>> print(type(a), type(b), type(c), type(d))

<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>

**注意：**在 Python2 中是没有布尔型的，它用数字 0 表示 False，用 1 表示 True。到 Python3 中，把 True 和 False 定义成关键字了，但它们的值还是 1 和 0，它们可以和数字相加。

### String（字符串）

Python中的字符串用单引号(')或双引号(")括起来，同时使用反斜杠(\)转义特殊字符。

字符串的截取的语法格式如下：

变量[头下标:尾下标]

索引值以 0 为开始值，-1 为从末尾的开始位置。

加号 (+) 是字符串的连接符， 星号 (\*) 表示复制当前字符串，紧跟的数字为复制的次数。实例如下：

str = 'atguigu'

print (str) # 输出字符串

print (str[0:-1]) # 输出第一个到倒数第二个的所有字符

print (str[0]) # 输出字符串第一个字符

print (str[2:5]) # 输出从第三个开始到第五个的字符

print (str[2:]) # 输出从第三个开始的后的所有字符

print (str \* 2) # 输出字符串两次

print (str + "TEST") # 连接字符串

### List（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。列表中元素的类型可以不相同，它支持数字，字符串甚至可以包含列表（所谓嵌套）。

列表是写在方括号([])之间、用逗号分隔开的元素列表。

和字符串一样，列表同样可以被索引和截取，列表被截取后返回一个包含所需元素的新列表。

列表截取的语法格式如下：

变量[头下标:尾下标]

list = [ 'abcd', 786 , 2.23, 'atguigu', 70.2 ]

tinylist = [123, 'atguigu']

print (list) # 输出完整列表

print (list[0]) # 输出列表第一个元素

print (list[1:3]) # 从第二个开始输出到第三个元素

print (list[2:]) # 输出从第三个元素开始的所有元素

print (tinylist \* 2) # 输出两次列表

print (list + tinylist) # 连接列表

注意：

1、List写在方括号之间，元素用逗号隔开。

2、和字符串一样，list可以被索引和切片。

3、List可以使用+操作符进行拼接。

4、List中的元素是可以改变的。

### Tuple（元组）

元组（tuple）与列表类似，不同之处在于元组的元素不能修改。元组写在小括号(**()**)里，元素之间用逗号隔开。

元组中的元素类型也可以不相同：

tuple = ( 'abcd', 786 , 2.23, 'atguigu', 70.2 )

tinytuple = (123, 'atguigu')

print (tuple) # 输出完整元组

print (tuple[0]) # 输出元组的第一个元素

print (tuple[1:3]) # 输出从第二个元素开始到第三个元素

print (tuple[2:]) # 输出从第三个元素开始的所有元素

print (tinytuple \* 2) # 输出两次元组

print (tuple + tinytuple) # 连接元组

元组与字符串类似，可以被索引且下标索引从0开始，-1 为从末尾开始的位置。也可以进行截取（看上面，这里不再赘述）。

其实，可以把字符串看作一种特殊的元组。

### Dictionary（字典）

字典（dictionary）是Python中另一个非常有用的内置数据类型。

列表是有序的对象结合，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。

字典是一种映射类型，字典用"{ }"标识，它是一个无序的**键(key) : 值(value)**对集合。

键(key)必须使用不可变类型。

在同一个字典中，键(key)必须是唯一的。



### Set（集合）

集合（set）是一个无序不重复元素的序列。

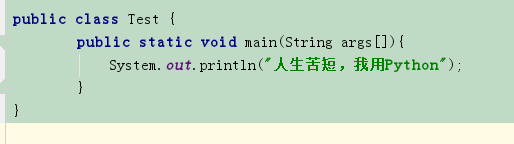
基本功能是进行成员关系测试和删除重复元素。

可以使用大括号 { } 或者 set() 函数创建集合，注意：创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。

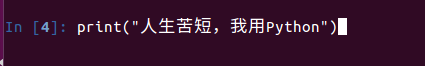
# Python对比其他语言优势

## 打印”人生苦短，我用Python”

### Java代码

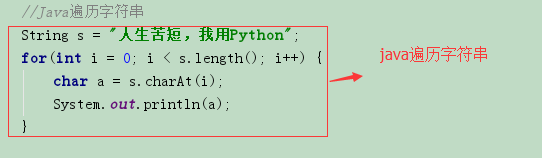


### Python代码

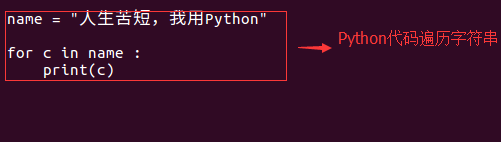


## Python和Java遍历字符串

### Java代码



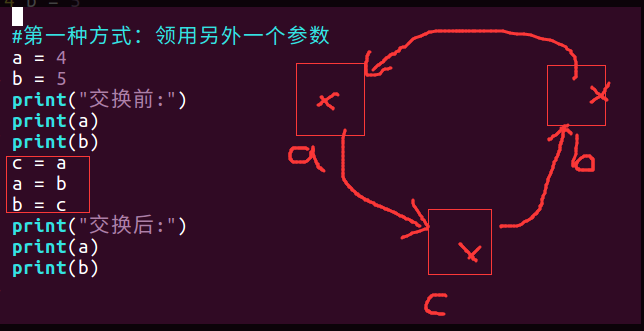
### Python代码



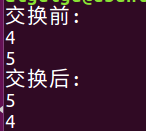
## 交换两个数

### 第一种方式（java和Python都支持）：使用第三个参数

其他语言也是用

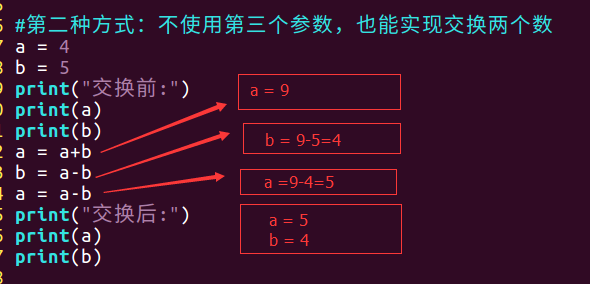


打印结果：

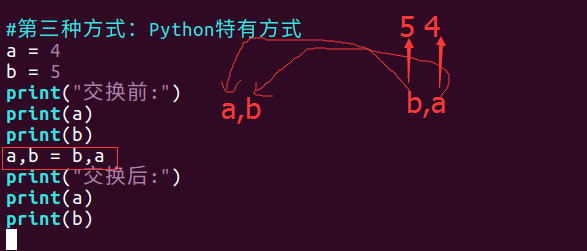


### 第二种方式（java和Python都支持）：使用第三个参数

就使用这两个参数，实现交换两个数



### 第三种方式：Python特有方式

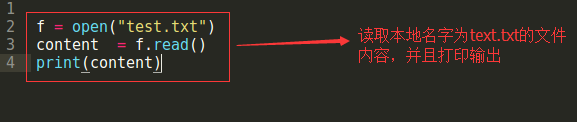


## 读取一个txt文件

### Java代码18行



### Python代码工具3行



# Python3 运算符

Python语言支持以下类型的运算符:

* [算术运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf1)
* [比较（关系）运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf2)
* [赋值运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf3)
* [逻辑运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf4)
* [位运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf5)
* [成员运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf6)
* [身份运算符](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf7)
* [运算符优先级](http://www.runoob.com/python3/python3-basic-operators.html" \l "ysf8)

接下来让我们一个个来学习Python的运算符。

## Python算术运算符

以下假设变量a为10，变量b为21：

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| + | 加 - 两个对象相加 | a + b 输出结果 31 |
| - | 减 - 得到负数或是一个数减去另一个数 | a - b 输出结果 -11 |
| \* | 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 210 |
| / | 除 - x 除以 y | b / a 输出结果 2.1 |
| % | 取模 - 返回除法的余数 | b % a 输出结果 1 |
| **\*\*** | **幂 - 返回x的y次幂** | a\*\*b 为10的21次方 |
| // | 取整除 - 返回商的整数部分 | 9//2 输出结果 4 , 9.0//2.0 输出结果 4.0 |

以下实例演示了Python所有算术运算符的操作：

## 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。 |
| > | 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。注意，这些变量名的大写。 | (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位运算符

按位运算符是把数字看作二进制来进行计算的。Python中的按位运算法则如下：

下表中变量 a 为 60，b 为 13二进制格式如下：

a = 0011 1100

b = 0000 1101

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

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| & | 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0 | (a & b) 输出结果 12 ，二进制解释： 0000 1100 |
| | | 按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。 | (a | b) 输出结果 61 ，二进制解释： 0011 1101 |
| ^ | 按位异或运算符：当两对应的二进位相异时，结果为1 | (a ^ b) 输出结果 49 ，二进制解释： 0011 0001 |
| ~ | 按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1。~x 类似于 -x-1 | (~a ) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。 |
| << | 左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。 | a << 2 输出结果 240 ，二进制解释： 1111 0000 |
| >> | 右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数 | a >> 2 输出结果 15 ，二进制解释： 0000 1111 |

## Python逻辑运算符

Python语言支持逻辑运算符，以下假设变量 a 为 10, b为 20:

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **逻辑表达式** | **描述** | **实例** |
| and | x and y | 布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。 | (a and b) 返回 20。 |
| or | x or y | 布尔"或" - 如果 x 是 True，它返回 x 的值，否则它返回 y 的计算值。 | (a or b) 返回 10。 |
| not | not x | 布尔"非" - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。 | not(a and b) 返回 False |

## Python身份运算符

身份运算符用于比较两个对象的存储单元

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| is | is 是判断两个标识符是不是引用自一个对象 | **x is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False |
| is not | is not 是判断两个标识符是不是引用自不同对象 | **x is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。 |

**注：** [id()](http://www.runoob.com/python/python-func-id.html" \t "http://www.runoob.com/python3/_blank) 函数用于获取对象内存地址。

## Python运算符优先级

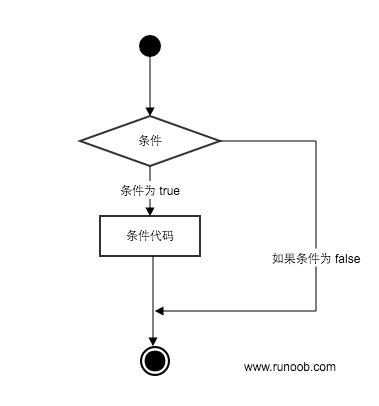
以下表格列出了从最高到最低优先级的所有运算符：

|  |  |
| --- | --- |
| **运算符** | **描述** |
| \*\* | 指数 (最高优先级) |
| ~ + - | 按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@) |
| \* / % // | 乘，除，取模和取整除 |
| + - | 加法减法 |
| >> << | 右移，左移运算符 |
| & | 位 'AND' |
| ^ | | 位运算符 |
| <= < > >= | 比较运算符 |
| <> == != | 等于运算符 |
| = %= /= //= -= += \*= \*\*= | 赋值运算符 |
| is is not | 身份运算符 |
| in not in | 成员运算符 |
| not or and | 逻辑运算符 |

# Python3 条件控制

Python条件语句是通过一条或多条语句的执行结果（True或者False）来决定执行的代码块。

可以通过下图来简单了解条件语句的执行过程:



## if 语句

Python中if语句的一般形式如下所示：

if condition\_1:

statement\_block\_1

elif condition\_2:

statement\_block\_2

else:

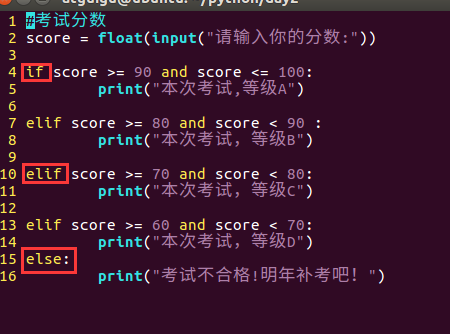
statement\_block\_3

* 如果 "condition\_1" 为 True 将执行 "statement\_block\_1" 块语句
* 如果 "condition\_1" 为False，将判断 "condition\_2"
* 如果"condition\_2" 为 True 将执行 "statement\_block\_2" 块语句
* 如果 "condition\_2" 为False，将执行"statement\_block\_3"块语句

Python 中用 **elif** 代替了 **else if**，所以if语句的关键字为：**if – elif – else**。

**注意：**

* 1、每个条件后面要使用冒号（:），表示接下来是满足条件后要执行的语句块。
* 2、使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。
* 3、在Python中没有switch – case语句。



## if 嵌套

在嵌套 if 语句中，可以把 if...elif...else 结构放在另外一个 if...elif...else 结构中。

if 表达式1:

语句

if 表达式2:

语句

elif 表达式3:

语句

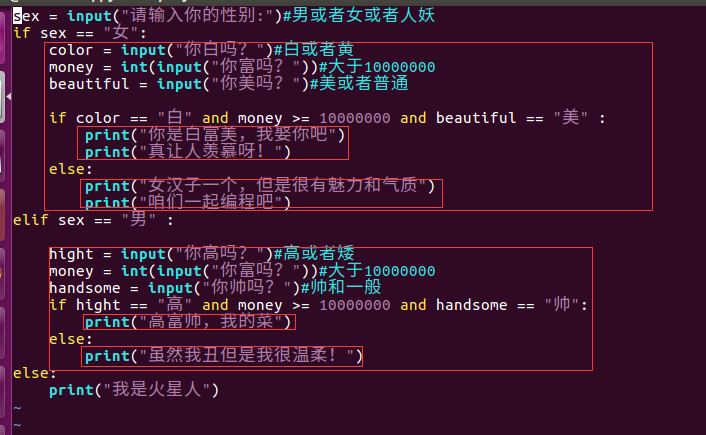
else:

语句elif 表达式4:

语句else:

语句

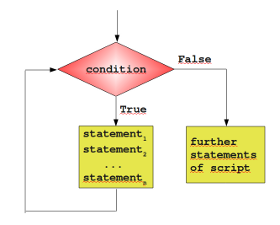
实例



# Python3 循环语句

Python中的循环语句有 for 和 while。

Python循环语句的控制结构图如下所示：



## while 循环

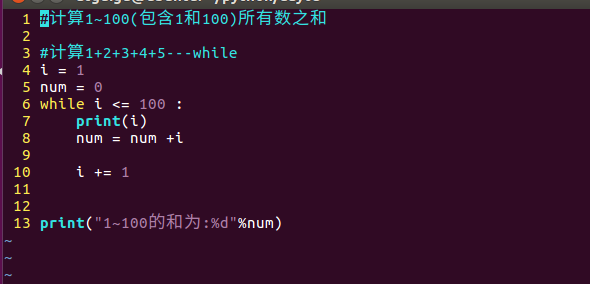
Python中while语句的一般形式：

while 判断条件：

语句

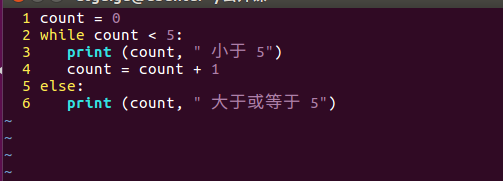
同样需要注意冒号和缩进。另外，在Python中没有do..while循环。

以下实例使用了 while 来计算 1 到 100 的总和：



## while 循环使用 else 语句

在 while … else 在条件语句为 false 时执行 else 的语句块：



## for 语句

Python for循环可以遍历任何序列的项目，如一个列表或者一个字符串。

for循环的一般格式如下：

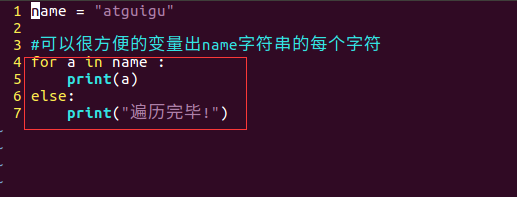
for循环的格式

for 临时变量 in 列表或者字符串等:

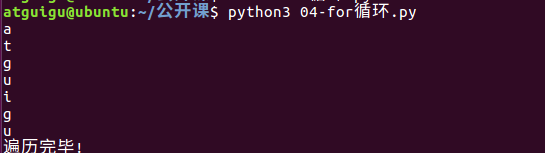
循环满足条件时执行的代码

else:

循环不满足条件时执行的代码



执行结果：



## break和continue语句及循环中的else子句

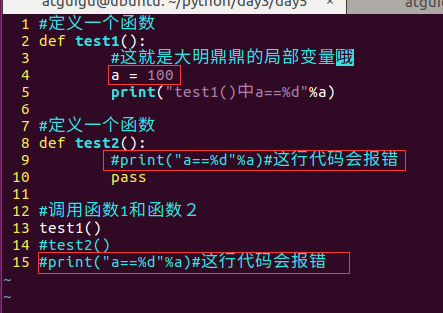
break 语句可以跳出 for 和 while 的循环体。如果你从 for 或 while 循环中终止，任何对应的循环 else 块将不执行。 实例如下：

continue语句被用来告诉Python跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

# 局部变量和全局变量

## <1>什么是局部变量

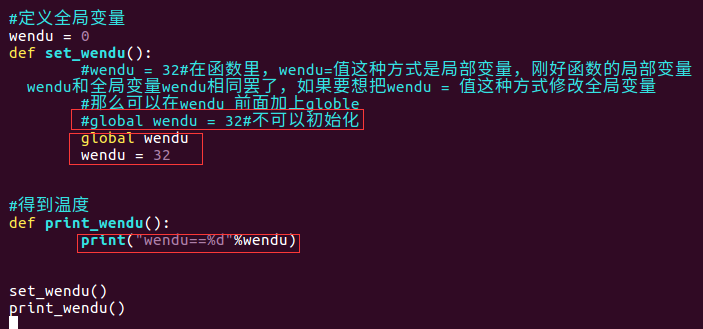
如下图所示:



## <3>在函数值添加global后修改全局变量

既然全局变量，就是能够在所以的函数中进行使用，那么可否进行修改呢？

### 代码如下:



# Python3 函数

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。但你也可以自己创建函数，这被叫做用户自定义函数。

## 定义一个函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

* 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号 ()。
* 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
* 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
* 函数内容以冒号起始，并且缩进。
* return [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

### **语法**

Python 定义函数使用 def 关键字，一般格式如下：

def 函数名（参数列表）:

函数体

默认情况下，参数值和参数名称是按函数声明中定义的的顺序匹配起来的。

### **实例1**

让我们使用函数来输出"Hello World！"：

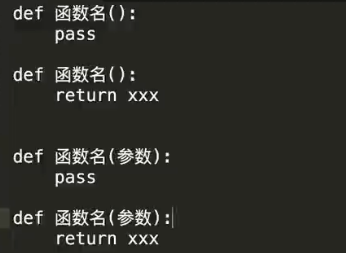
>>> def hello() :

print("Hello World!")

>>> hello()

Hello World!

>>>



### 实例2



### 执行多次的结果

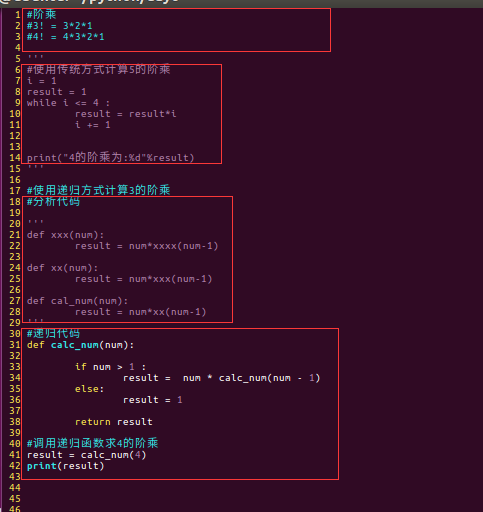


## 递归函数

### 什么是递归函数

通过前面的学习知道一个函数可以调用其他函数。

**如果一个函数在内部不调用其它的函数，而是自己本身的话，这个函数就是递归函数**。



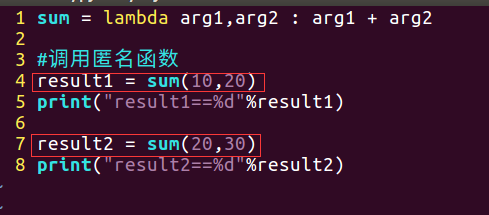
## 匿名函数

### 匿名函数定义

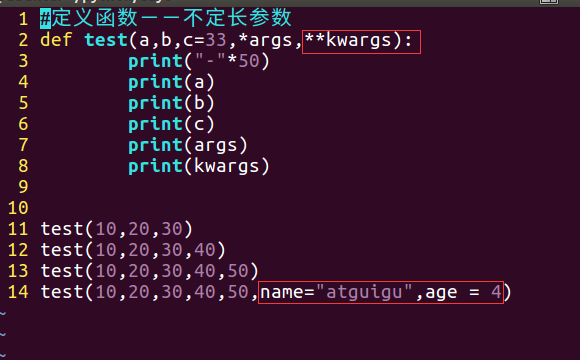
**用lambda关键词能创建小型匿名函数。这种函数得名于省略了用def声明函数的标准步骤**。

lambda函数的语法只包含一个语句，如下：

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



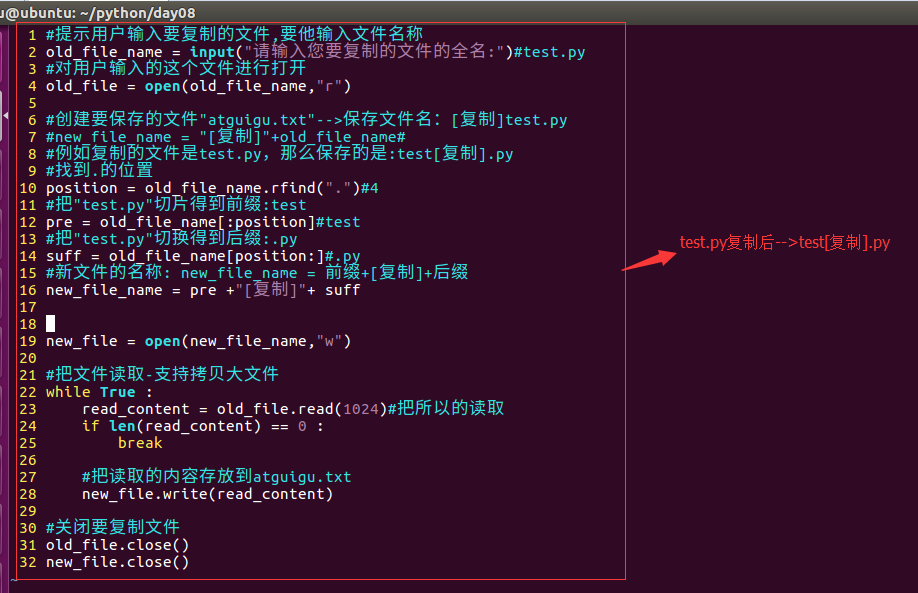
## 缺省参数和不定长参数



# Python3 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/python3/python3-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/python3/python3-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/python3/python3-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/python3/python3-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/python3/python3-file-next.html)  返回文件下一行。 |
| 6 | [file.read([size])](http://www.runoob.com/python3/python3-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/python3/python3-file-readline.html)  读取整行，包括 "\n" 字符。 |
| 8 | [file.readlines([sizeint])](http://www.runoob.com/python3/python3-file-readlines.html)  读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比 sizeint 较大, 因为需要填充缓冲区。 |
| 9 | [file.seek(offset[, whence])](http://www.runoob.com/python3/python3-file-seek.html)  设置文件当前位置 |
| 10 | [file.tell()](http://www.runoob.com/python3/python3-file-tell.html)  返回文件当前位置。 |
| 11 | [file.truncate([size])](http://www.runoob.com/python3/python3-file-truncate.html)  从文件的首行首字符开始截断，截断文件为 size 个字符，无 size 表示从当前位置截断；截断之后 V 后面的所有字符被删除，其中 Widnows 系统下的换行代表2个字符大小。 |
| 12 | [file.write(str)](http://www.runoob.com/python3/python3-file-write.html)  将字符串写入文件，没有返回值。 |
| 13 | [file.writelines(sequence)](http://www.runoob.com/python3/python3-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。 |



# **文件夹的相关操作**

实际开发中，有时需要用程序的方式对文件夹进行一定的操作，比如创建、删除等

就像对文件操作需要os模块一样，如果要操作文件夹，同样需要os模块

## **<1>创建文件夹**

不能创建./张三/test/hehe

import os

os.mkdir("张三")

## **<2>创建多层目录**

import os

os.makedirs("张三/李四/王二")

## **<3>获取当前目录**

import os

os.**getcwd**()

## **<4>改变默认目录**

进入到某个目录

**import os**

**os.chdir("../")**

## **<5>获取目录列表**

import os

os.listdir("./")

## **<6>删除文件夹**

删除目录 path，要求path必须是个空目录，否则抛出OSError错误

import os

os.rmdir("张三")

### 删除多层目录

import shutil  
shutil.rmtree(“张三”)

## <7>练习

用程序创建一个a目录，进入a目录，再创建b目录；进入b目录后，创建一文件”日记.txt”,

用程序写入内容:”今天我在硅谷学习”

# Python3 OS 文件/目录方法

**os** 模块提供了非常丰富的方法用来处理文件和目录。常用的方法如下表所示：

|  |  |
| --- | --- |
| **序号** | **方法及描述** |
| 1 | [os.access(path, mode)](http://www.runoob.com/python3/python3-os-access.html)  检验权限模式 |
| 2 | [os.chdir(path)](http://www.runoob.com/python3/python3-os-chdir.html)  改变当前工作目录 |
| 3 | [os.chflags(path, flags)](http://www.runoob.com/python3/python3-os-chflags.html)  设置路径的标记为数字标记。 |
| 4 | [os.chmod(path, mode)](http://www.runoob.com/python3/python3-os-chmod.html)  更改权限 |
| 5 | [os.chown(path, uid, gid)](http://www.runoob.com/python3/python3-os-chown.html)  更改文件所有者 |
| 6 | [os.chroot(path)](http://www.runoob.com/python3/python3-os-chroot.html)  改变当前进程的根目录 |
| 7 | [os.close(fd)](http://www.runoob.com/python3/python3-os-close.html)  关闭文件描述符 fd |
| 8 | [os.closerange(fd\_low, fd\_high)](http://www.runoob.com/python3/python3-os-closerange.html)  关闭所有文件描述符，从 fd\_low (包含) 到 fd\_high (不包含), 错误会忽略 |
| 9 | [os.dup(fd)](http://www.runoob.com/python3/python3-os-dup.html)  复制文件描述符 fd |
| 10 | [os.dup2(fd, fd2)](http://www.runoob.com/python3/python3-os-dup2.html)  将一个文件描述符 fd 复制到另一个 fd2 |
| 11 | [os.fchdir(fd)](http://www.runoob.com/python3/python3-os-fchdir.html)  通过文件描述符改变当前工作目录 |
| 12 | [os.fchmod(fd, mode)](http://www.runoob.com/python3/python3-os-fchmod.html)  改变一个文件的访问权限，该文件由参数fd指定，参数mode是Unix下的文件访问权限。 |
| 13 | [os.fchown(fd, uid, gid)](http://www.runoob.com/python3/python3-os-fchown.html)  修改一个文件的所有权，这个函数修改一个文件的用户ID和用户组ID，该文件由文件描述符fd指定。 |
| 14 | [os.fdatasync(fd)](http://www.runoob.com/python3/python3-os-fdatasync.html)  强制将文件写入磁盘，该文件由文件描述符fd指定，但是不强制更新文件的状态信息。 |
| 15 | [os.fdopen(fd[, mode[, bufsize]])](http://www.runoob.com/python3/python3-os-fdopen.html)  通过文件描述符 fd 创建一个文件对象，并返回这个文件对象 |
| 16 | [os.fpathconf(fd, name)](http://www.runoob.com/python3/python3-os-fpathconf.html)  返回一个打开的文件的系统配置信息。name为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多标准中指定（POSIX.1, Unix 95, Unix 98, 和其它）。 |
| 17 | [os.fstat(fd)](http://www.runoob.com/python3/python3-os-fstat.html)  返回文件描述符fd的状态，像stat()。 |
| 18 | [os.fstatvfs(fd)](http://www.runoob.com/python3/python3-os-fstatvfs.html)  返回包含文件描述符fd的文件的文件系统的信息，像 statvfs() |
| 19 | [os.fsync(fd)](http://www.runoob.com/python3/python3-os-fsync.html)  强制将文件描述符为fd的文件写入硬盘。 |
| 20 | [os.ftruncate(fd, length)](http://www.runoob.com/python3/python3-os-ftruncate.html)  裁剪文件描述符fd对应的文件, 所以它最大不能超过文件大小。 |
| 21 | [os.getcwd()](http://www.runoob.com/python3/python3-os-getcwd.html)  返回当前工作目录 |
| 22 | [os.getcwdu()](http://www.runoob.com/python3/python3-os-getcwdu.html)  返回一个当前工作目录的Unicode对象 |
| 23 | [os.isatty(fd)](http://www.runoob.com/python3/python3-os-isatty.html)  如果文件描述符fd是打开的，同时与tty(-like)设备相连，则返回true, 否则False。 |
| 24 | [os.lchflags(path, flags)](http://www.runoob.com/python3/python3-os-lchflags.html)  设置路径的标记为数字标记，类似 chflags()，但是没有软链接 |
| 25 | [os.lchmod(path, mode)](http://www.runoob.com/python3/python3-os-lchmod.html)  修改连接文件权限 |
| 26 | [os.lchown(path, uid, gid)](http://www.runoob.com/python3/python3-os-lchown.html)  更改文件所有者，类似 chown，但是不追踪链接。 |
| 27 | [os.link(src, dst)](http://www.runoob.com/python3/python3-os-link.html)  创建硬链接，名为参数 dst，指向参数 src |
| 28 | [os.listdir(path)](http://www.runoob.com/python3/python3-os-listdir.html)  返回path指定的文件夹包含的文件或文件夹的名字的列表。 |
| 29 | [os.lseek(fd, pos, how)](http://www.runoob.com/python3/python3-os-lseek.html)  设置文件描述符 fd当前位置为pos, how方式修改: SEEK\_SET 或者 0 设置从文件开始的计算的pos; SEEK\_CUR或者 1 则从当前位置计算; os.SEEK\_END或者2则从文件尾部开始. 在unix，Windows中有效 |
| 30 | [os.lstat(path)](http://www.runoob.com/python3/python3-os-lstat.html)  像stat(),但是没有软链接 |
| 31 | [os.major(device)](http://www.runoob.com/python3/python3-os-major.html)  从原始的设备号中提取设备major号码 (使用stat中的st\_dev或者st\_rdev field)。 |
| 32 | [os.makedev(major, minor)](http://www.runoob.com/python3/python3-os-makedev.html)  以major和minor设备号组成一个原始设备号 |
| 33 | [os.makedirs(path[, mode])](http://www.runoob.com/python3/python3-os-makedirs.html)  递归文件夹创建函数。像mkdir(), 但创建的所有intermediate-level文件夹需要包含子文件夹。 |
| 34 | [os.minor(device)](http://www.runoob.com/python3/python3-os-minor.html)  从原始的设备号中提取设备minor号码 (使用stat中的st\_dev或者st\_rdev field )。 |
| 35 | [os.mkdir(path[, mode])](http://www.runoob.com/python3/python3-os-mkdir.html)  以数字mode的mode创建一个名为path的文件夹.默认的 mode 是 0777 (八进制)。 |
| 36 | [os.mkfifo(path[, mode])](http://www.runoob.com/python3/python3-os-mkfifo.html)  创建命名管道，mode 为数字，默认为 0666 (八进制) |
| 37 | [os.mknod(filename[, mode=0600, device])](http://www.runoob.com/python3/python3-os-mknod.html) 创建一个名为filename文件系统节点（文件，设备特别文件或者命名pipe）。 |
| 38 | [os.open(file, flags[, mode])](http://www.runoob.com/python3/python3-os-open.html)  打开一个文件，并且设置需要的打开选项，mode参数是可选的 |
| 39 | [os.openpty()](http://www.runoob.com/python3/python3-os-openpty.html)  打开一个新的伪终端对。返回 pty 和 tty的文件描述符。 |
| 40 | [os.pathconf(path, name)](http://www.runoob.com/python3/python3-os-pathconf.html)  返回相关文件的系统配置信息。 |
| 41 | [os.pipe()](http://www.runoob.com/python3/python3-os-pipe.html)  创建一个管道. 返回一对文件描述符(r, w) 分别为读和写 |
| 42 | [os.popen(command[, mode[, bufsize]])](http://www.runoob.com/python3/python3-os-popen.html)  从一个 command 打开一个管道 |
| 43 | [os.read(fd, n)](http://www.runoob.com/python3/python3-os-read.html)  从文件描述符 fd 中读取最多 n 个字节，返回包含读取字节的字符串，文件描述符 fd对应文件已达到结尾, 返回一个空字符串。 |
| 44 | [os.readlink(path)](http://www.runoob.com/python3/python3-os-readlink.html)  返回软链接所指向的文件 |
| 45 | [os.remove(path)](http://www.runoob.com/python3/python3-os-remove.html)  删除路径为path的文件。如果path 是一个文件夹，将抛出OSError; 查看下面的rmdir()删除一个 directory。 |
| 46 | [os.removedirs(path)](http://www.runoob.com/python3/python3-os-removedirs.html)  递归删除目录。 |
| 47 | [os.rename(src, dst)](http://www.runoob.com/python3/python3-os-rename.html)  重命名文件或目录，从 src 到 dst |
| 48 | [os.renames(old, new)](http://www.runoob.com/python3/python3-os-renames.html)  递归地对目录进行更名，也可以对文件进行更名。 |
| 49 | [os.rmdir(path)](http://www.runoob.com/python3/python3-os-rmdir.html)  删除path指定的空目录，如果目录非空，则抛出一个OSError异常。 |
| 50 | [os.stat(path)](http://www.runoob.com/python3/python3-os-stat.html)  获取path指定的路径的信息，功能等同于C API中的stat()系统调用。 |
| 51 | [os.stat\_float\_times([newvalue])](http://www.runoob.com/python3/python3-os-stat_float_times.html) 决定stat\_result是否以float对象显示时间戳 |
| 52 | [os.statvfs(path)](http://www.runoob.com/python3/python3-os-statvfs.html)  获取指定路径的文件系统统计信息 |
| 53 | [os.symlink(src, dst)](http://www.runoob.com/python3/python3-os-symlink.html)  创建一个软链接 |
| 54 | [os.tcgetpgrp(fd)](http://www.runoob.com/python3/python3-os-tcgetpgrp.html)  返回与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组 |
| 55 | [os.tcsetpgrp(fd, pg)](http://www.runoob.com/python3/python3-os-tcsetpgrp.html)  设置与终端fd（一个由os.open()返回的打开的文件描述符）关联的进程组为pg。 |
| 56 | [os.tempnam([dir[, prefix]])](http://www.runoob.com/python3/python3-os-tempnam.html)  返回唯一的路径名用于创建临时文件。 |
| 57 | [os.tmpfile()](http://www.runoob.com/python3/python3-os-tmpfile.html)  返回一个打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。 |
| 58 | [os.tmpnam()](http://www.runoob.com/python3/python3-os-tmpnam.html)  为创建一个临时文件返回一个唯一的路径 |
| 59 | [os.ttyname(fd)](http://www.runoob.com/python3/python3-os-ttyname.html)  返回一个字符串，它表示与文件描述符fd 关联的终端设备。如果fd 没有与终端设备关联，则引发一个异常。 |
| 60 | [os.unlink(path)](http://www.runoob.com/python3/python3-os-unlink.html)  删除文件路径 |
| 61 | [os.utime(path, times)](http://www.runoob.com/python3/python3-os-utime.html)  返回指定的path文件的访问和修改的时间。 |
| 62 | [os.walk(top[, topdown=True[, onerror=None[, followlinks=False]]])](http://www.runoob.com/python3/python3-os-walk.html)  输出在文件夹中的文件名通过在树中游走，向上或者向下。 |
| 63 | [os.write(fd, str)](http://www.runoob.com/python3/python3-os-write.html)  写入字符串到文件描述符 fd中. 返回实际写入的字符串长度 |

# Python3 错误和异常

作为Python初学者，在刚学习Python编程时，经常会看到一些报错信息，在前面我们没有提及，这章节我们会专门介绍。

Python有两种错误很容易辨认：语法错误和异常。

## 语法错误

Python 的语法错误或者称之为解析错，是初学者经常碰到的，如下实例

>>> while True print('Hello world')

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

while True print('Hello world')

^SyntaxError: invalid syntax

这个例子中，函数 print() 被检查到有错误，是它前面缺少了一个冒号（:）。

语法分析器指出了出错的一行，并且在最先找到的错误的位置标记了一个小小的箭头。

## 异常

即便Python程序的语法是正确的，在运行它的时候，也有可能发生错误。运行期检测到的错误被称为异常。

大多数的异常都不会被程序处理，都以错误信息的形式展现在这里:

>>> 10 \* (1/0)Traceback (most recent call last):

File "<stdin>", line 1, in ?ZeroDivisionError: division by zero>>> 4 + spam\*3Traceback (most recent call last):

File "<stdin>", line 1, in ?NameError: name 'spam' is not defined>>> '2' + 2Traceback (most recent call last):

File "<stdin>", line 1, in ?TypeError: Can't convert 'int' object to str implicitly

异常以不同的类型出现，这些类型都作为信息的一部分打印出来: 例子中的类型有 ZeroDivisionError，NameError 和 TypeError。

错误信息的前面部分显示了异常发生的上下文，并以调用栈的形式显示具体信息。

## 异常处理

以下例子中，让用户输入一个合法的整数，但是允许用户中断这个程序（使用 Control-C 或者操作系统提供的方法）。用户中断的信息会引发一个 KeyboardInterrupt 异常。

>>> while True:

try:

x = int(input("Please enter a number: "))

break

except ValueError:

print("Oops! That was no valid number. Try again ")

try语句按照如下方式工作；

* 首先，执行try子句（在关键字try和关键字except之间的语句）
* 如果没有异常发生，忽略except子句，try子句执行后结束。
* 如果在执行try子句的过程中发生了异常，那么try子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的except子句将被执行。最后执行 try 语句之后的代码。
* 如果一个异常没有与任何的except匹配，那么这个异常将会传递给上层的try中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常。最多只有一个分支会被执行。

处理程序将只针对对应的try子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如:

except (RuntimeError, TypeError, NameError):

pass

最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再次把异常抛出。

## 抛出异常

Python 使用 raise 语句抛出一个指定的异常。例如:

>>> raise NameError('HiThere')Traceback (most recent call last):

File "<stdin>", line 1, in ?NameError: HiThere

raise 唯一的一个参数指定了要被抛出的异常。它必须是一个异常的实例或者是异常的类（也就是 Exception 的子类）。

如果你只想知道这是否抛出了一个异常，并不想去处理它，那么一个简单的 raise 语句就可以再次把它抛出。

>>> try:

raise NameError('HiThere')

except NameError:

print('An exception flew by!')

raise

An exception flew by!Traceback (most recent call last):

File "<stdin>", line 2, in ?NameError: HiThere

## 用户自定义异常

你可以通过创建一个新的exception类来拥有自己的异常。异常应该继承自 Exception 类，或者直接继承，或者间接继承，例如:

>>> class MyError(Exception):

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

self.value = value

def \_\_str\_\_(self):

return repr(self.value)

>>> try:

raise MyError(2\*2)

except MyError as e:

print('My exception occurred, value:', e.value)

My exception occurred, value: 4>>> raise MyError('oops!')Traceback (most recent call last):

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

\_\_main\_\_.MyError: 'oops!'

在这个例子中，类 Exception 默认的 \_\_init\_\_() 被覆盖。

<p异常的类可以像其他的类一样做任何事情，但是通常都会比较简单，只提供一些错误相关的属性，并且允许处理异常的代码方便的获取这些信息。< p="" style="color: rgb(51, 51, 51); font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, STHeiti, "Microsoft Yahei", sans-serif; font-size: 18px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: normal; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-decoration-color: initial;">

当创建一个模块有可能抛出多种不同的异常时，一种通常的做法是为这个包建立一个基础异常类，然后基于这个基础类为不同的错误情况创建不同的子类:

## 定义清理行为

try 语句还有另外一个可选的子句，它定义了无论在任何情况下都会执行的清理行为。 例如:

>>> try:...

raise KeyboardInterrupt...

finally:...

print('Goodbye, world!')...

Goodbye, world!Traceback (most recent call last):

File "<stdin>", line 2, in <module>KeyboardInterrupt

以上例子不管 try 子句里面有没有发生异常，finally 子句都会执行。

如果一个异常在 try 子句里（或者在 except 和 else 子句里）被抛出，而又没有任何的 except 把它截住，那么这个异常会在 finally 子句执行后再次被抛出。

下面是一个更加复杂的例子（在同一个 try 语句里包含 except 和 finally 子句）:

# Python3 面向对象

Python从设计之初就已经是一门面向对象的语言，正因为如此，在Python中创建一个类和对象是很容易的。我们将详细介绍Python的面向对象编程。

如果你以前没有接触过面向对象的编程语言，那你可能需要先了解一些面向对象语言的一些基本特征，在头脑里头形成一个基本的面向对象的概念，这样有助于你更容易的学习Python的面向对象编程。

接下来我们先来简单的了解下面向对象的一些基本特征。

## 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。

**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。

和其它编程语言相比，Python 在尽可能不增加新的语法和语义的情况下加入了类机制。

Python中的类提供了面向对象编程的所有基本功能：类的继承机制允许多个基类，派生类可以覆盖基类中的任何方法，方法中可以调用基类中的同名方法。

对象可以包含任意数量和类型的数据。

## 类定义

语法格式如下：

## <1>定义一个类，格式如下：

class 类名:

方法列表

## <2>demo定义一个Cat类

# 定义类

class Cat:

# 方法

def eat(self):

print('猫在吃鱼....')

def drink(self):

print("猫在喝东西...")

类实例化后，可以使用其属性，实际上，创建一个类之后，可以通过类名访问其属性。

## 类对象

类对象支持两种操作：属性引用和实例化。

属性引用使用和 Python 中所有的属性引用一样的标准语法：**obj.name**。

类对象创建后，类命名空间中所有的命名都是有效属性名。所以如果类定义是这样:

### **self代表类的实例，而非类**

类的方法与普通的函数只有一个特别的区别——它们必须有一个额外的**第一个参数名称**, 按照惯例它的名称是 self。

## 继承

Python 同样支持类的继承，如果一种语言不支持继承，类就没有什么意义。派生类的定义如下所示:

## 多继承

Python同样有限的支持多继承形式。多继承的类定义形如下例:

## 方法重写

如果你的父类方法的功能不能满足你的需求，你可以在子类重写你父类的方法，实例如下：

## 类属性与方法

### **类的私有属性**

**\_\_private\_attrs**：两个下划线开头，声明该属性为私有，不能在类地外部被使用或直接访问。在类内部的方法中使用时 **self.\_\_private\_attrs**。

### **类的方法**

在类地内部，使用 def 关键字来定义一个方法，与一般函数定义不同，类方法必须包含参数 **self**，且为第一个参数，**self** 代表的是类的实例。

**self** 的名字并不是规定死的，也可以使用 **this**，但是最好还是按照约定是用 **self**。

### **类的私有方法**

**\_\_private\_method**：两个下划线开头，声明该方法为私有方法，只能在类的内部调用 ，不能在类地外部调用。**self.\_\_private\_methods**。

### **实例**

### **类的专有方法：**

* \_\_init\_\_ : 构造函数，在生成对象时调用
* \_\_del\_\_ : 析构函数，释放对象时使用
* \_\_repr\_\_ : 打印，转换
* \_\_setitem\_\_ : 按照索引赋值
* \_\_getitem\_\_: 按照索引获取值
* \_\_len\_\_: 获得长度
* \_\_cmp\_\_: 比较运算
* \_\_call\_\_: 函数调用
* \_\_add\_\_: 加运算
* \_\_sub\_\_: 减运算
* \_\_mul\_\_: 乘运算
* \_\_div\_\_: 除运算
* \_\_mod\_\_: 求余运算
* \_\_pow\_\_: 乘方

# Python3 标准库概览

## 操作系统接口

os模块提供了不少与操作系统相关联的函数。

>>> import os

>>> os.getcwd() # 返回当前的工作目录

'C:\\Python34'

>>> os.chdir('/server/accesslogs') # 修改当前的工作目录

>>> os.system('mkdir today') # 执行系统命令 mkdir

0

建议使用 "import os" 风格而非 "from os import \*"。这样可以保证随操作系统不同而有所变化的 os.open() 不会覆盖内置函数 open()。

在使用 os 这样的大型模块时内置的 dir() 和 help() 函数非常有用:

>>> import os

>>> dir(os)<returns a list of all module functions

>>>> help(os)<returns an extensive manual page created from the module's docstrings>

针对日常的文件和目录管理任务，:mod:shutil 模块提供了一个易于使用的高级接口:

>>> import shutil

>>> shutil.copyfile('data.db', 'archive.db')

>>> shutil.move('/build/executables', 'installdir

## 文件通配符

glob模块提供了一个函数用于从目录通配符搜索中生成文件列表:

>>> import glob

>>> glob.glob('\*.py')

['primes.py', 'random.py', 'quote.py']

## 命令行参数

通用工具脚本经常调用命令行参数。这些命令行参数以链表形式存储于 sys 模块的 argv 变量。例如在命令行中执行 "python demo.py one two three" 后可以得到以下输出结果:

>>> import sys

>>> print(sys.argv)

['demo.py', 'one', 'two', 'three']

## 错误输出重定向和程序终止

sys 还有 stdin，stdout 和 stderr 属性，即使在 stdout 被重定向时，后者也可以用于显示警告和错误信息。

>>> sys.stderr.write('Warning, log file not found starting a new one\n')

Warning, log file not found starting a new one

大多脚本的定向终止都使用 "sys.exit()"。

## 字符串正则匹配

re模块为高级字符串处理提供了正则表达式工具。对于复杂的匹配和处理，正则表达式提供了简洁、优化的解决方案:

>>> import re

>>> re.findall(r'\bf[a-z]\*', 'which foot or hand fell fastest')

['foot', 'fell', 'fastest']

>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')

'cat in the hat'

如果只需要简单的功能，应该首先考虑字符串方法，因为它们非常简单，易于阅读和调试:

>>> 'tea for too'.replace('too', 'two')'tea for two'

## 数学

math模块为浮点运算提供了对底层C函数库的访问:

>>> import math

>>> math.cos(math.pi / 4)

0.70710678118654757

>>> math.log(1024, 2)

10.0

random提供了生成随机数的工具。

>>> import random

>>> random.choice(['apple', 'pear', 'banana'])

'apple'

>>> random.sample(range(100), 10) # sampling without replacement

[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]

>>> random.random() # random float

0.17970987693706186

>>> random.randrange(6) # random integer chosen from range(6)

4

## 访问互联网

有几个模块用于访问互联网以及处理网络通信协议。其中最简单的两个是用于处理从 urls 接收的数据的 urllib.request 以及用于发送电子邮件的 smtplib:

>>> from urllib.request import urlopen

>>> for line in urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):

... line = line.decode('utf-8') # Decoding the binary data to text.

... if 'EST' in line or 'EDT' in line: # look for Eastern Time

... print(line)

<BR>Nov. 25, 09:43:32 PM EST

>>> import smtplib

>>> server = smtplib.SMTP('localhost')

>>> server.sendmail('soothsayer@example.org', ['jcaesar@example.org',](mailto:'jcaesar@example.org',)

... """To: jcaesar@example.org

... From: soothsayer@example.org

...

... Beware the Ides of March.

... """)

>>> server.quit()

注意第二个例子需要本地有一个在运行的邮件服务器。

## 日期和时间

datetime模块为日期和时间处理同时提供了简单和复杂的方法。

支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。

该模块还支持时区处理:

>>> # dates are easily constructed and formatted

>>> from datetime import date

>>> now = date.today()

>>> now

datetime.date(2003, 12, 2)

>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic

>>> birthday = date(1964, 7, 31)

>>> age = now - birthday

>>> age.days

14368

## 数据压缩

以下模块直接支持通用的数据打包和压缩格式：zlib，gzip，bz2，zipfile，以及 tarfile。

>>> import zlib

>>> s = b'witch which has which witches wrist watch'

>>> len(s)

41

>>> t = zlib.compress(s)

>>> len(t)

37

>>> zlib.decompress(t)

b'witch which has which witches wrist watch'

>>> zlib.crc32(s)

226805979

## 性能度量

有些用户对了解解决同一问题的不同方法之间的性能差异很感兴趣。Python 提供了一个度量工具，为这些问题提供了直接答案。

例如，使用元组封装和拆封来交换元素看起来要比使用传统的方法要诱人的多,timeit 证明了现代的方法更快一些。

>>> from timeit import Timer

>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()

0.57535828626024577

>>> Timer('a,b = b,a', 'a=1; b=2').timeit()

0.54962537085770791

相对于 timeit 的细粒度，:mod:profile 和 pstats 模块提供了针对更大代码块的时间度量工具。

## 测试模块

开发高质量软件的方法之一是为每一个函数开发测试代码，并且在开发过程中经常进行测试

doctest模块提供了一个工具，扫描模块并根据程序中内嵌的文档字符串执行测试。

测试构造如同简单的将它的输出结果剪切并粘贴到文档字符串中。

通过用户提供的例子，它强化了文档，允许 doctest 模块确认代码的结果是否与文档一致:

def average(values):

"""Computes the arithmetic mean of a list of numbers.

>>> print(average([20, 30, 70]))

40.0

"""

return sum(values) / len(values)

import doctest

doctest.testmod() # 自动验证嵌入测试

unittest模块不像 doctest模块那么容易使用，不过它可以在一个独立的文件里提供一个更全面的测试集:

import unittest

class TestStatisticalFunctions(unittest.TestCase):

def test\_average(self):

self.assertEqual(average([20, 30, 70]), 40.0)

self.assertEqual(round(average([1, 5, 7]), 1), 4.3)

self.assertRaises(ZeroDivisionError, average, [])

self.assertRaises(TypeError, average, 20, 30, 70)

unittest.main() # Calling from the command line invokes all tests

# Python3 正则表达式

正则表达式是一个特殊的字符序列，它能帮助你方便的检查一个字符串是否与某种模式匹配。

Python 自1.5版本起增加了re 模块，它提供 Perl 风格的正则表达式模式。

re 模块使 Python 语言拥有全部的正则表达式功能。

compile 函数根据一个模式字符串和可选的标志参数生成一个正则表达式对象。该对象拥有一系列方法用于正则表达式匹配和替换。

re 模块也提供了与这些方法功能完全一致的函数，这些函数使用一个模式字符串做为它们的第一个参数。

本章节主要介绍Python中常用的正则表达式处理函数。

## re.match函数

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。

**函数语法**：

re.match(pattern, string, flags=0)

函数参数说明：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。 |

匹配成功re.match方法返回一个匹配的对象，否则返回None。

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

|  |  |
| --- | --- |
| **匹配对象方法** | **描述** |
| group(num=0) | 匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。 |
| groups() | 返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。 |

实例 1：

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

import reprint(re.match('www', 'www.atguigu.com').span()) # 在起始位置匹配print(re.match('com', 'www.atguigu.com')) # 不在起始位置匹配

以上实例运行输出结果为：

(0, 3)None

实例 2：

#!/usr/bin/python3import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.\*) are (.\*?) .\*', line, re.M|re.I)

if matchObj:

print ("matchObj.group() : ", matchObj.group())

print ("matchObj.group(1) : ", matchObj.group(1))

print ("matchObj.group(2) : ", matchObj.group(2))else:

print ("No match!!")

以上实例执行结果如下：

matchObj.group() : Cats are smarter than dogs

matchObj.group(1) : Cats

matchObj.group(2) : smarter

## re.search方法

re.search 扫描整个字符串并返回第一个成功的匹配。

函数语法：

re.search(pattern, string, flags=0)

函数参数说明：

|  |  |
| --- | --- |
| **参数** | **描述** |
| pattern | 匹配的正则表达式 |
| string | 要匹配的字符串。 |
| flags | 标志位，用于控制正则表达式的匹配方式，如：是否区分大小写，多行匹配等等。 |

匹配成功re.search方法返回一个匹配的对象，否则返回None。

我们可以使用group(num) 或 groups() 匹配对象函数来获取匹配表达式。

|  |  |
| --- | --- |
| **匹配对象方法** | **描述** |
| group(num=0) | 匹配的整个表达式的字符串，group() 可以一次输入多个组号，在这种情况下它将返回一个包含那些组所对应值的元组。 |
| groups() | 返回一个包含所有小组字符串的元组，从 1 到 所含的小组号。 |

实例 1：

#!/usr/bin/python3

import re

print(re.search('www', 'www.atguigu.com').span()) # 在起始位置匹配print(re.search('com', 'www.atguigu.com').span()) # 不在起始位置匹配

以上实例运行输出结果为：

(0, 3)(11, 14)

实例 2：

#!/usr/bin/python3

import re

line = "Cats are smarter than dogs";

searchObj = re.search( r'(.\*) are (.\*?) .\*', line, re.M|re.I)

if searchObj:

print ("searchObj.group() : ", searchObj.group())

print ("searchObj.group(1) : ", searchObj.group(1))

print ("searchObj.group(2) : ", searchObj.group(2))else:

print ("Nothing found!!")

以上实例执行结果如下：

searchObj.group() : Cats are smarter than dogs

searchObj.group(1) : Cats

searchObj.group(2) : smarter

## re.match与re.search的区别

re.match只匹配字符串的开始，如果字符串开始不符合正则表达式，则匹配失败，函数返回None；而re.search匹配整个字符串，直到找到一个匹配。

实例：

#!/usr/bin/python3

import re

line = "Cats are smarter than dogs";

matchObj = re.match( r'dogs', line, re.M|re.I)if matchObj:

print ("match --> matchObj.group() : ", matchObj.group())else:

print ("No match!!")

matchObj = re.search( r'dogs', line, re.M|re.I)if matchObj:

print ("search --> matchObj.group() : ", matchObj.group())else:

print ("No match!!")

以上实例运行结果如下：

No match!!

search --> matchObj.group() : dogs

## 检索和替换

Python 的re模块提供了re.sub用于替换字符串中的匹配项。

语法：

re.sub(pattern, repl, string, count=0)

参数：

* pattern : 正则中的模式字符串。
* repl : 替换的字符串，也可为一个函数。
* string : 要被查找替换的原始字符串。
* count : 模式匹配后替换的最大次数，默认 0 表示替换所有的匹配。

实例：

#!/usr/bin/python3import re

phone = "2004-959-559 # 这是一个电话号码"

# 删除注释

num = re.sub(r'#.\*$', "", phone)print ("电话号码 : ", num)

# 移除非数字的内容

num = re.sub(r'\D', "", phone)print ("电话号码 : ", num)

以上实例执行结果如下：

电话号码 : 2004-959-559 电话号码 : 2004959559

### **repl 参数是一个函数**

以下实例中将字符串中的匹配的数字乘于 2：

#!/usr/bin/python

import re

# 将匹配的数字乘于 2def double(matched):

value = int(matched.group('value'))

return str(value \* 2)

s = 'A23G4HFD567'print(re.sub('(?P<value>\d+)', double, s))

执行输出结果为：

A46G8HFD1134

## 正则表达式修饰符 - 可选标志

正则表达式可以包含一些可选标志修饰符来控制匹配的模式。修饰符被指定为一个可选的标志。多个标志可以通过按位 OR(|) 它们来指定。如 re.I | re.M 被设置成 I 和 M 标志：

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| re.I | 使匹配对大小写不敏感 |
| re.L | 做本地化识别（locale-aware）匹配 |
| re.M | 多行匹配，影响 ^ 和 $ |
| re.S | 使 . 匹配包括换行在内的所有字符 |
| re.U | 根据Unicode字符集解析字符。这个标志影响 \w, \W, \b, \B. |
| re.X | 该标志通过给予你更灵活的格式以便你将正则表达式写得更易于理解。 |

## 正则表达式模式

模式字符串使用特殊的语法来表示一个正则表达式：

字母和数字表示他们自身。一个正则表达式模式中的字母和数字匹配同样的字符串。

多数字母和数字前加一个反斜杠时会拥有不同的含义。

标点符号只有被转义时才匹配自身，否则它们表示特殊的含义。

反斜杠本身需要使用反斜杠转义。

由于正则表达式通常都包含反斜杠，所以你最好使用原始字符串来表示它们。模式元素(如 r'/t'，等价于'//t')匹配相应的特殊字符。

下表列出了正则表达式模式语法中的特殊元素。如果你使用模式的同时提供了可选的标志参数，某些模式元素的含义会改变。

|  |  |
| --- | --- |
| **模式** | **描述** |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的末尾。 |
| . | 匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。 |
| [...] | 用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k' |
| [^...] | 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。 |
| re\* | 匹配0个或多个的表达式。 |
| re+ | 匹配1个或多个的表达式。 |
| re? | 匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式 |
| re{ n} |  |
| re{ n,} | 精确匹配n个前面表达式。 |
| re{ n, m} | 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式 |
| a| b | 匹配a或b |
| (re) | G匹配括号内的表达式，也表示一个组 |
| (?imx) | 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。 |
| (?-imx) | 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。 |
| (?: re) | 类似 (...), 但是不表示一个组 |
| (?imx: re) | 在括号中使用i, m, 或 x 可选标志 |
| (?-imx: re) | 在括号中不使用i, m, 或 x 可选标志 |
| (?#...) | 注释. |
| (?= re) | 前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。 |
| (?! re) | 前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功 |
| (?> re) | 匹配的独立模式，省去回溯。 |
| \w | 匹配字母数字 |
| \W | 匹配非字母数字 |
| \s | 匹配任意空白字符，等价于 [\t\n\r\f]. |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于 [0-9]. |
| \D | 匹配任意非数字 |
| \A | 匹配字符串开始 |
| \Z | 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。c |
| \z | 匹配字符串结束 |
| \G | 匹配最后匹配完成的位置。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。 |
| \B | 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。 |
| \n, \t, 等. | 匹配一个换行符。匹配一个制表符。等 |
| \1...\9 | 匹配第n个分组的内容。 |
| \10 | 匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。 |

## 正则表达式实例

#### 字符匹配

|  |  |
| --- | --- |
| **实例** | **描述** |
| python | 匹配 "python". |

#### 字符类

|  |  |
| --- | --- |
| **实例** | **描述** |
| [Pp]ython | 匹配 "Python" 或 "python" |
| rub[ye] | 匹配 "ruby" 或 "rube" |
| [aeiou] | 匹配中括号内的任意一个字母 |
| [0-9] | 匹配任何数字。类似于 [0123456789] |
| [a-z] | 匹配任何小写字母 |
| [A-Z] | 匹配任何大写字母 |
| [a-zA-Z0-9] | 匹配任何字母及数字 |
| [^aeiou] | 除了aeiou字母以外的所有字符 |
| [^0-9] | 匹配除了数字外的字符 |

#### 特殊字符类

|  |  |
| --- | --- |
| **实例** | **描述** |
| . | 匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。 |
| \d | 匹配一个数字字符。等价于 [0-9]。 |
| \D | 匹配一个非数字字符。等价于 [^0-9]。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。 |
| \w | 匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9\_]'。 |
| \W | 匹配任何非单词字符。等价于 '[^A-Za-z0-9\_]'。 |

# Python CGI编程

## 什么是CGI

CGI 目前由NCSA维护，NCSA定义CGI如下：

CGI(Common Gateway Interface),通用网关接口,它是一段程序,运行在服务器上如：HTTP服务器，提供同客户端HTML页面的接口。

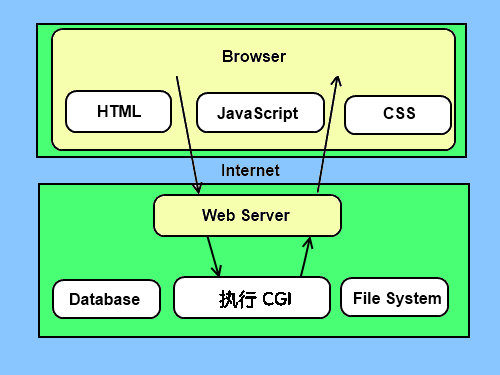
## 网页浏览

为了更好的了解CGI是如何工作的，我们可以从在网页上点击一个链接或URL的流程：

* 1、使用你的浏览器访问URL并连接到HTTP web 服务器。
* 2、Web服务器接收到请求信息后会解析URL，并查找访问的文件在服务器上是否存在，如果存在返回文件的内容，否则返回错误信息。
* 3、浏览器从服务器上接收信息，并显示接收的文件或者错误信息。

CGI程序可以是Python脚本，PERL脚本，SHELL脚本，C或者C++程序等。

## CGI架构图



## Web服务器支持及配置

在你进行CGI编程前，确保您的Web服务器支持CGI及已经配置了CGI的处理程序。

Apache 支持CGI 配置：

设置好CGI目录：

ScriptAlias /cgi-bin/ /var/www/cgi-bin/

所有的HTTP服务器执行CGI程序都保存在一个预先配置的目录。这个目录被称为CGI目录，并按照惯例，它被命名为/var/www/cgi-bin目录。

CGI文件的扩展名为.cgi，python也可以使用.py扩展名。

默认情况下，Linux服务器配置运行的cgi-bin目录中为/var/www。

如果你想指定其他运行CGI脚本的目录，可以修改httpd.conf配置文件，如下所示：

<Directory "/var/www/cgi-bin">

AllowOverride None

Options +ExecCGI

Order allow,deny

Allow from all</Directory>

在 AddHandler 中添加 .py 后缀，这样我们就可以访问 .py 结尾的 python 脚本文件：

AddHandler cgi-script .cgi .pl .py

# Python3 MySQL 数据库连接

本文我们为大家介绍 Python3 使用 [PyMySQL](https://github.com/PyMySQL/PyMySQL" \t "http://www.runoob.com/python3/_blank) 连接数据库，并实现简单的增删改查。

### **什么是 PyMySQL？**

PyMySQL 是在 Python3.x 版本中用于连接 MySQL 服务器的一个库，Python2中则使用mysqldb。

PyMySQL 遵循 Python 数据库 API v2.0 规范，并包含了 pure-Python MySQL 客户端库。

## PyMySQL 安装

在使用 PyMySQL 之前，我们需要确保 PyMySQL 已安装。

PyMySQL 下载地址：https://github.com/PyMySQL/PyMySQL。

如果还未安装，我们可以使用以下命令安装最新版的 PyMySQL：

$ pip install PyMySQL

如果你的系统不支持 pip 命令，可以使用以下方式安装：

1、使用 git 命令下载安装包安装(你也可以手动下载)：

$ git clone https://github.com/PyMySQL/PyMySQL

$ cd PyMySQL/

$ python3 setup.py install

2、如果需要制定版本号，可以使用 curl 命令来安装：

$ # X.X 为 PyMySQL 的版本号

$ curl -L https://github.com/PyMySQL/PyMySQL/tarball/pymysql-X.X | tar xz

$ cd PyMySQL\*

$ python3 setup.py install

$ # 现在你可以删除 PyMySQL\* 目录

**注意：**请确保您有root权限来安装上述模块。

*安装的过程中可能会出现"ImportError: No module named setuptools"的错误提示，意思是你没有安装setuptools，你可以访问[https://pypi.python.org/pypi/setuptools](https://pypi.python.org/pypi/setuptools" \t "http://www.runoob.com/python3/_blank) 找到各个系统的安装方法。*

*Linux 系统安装实例：*

$ wget https://bootstrap.pypa.io/ez\_setup.py

$ python3 ez\_setup.py

## 数据库连接

连接数据库前，请先确认以下事项：

* 您已经创建了数据库 TESTDB.
* 在TESTDB数据库中您已经创建了表 EMPLOYEE
* EMPLOYEE表字段为 FIRST\_NAME, LAST\_NAME, AGE, SEX 和 INCOME。
* 连接数据库TESTDB使用的用户名为 "testuser" ，密码为 "test123",你可以可以自己设定或者直接使用root用户名及其密码，Mysql数据库用户授权请使用Grant命令。
* 在你的机子上已经安装了 Python MySQLdb 模块。
* 如果您对sql语句不熟悉，可以访问我们的 [SQL基础教程](http://www.runoob.com/sql/sql-tutorial.html)

### **实例：**

以下实例链接Mysql的TESTDB数据库：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用 cursor() 方法创建一个游标对象 cursor

cursor = db.cursor()

# 使用 execute() 方法执行 SQL 查询

cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取单条数据.

data = cursor.fetchone()

print ("Database version : %s " % data)

# 关闭数据库连接

db.close()

执行以上脚本输出结果如下：

Database version : 5.5.20-log

## 创建数据库表

如果数据库连接存在我们可以使用execute()方法来为数据库创建表，如下所示创建表EMPLOYEE：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用 cursor() 方法创建一个游标对象 cursor

cursor = db.cursor()

# 使用 execute() 方法执行 SQL，如果表存在则删除

cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# 使用预处理语句创建表

sql = """CREATE TABLE EMPLOYEE (

FIRST\_NAME CHAR(20) NOT NULL,

LAST\_NAME CHAR(20),

AGE INT,

SEX CHAR(1),

INCOME FLOAT )"""

cursor.execute(sql)

# 关闭数据库连接

db.close()

## 数据库插入操作

以下实例使用执行 SQL INSERT 语句向表 EMPLOYEE 插入记录：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 插入语句

sql = """INSERT INTO EMPLOYEE(FIRST\_NAME,

LAST\_NAME, AGE, SEX, INCOME)

VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:

# 执行sql语句

cursor.execute(sql)

# 提交到数据库执行

db.commit()

except:

# 如果发生错误则回滚

db.rollback()

# 关闭数据库连接

db.close()

以上例子也可以写成如下形式：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 插入语句

sql = "INSERT INTO EMPLOYEE(FIRST\_NAME, \

LAST\_NAME, AGE, SEX, INCOME) \

VALUES ('%s', '%s', '%d', '%c', '%d' )" % \

('Mac', 'Mohan', 20, 'M', 2000)

try:

# 执行sql语句

cursor.execute(sql)

# 执行sql语句

db.commit()

except:

# 发生错误时回滚

db.rollback()

# 关闭数据库连接

db.close()

以下代码使用变量向SQL语句中传递参数:

..................................

user\_id = "test123"

password = "password"

con.execute('insert into Login values("%s", "%s")' % \

(user\_id, password))

..................................

## 数据库查询操作

Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。

* **fetchone():** 该方法获取下一个查询结果集。结果集是一个对象
* **fetchall():**接收全部的返回结果行.
* **rowcount:** 这是一个只读属性，并返回执行execute()方法后影响的行数。

### **实例：**

查询EMPLOYEE表中salary（工资）字段大于1000的所有数据：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 查询语句

sql = "SELECT \* FROM EMPLOYEE \

WHERE INCOME > '%d'" % (1000)

try:

# 执行SQL语句

cursor.execute(sql)

# 获取所有记录列表

results = cursor.fetchall()

for row in results:

fname = row[0]

lname = row[1]

age = row[2]

sex = row[3]

income = row[4]

# 打印结果

print ("fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \

(fname, lname, age, sex, income ))

except:

print ("Error: unable to fetch data")

# 关闭数据库连接

db.close()

以上脚本执行结果如下：

fname=Mac, lname=Mohan, age=20, sex=M, income=2000

## 数据库更新操作

更新操作用于更新数据表的的数据，以下实例将 TESTDB表中的 SEX 字段全部修改为 'M'，AGE 字段递增1：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 更新语句

sql = "UPDATE EMPLOYEE SET AGE = AGE + 1

WHERE SEX = '%c'" % ('M')

try:

# 执行SQL语句

cursor.execute(sql)

# 提交到数据库执行

db.commit()

except:

# 发生错误时回滚

db.rollback()

# 关闭数据库连接

db.close()

## 删除操作

删除操作用于删除数据表中的数据，以下实例演示了删除数据表 EMPLOYEE 中 AGE 大于 20 的所有数据：

#!/usr/bin/python3

import pymysql

# 打开数据库连接

db = pymysql.connect("localhost","testuser","test123","TESTDB" )

# 使用cursor()方法获取操作游标

cursor = db.cursor()

# SQL 删除语句

sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

try:

# 执行SQL语句

cursor.execute(sql)

# 提交修改

db.commit()

except:

# 发生错误时回滚

db.rollback()

# 关闭连接

db.close()

## 执行事务

事务机制可以确保数据一致性。

事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。

* 原子性（atomicity）。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
* 一致性（consistency）。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
* 隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
* 持久性（durability）。持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

Python DB API 2.0 的事务提供了两个方法 commit 或 rollback。

### **实例**

# SQL删除记录语句

sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

try:

# 执行SQL语句

cursor.execute(sql)

# 向数据库提交

db.commit()

except:

# 发生错误时回滚

db.rollback()

对于支持事务的数据库， 在Python数据库编程中，当游标建立之时，就自动开始了一个隐形的数据库事务。

commit()方法游标的所有更新操作，rollback（）方法回滚当前游标的所有操作。每一个方法都开始了一个新的事务。

## 错误处理

DB API中定义了一些数据库操作的错误及异常，下表列出了这些错误和异常:

|  |  |
| --- | --- |
| **异常** | **描述** |
| Warning | 当有严重警告时触发，例如插入数据是被截断等等。必须是 StandardError 的子类。 |
| Error | 警告以外所有其他错误类。必须是 StandardError 的子类。 |
| InterfaceError | 当有数据库接口模块本身的错误（而不是数据库的错误）发生时触发。 必须是Error的子类。 |
| DatabaseError | 和数据库有关的错误发生时触发。 必须是Error的子类。 |
| DataError | 当有数据处理时的错误发生时触发，例如：除零错误，数据超范围等等。 必须是DatabaseError的子类。 |
| OperationalError | 指非用户控制的，而是操作数据库时发生的错误。例如：连接意外断开、 数据库名未找到、事务处理失败、内存分配错误等等操作数据库是发生的错误。 必须是DatabaseError的子类。 |
| IntegrityError | 完整性相关的错误，例如外键检查失败等。必须是DatabaseError子类。 |
| InternalError | 数据库的内部错误，例如游标（cursor）失效了、事务同步失败等等。 必须是DatabaseError子类。 |
| ProgrammingError | 程序错误，例如数据表（table）没找到或已存在、SQL语句语法错误、 参数数量错误等等。必须是DatabaseError的子类。 |
| NotSupportedError | 不支持错误，指使用了数据库不支持的函数或API等。例如在连接对象上 使用.rollback()函数，然而数据库并不支持事务或者事务已关闭。 必须是DatabaseError的子类。 |

# Python3 SMTP发送邮件

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。

python的smtplib提供了一种很方便的途径发送电子邮件。它对smtp协议进行了简单的封装。

Python创建 SMTP 对象语法如下：

import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local\_hostname]]] )

参数说明：

* host: SMTP 服务器主机。 你可以指定主机的ip地址或者域名如:w3cschool.cc，这个是可选参数。
* port: 如果你提供了 host 参数, 你需要指定 SMTP 服务使用的端口号，一般情况下SMTP端口号为25。
* local\_hostname: 如果SMTP在你的本机上，你只需要指定服务器地址为 localhost 即可。

Python SMTP对象使用sendmail方法发送邮件，语法如下：

SMTP.sendmail(from\_addr, to\_addrs, msg[, mail\_options, rcpt\_options]

参数说明：

* from\_addr: 邮件发送者地址。
* to\_addrs: 字符串列表，邮件发送地址。
* msg: 发送消息

这里要注意一下第三个参数，msg是字符串，表示邮件。我们知道邮件一般由标题，发信人，收件人，邮件内容，附件等构成，发送邮件的时候，要注意msg的格式。这个格式就是smtp协议中定义的格式。

## 使用Python发送HTML格式的邮件

Python发送HTML格式的邮件与发送纯文本消息的邮件不同之处就是将MIMEText中\_subtype设置为html

## Python 发送带附件的邮件

发送带附件的邮件，首先要创建MIMEMultipart()实例，然后构造附件，如果有多个附件，可依次构造，最后利用smtplib.smtp发送。

## 在 HTML 文本中添加图片

邮件的 HTML 文本中一般邮件服务商添加外链是无效的，正确添加突破的实例如下所示：

# Python3 多线程

多线程类似于同时执行多个不同程序，多线程运行有如下优点：

* 使用线程可以把占据长时间的程序中的任务放到后台去处理。
* 用户界面可以更加吸引人，这样比如用户点击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度
* 程序的运行速度可能加快
* 在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下我们可以释放一些珍贵的资源如内存占用等等。

线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

每个线程都有他自己的一组CPU寄存器，称为线程的上下文，该上下文反映了线程上次运行该线程的CPU寄存器的状态。

指令指针和堆栈指针寄存器是线程上下文中两个最重要的寄存器，线程总是在进程得到上下文中运行的，这些地址都用于标志拥有线程的进程地址空间中的内存。

* 线程可以被抢占（中断）。
* 在其他线程正在运行时，线程可以暂时搁置（也称为睡眠） -- 这就是线程的退让。

线程可以分为:

* **内核线程：**由操作系统内核创建和撤销。
* **用户线程：**不需要内核支持而在用户程序中实现的线程。

Python3 线程中常用的两个模块为：

* **\_thread**
* **threading(推荐使用)**

thread 模块已被废弃。用户可以使用 threading 模块代替。所以，在 Python3 中不能再使用"thread" 模块。为了兼容性，Python3 将 thread 重命名为 "\_thread"。

## 开始学习Python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：

#!/usr/bin/python3

import \_threadimport 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: 无法启动线程")

while 1:

pass

执行以上程序输出结果如下：

Thread-1: Wed Apr 6 11:36:31 2016Thread-1: Wed Apr 6 11:36:33 2016Thread-2: Wed Apr 6 11:36:33 2016Thread-1: Wed Apr 6 11:36:35 2016Thread-1: Wed Apr 6 11:36:37 2016Thread-2: Wed Apr 6 11:36:37 2016Thread-1: Wed Apr 6 11:36:39 2016Thread-2: Wed Apr 6 11:36:41 2016Thread-2: Wed Apr 6 11:36:45 2016Thread-2: Wed Apr 6 11:36:49 2016

执行以上程后可以按下 ctrl-c to 退出。

## 线程模块

Python3 通过两个标准库 \_thread 和 threading 提供对线程的支持。

\_thread 提供了低级别的、原始的线程以及一个简单的锁，它相比于 threading 模块的功能还是比较有限的。

threading 模块除了包含 \_thread 模块中的所有方法外，还提供的其他方法：

* threading.currentThread(): 返回当前的线程变量。
* threading.enumerate(): 返回一个包含正在运行的线程的list。正在运行指线程启动后、结束前，不包括启动前和终止后的线程。
* threading.activeCount(): 返回正在运行的线程数量，与len(threading.enumerate())有相同的结果。

除了使用方法外，线程模块同样提供了Thread类来处理线程，Thread类提供了以下方法:

* **run():** 用以表示线程活动的方法。
* **start():**启动线程活动。
* **join([time]):** 等待至线程中止。这阻塞调用线程直至线程的join() 方法被调用中止-正常退出或者抛出未处理的异常-或者是可选的超时发生。
* **isAlive():** 返回线程是否活动的。
* **getName():** 返回线程名。
* **setName():** 设置线程名。

## 使用 threading 模块创建线程

我们可以通过直接从 threading.Thread 继承创建一个新的子类，并实例化后调用 start() 方法启动新线程，即它调用了线程的 run() 方法：

#!/usr/bin/python3

import threadingimport time

exitFlag = 0

class myThread (threading.Thread):

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

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("开始线程：" + self.name)

print\_time(self.name, self.counter, 5)

print ("退出线程：" + self.name)

def print\_time(threadName, delay, counter):

while counter:

if exitFlag:

threadName.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)

# 开启新线程

thread1.start()

thread2.start()

thread1.join()

thread2.join()print ("退出主线程")

以上程序执行结果如下；

开始线程：Thread-1开始线程：Thread-2Thread-1: Wed Apr 6 11:46:46 2016Thread-1: Wed Apr 6 11:46:47 2016Thread-2: Wed Apr 6 11:46:47 2016Thread-1: Wed Apr 6 11:46:48 2016Thread-1: Wed Apr 6 11:46:49 2016Thread-2: Wed Apr 6 11:46:49 2016Thread-1: Wed Apr 6 11:46:50 2016退出线程：Thread-1Thread-2: Wed Apr 6 11:46:51 2016Thread-2: Wed Apr 6 11:46:53 2016Thread-2: Wed Apr 6 11:46:55 2016退出线程：Thread-2退出主线程

## 线程同步

如果多个线程共同对某个数据修改，则可能出现不可预料的结果，为了保证数据的正确性，需要对多个线程进行同步。

使用 Thread 对象的 Lock 和 Rlock 可以实现简单的线程同步，这两个对象都有 acquire 方法和 release 方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到 acquire 和 release 方法之间。如下：

多线程的优势在于可以同时运行多个任务（至少感觉起来是这样）。但是当线程需要共享数据时，可能存在数据不同步的问题。

考虑这样一种情况：一个列表里所有元素都是0，线程"set"从后向前把所有元素改成1，而线程"print"负责从前往后读取列表并打印。

那么，可能线程"set"开始改的时候，线程"print"便来打印列表了，输出就成了一半0一半1，这就是数据的不同步。为了避免这种情况，引入了锁的概念。

锁有两种状态——锁定和未锁定。每当一个线程比如"set"要访问共享数据时，必须先获得锁定；如果已经有别的线程比如"print"获得锁定了，那么就让线程"set"暂停，也就是同步阻塞；等到线程"print"访问完毕，释放锁以后，再让线程"set"继续。

经过这样的处理，打印列表时要么全部输出0，要么全部输出1，不会再出现一半0一半1的尴尬场面。

实例：

#!/usr/bin/python3

import threadingimport time

class myThread (threading.Thread):

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

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print ("开启线程： " + self.name)

# 获取锁，用于线程同步

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# 释放锁，开启下一个线程

threadLock.release()

def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print ("%s: %s" % (threadName, time.ctime(time.time())))

counter -= 1

threadLock = threading.Lock()

threads = []

# 创建新线程

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)

# 开启新线程

thread1.start()

thread2.start()

# 添加线程到线程列表

threads.append(thread1)

threads.append(thread2)

# 等待所有线程完成for t in threads:

t.join()print ("退出主线程")

执行以上程序，输出结果为：

开启线程： Thread-1开启线程： Thread-2Thread-1: Wed Apr 6 11:52:57 2016Thread-1: Wed Apr 6 11:52:58 2016Thread-1: Wed Apr 6 11:52:59 2016Thread-2: Wed Apr 6 11:53:01 2016Thread-2: Wed Apr 6 11:53:03 2016Thread-2: Wed Apr 6 11:53:05 2016退出主线程

## 线程优先级队列（ Queue）

Python 的 Queue 模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出)队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列 PriorityQueue。

这些队列都实现了锁原语，能够在多线程中直接使用，可以使用队列来实现线程间的同步。

Queue 模块中的常用方法:

* Queue.qsize() 返回队列的大小
* Queue.empty() 如果队列为空，返回True,反之False
* Queue.full() 如果队列满了，返回True,反之False
* Queue.full 与 maxsize 大小对应
* Queue.get([block[, timeout]])获取队列，timeout等待时间
* Queue.get\_nowait() 相当Queue.get(False)
* Queue.put(item) 写入队列，timeout等待时间
* Queue.put\_nowait(item) 相当Queue.put(item, False)
* Queue.task\_done() 在完成一项工作之后，Queue.task\_done()函数向任务已经完成的队列发送一个信号
* Queue.join() 实际上意味着等到队列为空，再执行别的操作

实例:

#!/usr/bin/python3

import queueimport threadingimport time

exitFlag = 0

class myThread (threading.Thread):

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

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.q = q

def run(self):

print ("开启线程：" + self.name)

process\_data(self.name, self.q)

print ("退出线程：" + self.name)

def process\_data(threadName, q):

while not exitFlag:

queueLock.acquire()

if not workQueue.empty():

data = q.get()

queueLock.release()

print ("%s processing %s" % (threadName, data))

else:

queueLock.release()

time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]

nameList = ["One", "Two", "Three", "Four", "Five"]

queueLock = threading.Lock()

workQueue = queue.Queue(10)

threads = []

threadID = 1

# 创建新线程for tName in threadList:

thread = myThread(threadID, tName, workQueue)

thread.start()

threads.append(thread)

threadID += 1

# 填充队列

queueLock.acquire()for word in nameList:

workQueue.put(word)

queueLock.release()

# 等待队列清空while not workQueue.empty():

pass

# 通知线程是时候退出

exitFlag = 1

# 等待所有线程完成for t in threads:

t.join()print ("退出主线程")

以上程序执行结果：

开启线程：Thread-1开启线程：Thread-2开启线程：Thread-3Thread-3 processing OneThread-1 processing TwoThread-2 processing ThreeThread-3 processing FourThread-1 processing Five退出线程：Thread-3退出线程：Thread-2退出线程：Thread-1退出主线程

# Python3 XML解析

## 什么是XML？

XML 指可扩展标记语言（e**X**tensible **M**arkup **L**anguage），标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。 你可以通过本站学习[XML教程](http://www.runoob.com/xml/xml-tutorial.html" \o "XML教程" \t "http://www.runoob.com/python3/_blank)

XML 被设计用来传输和存储数据。

XML是一套定义语义标记的规则，这些标记将文档分成许多部件并对这些部件加以标识。

它也是元标记语言，即定义了用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。

## python对XML的解析

常见的XML编程接口有DOM和SAX，这两种接口处理XML文件的方式不同，当然使用场合也不同。

python有三种方法解析XML，SAX，DOM，以及ElementTree:

### **1.SAX (simple API for XML )**

python 标准库包含SAX解析器，SAX用事件驱动模型，通过在解析XML的过程中触发一个个的事件并调用用户定义的回调函数来处理XML文件。

### **2.DOM(Document Object Model)**

将XML数据在内存中解析成一个树，通过对树的操作来操作XML。

本章节使用到的XML实例文件movies.xml内容如下：

<collection shelf="New Arrivals"><movie title="Enemy Behind">

<type>War, Thriller</type>

<format>DVD</format>

<year>2003</year>

<rating>PG</rating>

<stars>10</stars>

<description>Talk about a US-Japan war</description></movie><movie title="Transformers">

<type>Anime, Science Fiction</type>

<format>DVD</format>

<year>1989</year>

<rating>R</rating>

<stars>8</stars>

<description>A schientific fiction</description></movie>

<movie title="Trigun">

<type>Anime, Action</type>

<format>DVD</format>

<episodes>4</episodes>

<rating>PG</rating>

<stars>10</stars>

<description>Vash the Stampede!</description></movie><movie title="Ishtar">

<type>Comedy</type>

<format>VHS</format>

<rating>PG</rating>

<stars>2</stars>

<description>Viewable boredom</description></movie></collection>

## python使用SAX解析xml

SAX是一种基于事件驱动的API。

利用SAX解析XML文档牵涉到两个部分:解析器和事件处理器。

解析器负责读取XML文档,并向事件处理器发送事件,如元素开始跟元素结束事件;

而事件处理器则负责对事件作出相应,对传递的XML数据进行处理。

<psax适于处理下面的问题：< p="" style="color: rgb(51, 51, 51); font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, STHeiti, "Microsoft Yahei", sans-serif; font-size: 18px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: normal; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-decoration-color: initial;">

* 1、对大型文件进行处理；
* 2、只需要文件的部分内容，或者只需从文件中得到特定信息。
* 3、想建立自己的对象模型的时候。

在python中使用sax方式处理xml要先引入xml.sax中的parse函数，还有xml.sax.handler中的ContentHandler。

### **ContentHandler类方法介绍**

**characters(content)方法**

调用时机：

从行开始，遇到标签之前，存在字符，content的值为这些字符串。

从一个标签，遇到下一个标签之前， 存在字符，content的值为这些字符串。

从一个标签，遇到行结束符之前，存在字符，content的值为这些字符串。

标签可以是开始标签，也可以是结束标签。

**startDocument()方法**

文档启动的时候调用。

**endDocument()方法**

解析器到达文档结尾时调用。

**startElement(name, attrs)方法**

遇到XML开始标签时调用，name是标签的名字，attrs是标签的属性值字典。

**endElement(name)方法**

遇到XML结束标签时调用。

## make\_parser方法

以下方法创建一个新的解析器对象并返回。

xml.sax.make\_parser( [parser\_list] )

参数说明:

* **parser\_list** - 可选参数，解析器列表

## parser方法

以下方法创建一个 SAX 解析器并解析xml文档：

xml.sax.parse( xmlfile, contenthandler[, errorhandler])

参数说明:

* **xmlfile** - xml文件名
* **contenthandler** - 必须是一个ContentHandler的对象
* **errorhandler** - 如果指定该参数，errorhandler必须是一个SAX ErrorHandler对象

## parseString方法

parseString方法创建一个XML解析器并解析xml字符串：

xml.sax.parseString(xmlstring, contenthandler[, errorhandler])

参数说明:

* **xmlstring** - xml字符串
* **contenthandler** - 必须是一个ContentHandler的对象
* **errorhandler** - 如果指定该参数，errorhandler必须是一个SAX ErrorHandler对象

## Python 解析XML实例

#!/usr/bin/python3

import xml.sax

class MovieHandler( xml.sax.ContentHandler ):

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

self.CurrentData = ""

self.type = ""

self.format = ""

self.year = ""

self.rating = ""

self.stars = ""

self.description = ""

# 元素开始调用

def startElement(self, tag, attributes):

self.CurrentData = tag

if tag == "movie":

print ("\*\*\*\*\*Movie\*\*\*\*\*")

title = attributes["title"]

print ("Title:", title)

# 元素结束调用

def endElement(self, tag):

if self.CurrentData == "type":

print ("Type:", self.type)

elif self.CurrentData == "format":

print ("Format:", self.format)

elif self.CurrentData == "year":

print ("Year:", self.year)

elif self.CurrentData == "rating":

print ("Rating:", self.rating)

elif self.CurrentData == "stars":

print ("Stars:", self.stars)

elif self.CurrentData == "description":

print ("Description:", self.description)

self.CurrentData = ""

# 读取字符时调用

def characters(self, content):

if self.CurrentData == "type":

self.type = content

elif self.CurrentData == "format":

self.format = content

elif self.CurrentData == "year":

self.year = content

elif self.CurrentData == "rating":

self.rating = content

elif self.CurrentData == "stars":

self.stars = content

elif self.CurrentData == "description":

self.description = content

if ( \_\_name\_\_ == "\_\_main\_\_"):

# 创建一个 XMLReader

parser = xml.sax.make\_parser()

# turn off namepsaces

parser.setFeature(xml.sax.handler.feature\_namespaces, 0)

# 重写 ContextHandler

Handler = MovieHandler()

parser.setContentHandler( Handler )

parser.parse("movies.xml")

以上代码执行结果如下：

\*\*\*\*\*Movie\*\*\*\*\*Title: Enemy BehindType: War, ThrillerFormat: DVDYear: 2003Rating: PGStars: 10Description: Talk about a US-Japan war\*\*\*\*\*Movie\*\*\*\*\*Title: TransformersType: Anime, Science FictionFormat: DVDYear: 1989Rating: RStars: 8Description: A schientific fiction\*\*\*\*\*Movie\*\*\*\*\*Title: TrigunType: Anime, ActionFormat: DVDRating: PGStars: 10Description: Vash the Stampede!\*\*\*\*\*Movie\*\*\*\*\*Title: IshtarType: ComedyFormat: VHSRating: PGStars: 2Description: Viewable boredom

完整的 SAX API 文档请查阅[Python SAX APIs](http://docs.python.org/library/xml.sax.html" \t "http://www.runoob.com/python3/_blank)

## 使用xml.dom解析xml

文件对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展置标语言的标准编程接口。

一个 DOM 的解析器在解析一个 XML 文档时，一次性读取整个文档，把文档中所有元素保存在内存中的一个树结构里，之后你可以利用DOM 提供的不同的函数来读取或修改文档的内容和结构，也可以把修改过的内容写入xml文件。

python中用xml.dom.minidom来解析xml文件，实例如下：

#!/usr/bin/python3

from xml.dom.minidom import parseimport xml.dom.minidom

# 使用minidom解析器打开 XML 文档DOMTree = xml.dom.minidom.parse("movies.xml")

collection = DOMTree.documentElementif collection.hasAttribute("shelf"):

print ("Root element : %s" % collection.getAttribute("shelf"))

# 在集合中获取所有电影

movies = collection.getElementsByTagName("movie")

# 打印每部电影的详细信息for movie in movies:

print ("\*\*\*\*\*Movie\*\*\*\*\*")

if movie.hasAttribute("title"):

print ("Title: %s" % movie.getAttribute("title"))

type = movie.getElementsByTagName('type')[0]

print ("Type: %s" % type.childNodes[0].data)

format = movie.getElementsByTagName('format')[0]

print ("Format: %s" % format.childNodes[0].data)

rating = movie.getElementsByTagName('rating')[0]

print ("Rating: %s" % rating.childNodes[0].data)

description = movie.getElementsByTagName('description')[0]

print ("Description: %s" % description.childNodes[0].data)

以上程序执行结果如下：

Root element : New Arrivals\*\*\*\*\*Movie\*\*\*\*\*Title: Enemy BehindType: War, ThrillerFormat: DVDRating: PGDescription: Talk about a US-Japan war\*\*\*\*\*Movie\*\*\*\*\*Title: TransformersType: Anime, Science FictionFormat: DVDRating: RDescription: A schientific fiction\*\*\*\*\*Movie\*\*\*\*\*Title: TrigunType: Anime, ActionFormat: DVDRating: PGDescription: Vash the Stampede!\*\*\*\*\*Movie\*\*\*\*\*Title: IshtarType: ComedyFormat: VHSRating: PGDescription: Viewable boredom

# Python3 JSON 数据解析

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于ECMAScript的一个子集。

Python3 中可以使用 json 模块来对 JSON 数据进行编解码，它包含了两个函数：

* **json.dumps():**对数据进行编码。
* **json.loads():**对数据进行解码。

在json的编解码过程中，python 的原始类型与json类型会相互转换，具体的转化对照如下：

### **Python 编码为 JSON 类型转换对应表：**

| **Python** | **JSON** |
| --- | --- |
| dict | object |
| list, tuple | array |
| str | string |
| int, float, int- & float-derived Enums | number |
| True | true |
| False | false |
| None | null |

### **JSON 解码为 Python 类型转换对应表：**

| **JSON** | **Python** |
| --- | --- |
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

### **json.dumps 与 json.loads 实例**

以下实例演示了 Python 数据结构转换为JSON：

#!/usr/bin/python3

import json

# Python 字典类型转换为 JSON 对象

data = {

'no' : 1,

'name' : 'atguigu',

'url' : 'http://www.atguigu.com'}

json\_str = json.dumps(data)print ("Python 原始数据：", repr(data))print ("JSON 对象：", json\_str)

执行以上代码输出结果为：

Python 原始数据： {'url': 'http://www.atguigu.com', 'no': 1, 'name': 'atguigu'}

JSON 对象： {"url": "http://www.atguigu.com", "no": 1, "name": "atguigu"}

通过输出的结果可以看出，简单类型通过编码后跟其原始的repr()输出结果非常相似。

接着以上实例，我们可以将一个JSON编码的字符串转换回一个Python数据结构：

#!/usr/bin/python3

import json

# Python 字典类型转换为 JSON 对象

data1 = {

'no' : 1,

'name' : 'atguigu',

'url' : 'http://www.atguigu.com'}

json\_str = json.dumps(data1)print ("Python 原始数据：", repr(data1))print ("JSON 对象：", json\_str)

# 将 JSON 对象转换为 Python 字典

data2 = json.loads(json\_str)print ("data2['name']: ", data2['name'])print ("data2['url']: ", data2['url'])

执行以上代码输出结果为：

Python 原始数据： {'name': 'atguigu', 'no': 1, 'url': 'http://www.atguigu.com'}

JSON 对象： {"name": "atguigu", "no": 1, "url": "http://www.atguigu.com"}

data2['name']: atguigu

data2['url']: http://www.atguigu.com

如果你要处理的是文件而不是字符串，你可以使用 **json.dump()** 和 **json.load()** 来编码和解码JSON数据。例如：

# 写入 JSON 数据with open('data.json', 'w') as f:

json.dump(data, f)

# 读取数据with open('data.json', 'r') as f:

data = json.load(f)

# Python3 日期和时间

Python 程序能用很多方式处理日期和时间，转换日期格式是一个常见的功能。

Python 提供了一个 time 和 calendar 模块可以用于格式化日期和时间。

时间间隔是以秒为单位的浮点小数。

每个时间戳都以自从1970年1月1日午夜（历元）经过了多长时间来表示。

Python 的 time 模块下有很多函数可以转换常见日期格式。如函数time.time()用于获取当前时间戳, 如下实例:

#!/usr/bin/python3

import time; # 引入time模块

ticks = time.time()

print ("当前时间戳为:", ticks)

以上实例输出结果：

当前时间戳为: 1459996086.7115328

时间戳单位最适于做日期运算。但是1970年之前的日期就无法以此表示了。太遥远的日期也不行，UNIX和Windows只支持到2038年。

## 什么是时间元组？

很多Python函数用一个元组装起来的9组数字处理时间:

|  |  |  |
| --- | --- | --- |
| **序号** | **字段** | **值** |
| 0 | 4位数年 | 2008 |
| 1 | 月 | 1 到 12 |
| 2 | 日 | 1到31 |
| 3 | 小时 | 0到23 |
| 4 | 分钟 | 0到59 |
| 5 | 秒 | 0到61 (60或61 是闰秒) |
| 6 | 一周的第几日 | 0到6 (0是周一) |
| 7 | 一年的第几日 | 1到366 (儒略历) |
| 8 | 夏令时 | -1, 0, 1, -1是决定是否为夏令时的旗帜 |

上述也就是struct\_time元组。这种结构具有如下属性：

|  |  |  |
| --- | --- | --- |
| **序号** | **属性** | **值** |
| 0 | tm\_year | 2008 |
| 1 | tm\_mon | 1 到 12 |
| 2 | tm\_mday | 1 到 31 |
| 3 | tm\_hour | 0 到 23 |
| 4 | tm\_min | 0 到 59 |
| 5 | tm\_sec | 0 到 61 (60或61 是闰秒) |
| 6 | tm\_wday | 0到6 (0是周一) |
| 7 | tm\_yday | 一年中的第几天，1 到 366 |
| 8 | tm\_isdst | 是否为夏令时，值有：1(夏令时)、0(不是夏令时)、-1(未知)，默认 -1 |

## 获取当前时间

从返回浮点数的时间辍方式向时间元组转换，只要将浮点数传递给如localtime之类的函数。

#!/usr/bin/python3

import time

localtime = time.localtime(time.time())

print ("本地时间为 :", localtime)

以上实例输出结果：

本地时间为 : time.struct\_time(tm\_year=2016, tm\_mon=4, tm\_mday=7, tm\_hour=10, tm\_min=28, tm\_sec=49, tm\_wday=3, tm\_yday=98, tm\_isdst=0)

## 获取格式化的时间

你可以根据需求选取各种格式，但是最简单的获取可读的时间模式的函数是asctime():

#!/usr/bin/python3

import time

localtime = time.asctime( time.localtime(time.time()) )

print ("本地时间为 :", localtime)

以上实例输出结果：

本地时间为 : Thu Apr 7 10:29:13 2016

## 格式化日期

我们可以使用 time 模块的 strftime 方法来格式化日期，：

time.strftime(format[, t])

#!/usr/bin/python3

import time

# 格式化成2016-03-20 11:45:39形式

print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))

# 格式化成Sat Mar 28 22:24:24 2016形式

print (time.strftime("%a %b %d %H:%M:%S %Y", time.localtime()))

# 将格式字符串转换为时间戳

a = "Sat Mar 28 22:24:24 2016"

print (time.mktime(time.strptime(a,"%a %b %d %H:%M:%S %Y")))

以上实例输出结果：

2016-04-07 10:29:46

Thu Apr 07 10:29:46 2016

1459175064.0

python中时间日期格式化符号：

* %y 两位数的年份表示（00-99）
* %Y 四位数的年份表示（000-9999）
* %m 月份（01-12）
* %d 月内中的一天（0-31）
* %H 24小时制小时数（0-23）
* %I 12小时制小时数（01-12）
* %M 分钟数（00=59）
* %S 秒（00-59）
* %a 本地简化星期名称
* %A 本地完整星期名称
* %b 本地简化的月份名称
* %B 本地完整的月份名称
* %c 本地相应的日期表示和时间表示
* %j 年内的一天（001-366）
* %p 本地A.M.或P.M.的等价符
* %U 一年中的星期数（00-53）星期天为星期的开始
* %w 星期（0-6），星期天为星期的开始
* %W 一年中的星期数（00-53）星期一为星期的开始
* %x 本地相应的日期表示
* %X 本地相应的时间表示
* %Z 当前时区的名称
* %% %号本身

## 获取某月日历

Calendar模块有很广泛的方法用来处理年历和月历，例如打印某月的月历：

#!/usr/bin/python3

import calendar

cal = calendar.month(2016, 1)

print ("以下输出2016年1月份的日历:")

print (cal)

以上实例输出结果：

以下输出2016年1月份的日历:

January 2016

Mo Tu We Th Fr Sa Su

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

## Time 模块

Time 模块包含了以下内置函数，既有时间处理相的，也有转换时间格式的：

|  |  |  |
| --- | --- | --- |
| **序号** | **函数及描述** | **实例** |
| 1 | time.altzone 返回格林威治西部的夏令时地区的偏移秒数。如果该地区在格林威治东部会返回负值（如西欧，包括英国）。对夏令时启用地区才能使用。 | 以下实例展示了 altzone()函数的使用方法：  >>> import time  >>> print ("time.altzone %d " % time.altzone)  time.altzone -28800 |
| 2 | time.asctime([tupletime]) 接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2008"（2008年12月11日 周二18时07分14秒）的24个字符的字符串。 | 以下实例展示了 asctime()函数的使用方法：  >>> import time  >>> t = time.localtime()  >>> print ("time.asctime(t): %s " % time.asctime(t))  time.asctime(t): Thu Apr 7 10:36:20 2016 |
| 3 | [time.clock()](http://www.runoob.com/python3/python3-att-time-clock.html" \t "http://www.runoob.com/python3/_blank) 用以浮点数计算的秒数返回当前的CPU时间。用来衡量不同程序的耗时，比time.time()更有用。 | [实例](http://www.runoob.com/python3/python3-att-time-clock.html" \t "http://www.runoob.com/python3/_blank) |
| 4 | time.ctime([secs]) 作用相当于asctime(localtime(secs))，未给参数相当于asctime() | 以下实例展示了 ctime()函数的使用方法：  >>> import time  >>> print ("time.ctime() : %s" % time.ctime())  time.ctime() : Thu Apr 7 10:51:58 2016 |
| 5 | time.gmtime([secs]) 接收时间辍（1970纪元后经过的浮点秒数）并返回格林威治天文时间下的时间元组t。注：t.tm\_isdst始终为0 | 以下实例展示了 gmtime()函数的使用方法：  >>> import time  >>> print ("gmtime :", time.gmtime(1455508609.34375))  gmtime : time.struct\_time(tm\_year=2016, tm\_mon=2, tm\_mday=15, tm\_hour=3, tm\_min=56, tm\_sec=49, tm\_wday=0, tm\_yday=46, tm\_isdst=0) |
| 6 | time.localtime([secs] 接收时间辍（1970纪元后经过的浮点秒数）并返回当地时间下的时间元组t（t.tm\_isdst可取0或1，取决于当地当时是不是夏令时）。 | 以下实例展示了 localtime()函数的使用方法：  >>> import time  >>> print ("localtime(): ", time.localtime(1455508609.34375))  localtime(): time.struct\_time(tm\_year=2016, tm\_mon=2, tm\_mday=15, tm\_hour=11, tm\_min=56, tm\_sec=49, tm\_wday=0, tm\_yday=46, tm\_isdst=0) |
| 7 | [time.mktime(tupletime)](http://www.runoob.com/python3/python3-att-time-mktime.html" \t "http://www.runoob.com/python3/_blank) 接受时间元组并返回时间辍（1970纪元后经过的浮点秒数）。 | [实例](http://www.runoob.com/python3/python3-att-time-mktime.html" \t "http://www.runoob.com/python3/_blank) |
| 8 | time.sleep(secs) 推迟调用线程的运行，secs指秒数。 | 以下实例展示了 sleep()函数的使用方法：  #!/usr/bin/python3  import time  print ("Start : %s" % time.ctime())  time.sleep( 5 )  print ("End : %s" % time.ctime()) |
| 9 | time.strftime(fmt[,tupletime]) 接收以时间元组，并返回以可读字符串表示的当地时间，格式由fmt决定。 | 以下实例展示了 strftime()函数的使用方法：  >>> import time  >>> print (time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))  2016-04-07 11:18:05 |
| 10 | time.strptime(str,fmt='%a %b %d %H:%M:%S %Y') 根据fmt的格式把一个时间字符串解析为时间元组。 | 以下实例展示了 strftime()函数的使用方法：  >>> import time  >>> struct\_time = time.strptime("30 Nov 00", "%d %b %y")  >>> print ("返回元组: ", struct\_time)  返回元组: time.struct\_time(tm\_year=2000, tm\_mon=11, tm\_mday=30, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=335, tm\_isdst=-1) |
| 11 | time.time( ) 返回当前时间的时间戳（1970纪元后经过的浮点秒数）。 | 以下实例展示了 time()函数的使用方法：  >>> import time  >>> print(time.time())  1459999336.1963577 |
| 12 | [time.tzset()](http://www.runoob.com/python3/python3-att-time-tzset.html" \t "http://www.runoob.com/python3/_blank) 根据环境变量TZ重新初始化时间相关设置。 | [实例](http://www.runoob.com/python3/python3-att-time-tzset.html" \t "http://www.runoob.com/python3/_blank) |

Time模块包含了以下2个非常重要的属性：

|  |  |
| --- | --- |
| **序号** | **属性及描述** |
| 1 | **time.timezone** 属性time.timezone是当地时区（未启动夏令时）距离格林威治的偏移秒数（>0，美洲;<=0大部分欧洲，亚洲，非洲）。 |
| 2 | **time.tzname** 属性time.tzname包含一对根据情况的不同而不同的字符串，分别是带夏令时的本地时区名称，和不带的。 |

## 日历（Calendar）模块

此模块的函数都是日历相关的，例如打印某月的字符月历。

星期一是默认的每周第一天，星期天是默认的最后一天。更改设置需调用calendar.setfirstweekday()函数。模块包含了以下内置函数：

|  |  |
| --- | --- |
| **序号** | **函数及描述** |
| 1 | **calendar.calendar(year,w=2,l=1,c=6)** 返回一个多行字符串格式的year年年历，3个月一行，间隔距离为c。 每日宽度间隔为w字符。每行长度为21\* W+18+2\* C。l是每星期行数。 |
| 2 | **calendar.firstweekday( )** 返回当前每周起始日期的设置。默认情况下，首次载入caendar模块时返回0，即星期一。 |
| 3 | **calendar.isleap(year)** 是闰年返回True，否则为false。 |
| 4 | **calendar.leapdays(y1,y2)** 返回在Y1，Y2两年之间的闰年总数。 |
| 5 | **calendar.month(year,month,w=2,l=1)** 返回一个多行字符串格式的year年month月日历，两行标题，一周一行。每日宽度间隔为w字符。每行的长度为7\* w+6。l是每星期的行数。 |
| 6 | **calendar.monthcalendar(year,month)** 返回一个整数的单层嵌套列表。每个子列表装载代表一个星期的整数。Year年month月外的日期都设为0;范围内的日子都由该月第几日表示，从1开始。 |
| 7 | **calendar.monthrange(year,month)** 返回两个整数。第一个是该月的星期几的日期码，第二个是该月的日期码。日从0（星期一）到6（星期日）;月从1到12。 |
| 8 | **calendar.prcal(year,w=2,l=1,c=6)** 相当于 print calendar.calendar(year,w,l,c). |
| 9 | **calendar.prmonth(year,month,w=2,l=1)** 相当于 print calendar.calendar（year，w，l，c）。 |
| 10 | **calendar.setfirstweekday(weekday)** 设置每周的起始日期码。0（星期一）到6（星期日）。 |
| 11 | **calendar.timegm(tupletime)** 和time.gmtime相反：接受一个时间元组形式，返回该时刻的时间辍（1970纪元后经过的浮点秒数）。 |
| 12 | **calendar.weekday(year,month,day)** 返回给定日期的日期码。0（星期一）到6（星期日）。月份为 1（一月） 到 12（12月）。 |

## 其他相关模块和函数

在Python中，其他处理日期和时间的模块还有：

* [time 模块](https://docs.python.org/3/library/time.html" \t "http://www.runoob.com/python3/_blank)
* [datetime模块](https://docs.python.org/3/library/datetime.html" \t "http://www.runoob.com/python3/_blank)

# Python 输出指定范围内的素数

[IMG_256 Python3 实例](http://www.runoob.com/python3/python3-examples.html)

素数（prime number）又称质数，有无限个。除了1和它本身以外不再被其他的除数整除。

以下实例可以输出指定范围内的素数：

#!/usr/bin/python3

# 输出指定范围内的素数

# take input from the user

lower = int(input("输入区间最小值: "))

upper = int(input("输入区间最大值: "))

for num in range(lower,upper + 1):

# 素数大于 1

if num > 1:

for i in range(2,num):

if (num % i) == 0:

break

else:

print(num)

执行以上程序，输出结果为：

$ python3 test.py

输入区间最小值: 1

输入区间最大值: 100

2

3

5

7

11

13

17

19

23

29

31

37

41

43

47

53

59

61

67

71

73

79

83

89

97

[IMG_257 Python3 实例](http://www.runoob.com/python3/python3-examples.html)