UNIX 操作系统及应用
附录 E — Python脚本编程语言

唐晓晟　　李亦农

Contents

1 概述

Figure 1: Python Logo

- **Python**是一种面向对象、解释型编程语言，具有 20 余年发展历史，成熟稳定

- Python 包含了一组完善而且容易理解的标准库，能够轻松完成很多常见的任务。其语法简洁清晰，尽量使用无异议的英语单词，与其他大多数程序设计语言使用大括号不一样，它使用缩进来定义语句块
- 与 Scheme、Ruby、Perl、Tcl 等脚本语言一样，Python 具备垃圾回收功能，能够自动管理内存使用，它经常被用于任务管理和网络应用开发，其实它也非常适合完成更高级的任务
- Python 在所有的操作系统中都有相应的解释器，使用诸如py2exe、PyPy、PyInstaller之类的工具可以将 Python 代码转换为可以脱离 Python 解释器执行的程序
- Python 的官方解释器是 CPython，该解释器使用 C 语言编写，是一个由社区驱动的自由软件，目前由 Python 软件基金会管理
- Python 支持面向过程编程、面向对象编程、函数式编程、面向侧面编程、泛编程等多种编程范式

2 历史

- Python 的创始人为吉多. 范罗苏姆 (Guido van Rossum)，1989 年圣诞节期间，他为了在阿姆斯特丹打发时间，决定开发一个新的脚本解释语言，做为 ABC 语言的一种继承，其第一个实现是在 Mac 机上。对了，作者是一个吉他手。
- 可以说，Python 是由 ABC 发展而来，主要是受到了 Modula-3 的影响，并且结合了 Unix shell 和 C 的习惯
- Python 2.0 于 2000 年 10 月 16 日发布，实现完整的垃圾回收，并且支持 Unicode，同时，整个开发过程更加透明，社区对开发进度的影响逐渐扩大
- Python 3.0 于 2008 年 12 月 3 日发布，此版本不完全兼容之前的版本，不过，很多新特性也被移植到旧的 Python 2.6/2.7 版本中
- Python 全面支持面向对象编程，函数、模块、数字、字符串都是对象，并且完全支持继承、重载、派生、多继承。Python 支持运算符重载，因此，也支持泛型设计。函数式编程方面，Python 提供了 functools 和 itertools 予以支持
- Python 被应用于很多大型软件开发，如 Zope、Mnet 及 BitTorrent 等，Google 内部也广泛地使用它。相比其他脚本语言如 shell script、VBScript 等，Python 功能要强大很多
- Python 本身被设计为可扩充的，并非所有特性和功能都集成到语言核心。Python 提供了丰富的 API 和工具，以便程序员能够轻松地使用 C、C++、Cython 等来编写模块对其进行功能扩充
- Python 编译器本身也可以被集成到其他需要脚本语言的程序中，因此，Python 很多情况下也被当成一种胶水语言（Glue Language）使用，将其他语言编写的程序进行继承和封装，Google 公司 2004 年起内部开始使用 Python，其口号是 Python where we can, C++ where we must

3 设计哲学

- Python 的设计哲学是优雅、明确、简单
  - Perl 中 “总是有很多种方法来做同一件事情” 的理念在 Python 中通常是难以忍受的
  - Python 希望 “是用一种方法，最好是只有一种方法” 来做一件事
  - 拒绝花哨的语法，其代码可读性通常比 Perl 好很多
  - 在 Python 解释器内运行 import this 可获得完整的列表
- 尽量避开不成熟或者不重要的优化
  - 非重要部位的加快运行速度的补丁通常不会合并到 Python 内
  - 二八原则，绝大多数程序对速度要求不高，当有特别要求时，可使用 JIT 技术（如 PyPy），或者使用 C/C++ 重写相关代码

4 应用范围

- Web 开发
  - 定义 WSGI 应用接口，通过mod_wsgi模块，配合 Apache 使用
  - Python 编写的 Gunicom 可做为 Web 服务器
  - 提供了很多优秀的框架，如 Django、TurboGears、web2py、Zope、flask 等，方面开发者轻松、快速开发复杂的 Web 应用
  - 各种网络协议的完善支持，提供了 Twisted 和 gevent 等库，被广泛应用于高性能网络服务和应用的开发
- GUI 开发
  - 相比 C++，其应用的开发更快更容易
  - 提供 Tkinter 库，支持简单的 GUI 开发
  - wxPython 和 PyQt 可以被应用于开发跨平台的桌面软件，与用户的桌面环境相契合
  - 通过 PyInstaller 可将程序发布为独立的安装程序包
- 操作系统
  - Python 是很多操作系统的标准安装组件
  - 一些 Linux 发行版的安装器使用 Python 编写，如 Ubuntu 的 Ubiquity 安装器、Red Hat 和 Fedora 的 Anaconda 安装器、Gentoo 的 Portage 包管理器等
  - 通过 pywin32，Python 能访问 Windows 的 COM 及其他 Windows API，通过 IronPython，Python 能够直接调用.Net Framework
  - Python 编写的系统管理脚本在可读性、性能、源程序重用性、扩展性方面都优于普通 shell 脚本
- 其他
  - NumPy、SciPy、Matplotlib 可用于编写科学计算程序
  - 一些公司可能会使用 Scons 代替make来构建 C++ 程序
  - 很多游戏使用 C++ 编写图形显示等高性能模块，使用 Python 或者 Lua 来编写游戏的逻辑和服务器
  - Youtube、Google、Yahoo!、NASA 等都在内部大量使用 Python，支持 OLPC 处理的 Sugar 项目的大多数软件都使用 Python 编写

5 Hello world

```
1 #!/usr/local/bin/python
2
3 print ("Hello, world!") # for python 3.0
4 print "Hello, world!"   # for python 2.x
```

- 第一行的仍然是环境指示器（或者说是语法指示器）
- #开头的仍然是注释
- 将以上代码存储为 hello.py
  - 可以使用 python hello.py 解释执行
  - 亦可以 chmod +x hello.py 添加可执行权限，然后./hello.py 执行
- 运行 python 后，可在>>>提示符下输入需要执行的命令，按 Enter 键输出结果

6 语法

- 缩排
  - Python 开发者有意让违反了缩排规则的程序不能通过编译，以此来强制程序员养成良好的编程习惯，并且 Python 语言利用缩排表示语句块的开始和结束（Off-side 规则 ）
  - 缩排成为了语法的一部分

```
if age < 18:
    print "你不能买酒"
    print "你可以买口香糖"
print "这句话在if语句块外面"
```

  - PEP（Python Enhancement Proposals）规定，必须使用 4 个空格来表示每级缩排

- 语句和控制流

`if` 条件成立时执行语句块，经常与`else`、`elif`配合使用

`for` 遍历列表、字符串、字典、集合等容器，依次处理容器中的每个元素

`while` 当条件为真时，循环执行语句块

`try` 与`exception`、`finally`、`else`配合使用，处理程序中出现的异常情况

`class` 定义类

`def` 定义函数和类型的方法

`pass` 表示此行为空，不执行任何操作

`assert` 用于程序调试阶段时测试执行条件是否满足

`with` 在一个场景中执行某个语句块

`yield` 迭代函数内使用，用于返回一个元素，Python 2.5 后，这个语句变成一个运算符

`raise` 抛出一个异常

`import` 导入一个模块或包，常用写法

```
from module import name
import module as name
from module import name as anothername
```

- 表达式
  - 主要算术运算符与 C/C++ 类似，包括`+`，`-`，`*`，`/`，`//`（整除），`**`，（取补），`%`，以及`>>`，`<<`，`&`，`|`，`^`，`>`，`<`，`==`，`!=`，`<=`，`>=`，其中，`|`，`^`，`&`，`<<`，`>>`必须应用于整数
  - `and`，`or`，`not`表示逻辑运算
  - `is`，`is not`用于比较两个变量是否是同一对象
  - `in`，`not in`用于判断一个对象是否属于另外一个对象
  - 支持字典、集合、列表的推导式 (dict/set/list comprehension)，如

```
>>> [x+3 for x in range(4)]
[3, 4, 5, 6]
>>> {x+3 for x in range(4)}
{3, 4, 5, 6}
>>> {x: x+3 for x in range(4)}
{0:3, 1:4, 2:5, 3:6}
```

  - 支持“迭代表达式”（generator comprehension），如计算 0-9 的平方和

```
>>> sum(x*x for x in range(10))
285
```

- 使用lambda表示匿名函数，如

```
>>> add = lambda x,y : x + y
>>> add(3,2)
5
```

- 使用`y if cond else x`表示条件表达式
- 区分**列表**（list）和**元组**（tuple）两种类型，list 的写法是`[1,2,3]`，而 tuple 的写法是`(1,2,3)`，可以改变 list 中的元素，但不能修改 tuple
- 可使用单引号和双引号来表示字符串，两种符号作用相同，字符串中的\表示转义，表达式前面加r指示 Python 不解释字符串中的\，通常用于正则表达式或 Windows 文件路径名
- 支持列表切割，可以获得完整列表中的一部分，支持切割操作的类型有 str，bytes，list，tuple 等，其语法是`[left:right]`或者`[left:right:stride]`，假如nums变量的值是`[1,3,5,7,8,13,20]`，则
  - `nums[2:5] == [5,7,8]`，不包括最后一个
  - `nums[1:] == [3,5,7,8,13,20]`，切割到最后
  - `nums[:-3] == [1,3,5,7]`，最开始到倒数第 3 个
  - `nums[:]` == 全部元素
  - `nums[1:5:2] == [3,7]`，步长为 2

- 函数
  - 支持递归、默认参数、可变参数，但不支持函数重载
  - 为增强代码的可读性，可以为函数撰写“文档字符串”（Documentation Strings, 或 docstrings），用于解释函数的作用、参数的类型与意义、返回值类型与取值范围等，可用内置函数 help() 打印出函数的使用帮助

```
>>> def randint(a,b):
... "Return random integer in range [a,b]"
...
>>> help(randint)
Help on function randint in module __main__:

randint(a,b)
  Return random integer in range [a,b]
```

- 面向对象开发
  - 对于对象的方法，调用语法是instance.method(arguments)，等价于Class.method(instance,arguments)
  - 定义对象方法时，必须显式地定义第一个参数，一般该参数名都使用 self，相当于 this

```
class Fish:
    def eat(self, food):
        if food is not None:
            self.hungry = False
class User:
    def __init__(myself, name):
        myself.name = name
#构造Fish的实例
f = Fish()
#以下两种调用形式是等价的
Fish.eat(f, "earthworm")
f.eat("earthworm")

u = User('username')
u.name
```

  - Python 认识一些以”__” 并以”__” 结束的特殊方法名，它们用于运算符重载和实现多种特殊功能

- 数据类型 & 动态类型

  - Python 采用动态类型系统
  - 编译时，Python 不会检查对象是否拥有被调用的方法或属性，直至运行时，才做出检查
  - Python 允许程序员定义类型，类型本身也是特殊类型 type 的对象，这种特殊的设计允许对类型进行反射编程
  - Python 内置丰富的数据类型，与 Java、C++ 相比，这些数据类型有效地减少代码的长度，如下：

    str 由字符组成的不可更改的有序列，如`'Wikipedia', "Wikipedia"`

    bytes 由字节组成的不可更改的有序列，如`b'Some ASCII'`

    list 可以包含多种类型的可改变的有序列，如`[4.0, 'string', True]`

    tuple 可以包含多种类型的不可改变的有序列，如`(4.0, 'string', True)`

    set 与数学中集合的概念类型，无序、无重复元素，如`{4.0, 'sting', True}`

    dict 可改变的由键值对组成的无序列，如`{'key1':1.0, 3:False}`

    int 精度不限的整数

    float 浮点数，精度与系统相关

    complex 复数，如`3+2i`

    bool 逻辑值，只有 True 或 False
  - 此外，Python 还用类型来表示函数、模块、类型本身、对象的方法、编译后的 Python 代码、运行时信息等

- 数学运算
  - Python 使用与 C、Java 类似的运算符，支持整数与浮点数的数学运算，同时支持复数与无穷位数（实际受限于计算机的能力）的整数运算
  - 大多数数学函数处于 math（实数）和 cmath（复数）模块内

```
>>> import math
>>> print(math.sin(math.pi/2))
1.0
```

  - fractions 模块用于支持分数运算，decimal 模块用于支持高精度的浮点数运算

7 标准库

- Python 语言的核心只包括数字、字符串、列表、字典、文件等常见类型和函数，标准库提供了各种用户所需要的功能，包括：

  文本处理 包含文本格式化、正则表达式匹配、文本差异计算与合并、Unicode 支持、二进制数据处理等

  文件处理 包括文件操作、创建临时文件、文件压缩与归档、访问配置文件等功能

  操作系统相关 包括线程与进程支持、IO 复用、日期与时间处理、调用系统函数、日志等功能

  网络通信 包括网络套接字、SSL 加密通信、异步网络通信等

  网络协议 支持 HTTP、FTP、SMTP、POP、IMAP、NNTP、XMLRPC 等多种网络协议，并提供了编写网络服务器的框架

  W3C 支持 包括 HTML、SGML、XML 的处理

  其他功能 包括国际化支持、数学计算、HASH、Tkinter 等

- Python 社区提供了大量的第三方模块，使用方式与标准库类似，功能无所不包，覆盖科学计算、web 开发、数据库接口、图形系统多个领域，并且大多成熟而稳定
  - 第三方模块可使用 Python 或 C 编写
  - SWIG 常用于将 C 语言编写的程序库转化成 Python 模块
  - Boost C++ Libraries 包括了一组函数库（Boost.Python），使得以 Python 或者 C++ 编写的程序能互相调用
  - Python 已成为一种强大的应用于其他语言与工具之间的 “胶水” 语言

8 Python 3.0

- Python 3.0，常被称为 Python 3000 或 Py3k
- 这是一个较大的升级，没有考虑向下兼容，Python 2.6 是一个过渡版本，允许使用部分 3.0 的语法与函数
- 一个 2to3 的转换工具可以将正常运行于 Python 2.6 且无警告的代码无缝迁移到 Python 3.0
- Python 2.7 是 Python 最后一个 2.x 版本
- 请上网查阅 Python 2.x 和 3.x 之间的差异

9 开发环境

- 通用 IDE
  - eclipse + pydev 插件
  - emacs + 插件
  - NetBeans + 插件
  - SlickEdit
  - TextMate
  - Vim + 插件
  - Sublime Text + 插件
  - EditPlus
  - UltraEdit
  - PSPad
  - PyCharm，jetbrains 出品
- 专用 IDE

  Eric 基于 PyQt 的自由软件

  IDLE Python“标准”IDE，一般随 Python 而安装，编辑功能较少、调试功能也较弱

  Komodo 亦可用于 PHP、Ruby、Javascript、Perl、Web 和云开发

  PyCharm 由 JetBrains 打造，其 Java IDE 软件 IntelliJ 拥有海量的使用者

  PythonWin 包含在 pywin32 内的编辑器，仅适用于 Windows

  SPE Stani’s Python Editor，功能较多的免费软件，依赖 wxPython

  Ulipad 功能较全的免费软件，依赖 wxPython

  WingIDE 可能是功能最全的 IDE，不免费

  PyScripter 功能较全的开源 IDE，使用 Delphi 开发

10 使用 Python 编写的著名应用

Reddit 社交分享网站

Dropbox 文件分享服务

Django 鼓励快速开发的 Web 应用框架

Pylons Web 应用框架

Zope 应用服务器

TurboGears 另一个 Web 应用快速开发框架

Twisted Python 的网络应用程序框架

Fabric 用于管理成百上千台 Linux 主机的程序库

MoinMoinWiki Python 写成的 Wiki 程序

Trac 使用 Python 编写的 BUG 管理程序

Mailman 使用 Python 编写的邮件列表软件

Mezzanine 基于 Django 编写的内容管理系统

flask Python 微 Web 框架

Webpy 同上

Bottle 同上

EVE 应用于网络游戏

Blender 以 C 与 Python 开发的开源 3D 绘图软件

11 参考文献

1. Guido van Rossum. Foreword for "Programming Python" (1st ed.) 1996/5/1 （英文）
2. Python Language Guide (v1.0) Google Documents List Data API v1.0. Google. （原始内容存档于 10 August 2010）.
3. Heavy usage of Python at Google
4. Przemyslaw Piotrowski, Build a Rapid Web Development Environment for Python Server Pages and Oracle, Oracle Technology Network, July 2006. Accessed October 21, 2008.
5. Python Software Foundation. Python 3.0 Release. 2008/8/20 [2008/8/30] （英文）
6. Python Software Foundation. Conversion tool for Python 2.x code: 2to3 [2008/8/30] （英文）
7. Python Software Foundation, Should I use Python 2 or Python 3 for my development activity? 2010/9/14 （英文）
8. Guido van Rossum, What's New in Python 3.0. Python Software Foundation. 2009/2/14 [2011/2/22] （英文）.
9. http://www.Python.org, CPython

12 代码示例

- 表达式

```
#!/usr/bin/env python
#Filename: expression.py

length = 5
breadth = 2
area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

- 变量

```
#!/usr/bin/env python
#Filename: var1.py

i = 5
print i
i = i + 1
print i

s = '''This is a multi-line string.
This is the second line.'''
print s
```

- if语句

```
#!/usr/bin/env python
#Filename: if.py

number = 23
guess = int(raw_input('Enter an integer : '))

if guess == number:
    print 'Congratulations, you guessed it.' #New block starts here
    print "(but you do not win any prizes!)" #New block ends here
elif guess < number:
    print 'No, it is a little higher than that' # Another block
    # You can do whatever you want in a block ...
else:
    print 'No, it is a little lower than that'
    # you must have guess > number to reach here

print 'Done'
# This last statement is always executed,
# after the if statement is executed
```

- while语句

```
#!/usr/bin/env python
#Filename: while.py

import random

number = random.randint(0,1024)
running = True

while running:
    guess = int(raw_input('Enter an integer : '))

    if guess == number:
        print 'Congratulations, you guessed it.'
        running = False # this causes the while loop to stop
    elif guess < number:
        print 'No, try a big one'
    else:
        print 'No, try a small one'
else:
    print 'The while loop is over.'
    # Do anything else you want to do here
```

```
22
23 print 'Done'
```

- for语句

```
1 #!/usr/bin/env python
2 #Filename: for.py
3
4 for i in range(1, 5):
5     print i
6 else:
7     print 'The for loop is over'
```

- break语句

```
1 #!/usr/bin/env python
2 #Filename: break.py
3
4 while True:
5     s = raw_input('Enter something : ')
6     if s == 'quit':
7         break
8     print 'Length of the string is', len(s)
9 print 'Done'
```

- continue语句

```
1 #!/usr/bin/env python
2 # Filename: continue.py
3
4 while True:
5     s = raw_input('Enter something : ')
6     if s == 'quit':
7         break
8     if len(s) < 3:
9         continue
10    print 'Input is of sufficient length'
11    # Do other kinds of processing here...
```

- 定义函数

```
1 #!/usr/bin/env python
2 #Filename: function1.py
3
4 def sayHello():
5   print 'Hello, world!' # block beloing to the function
6
7 sayHello() # call the function
```

- 使用函数形参

```
1 #!/usr/bin/env python
2 #Filename: func_param.py
3
4 def printMax(a, b):
5     if a > b:
6         print a, 'is maximum'
7     else:
8         print b, 'is maximum'
9
10 printMax(3, 4) # directly give literal values
11
12 x = 5
13 y = 7
14
15 printMax(x, y) # give variables as arguments
```

- 使用局部变量

```
#!/usr/bin/env python
#Filename: func_local.py

def func(x):
    print 'x is', x
    x = 2
    print 'Changed local x to', x

x = 50
func(x)
print 'x is still', x
```

- 使用全局变量

```
#!/usr/bin/env python
#Filename: func_global.py

def func():
    global x

    print 'x is', x
    x = 2
    print 'Changed local x to', x

x = 50
func()
print 'Value of x is', x
```

- 使用默认参数

```
#!/usr/bin/env python
#Filename: func_default.py

def say(message, times = 1):
    print message * times

say('Hello')
say('World', 5)
```

- 使用关键参数

```
#!/usr/bin/env python
#Filename: func_key.py

def func(a, b=5, c=10):
    print 'a is', a, 'and b is', b, 'and c is', c

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

- 使用 return

```
#!/usr/bin/env python
#Filename: func_return.py

def maximum(x, y):
    if x > y:
        return x
    else:
        return y

print maximum(2, 3)
```

- 使用 DocStrings

```
#!/usr/bin/env python
#Filename: func_doc.py

```

```
def printMax(x, y):
    '''Prints the maximum of two numbers.

    The two values must be integers.'''
    x = int(x) # convert to integers, if possible
    y = int(y)

    if x > y:
        print x, 'is maximum'
    else:
        print y, 'is maximum'

printMax(3, 5)
print printMax.__doc__
```

- 使用 sys 模块

```
#!/usr/bin/env python
#Filename: using_sys.py

import sys

print 'The command line arguments are:'
for i in sys.argv:
    print i

print '\n\nThe PYTHONPATH is', sys.path, '\n'
```

- 模块的__name__属性

```
#!/usr/bin/env python
#Filename: using_name.py

if __name__ == '__main__':
    print 'This program is being run by itself'
else:
    print 'I am being imported from another module'

# try
# 1) ./using_name.py
# 2) python
#    >>> import using_name
```

- 创建自己的模块

```
#!/usr/bin/env python
#Filename: mymodule.py

def sayhi():
    print 'Hi, this is mymodule speaking.'

version = '0.1'

# End of mymodule.py
```

```
#!/usr/bin/env python
#Filename: mymodule_demo1.py

import mymodule

mymodule.sayhi()

print 'Version', mymodule.version
```

```
#!/usr/bin/env python
#Filename: mymodule_demo2.py

from mymodule import sayhi, version

#Alternative:
#from mymodule import *
```

```
8
9  sayhi()
10
11 print 'Version', version
```

- 使用列表

```
1  #!/usr/bin/env python
2  #Filename: using_list.py
3
4  # This is my shopping list
5  shoplist = ['apple', 'mango', 'carrot', 'banana']
6
7  print 'I have', len(shoplist),'items to purchase.'
8
9  print 'These items are:', # Notice the comma at end of the line
10 for item in shoplist:
11     print item,
12
13 print '\nI also have to buy rice.'
14 shoplist.append('rice')
15 print 'My shopping list is now', shoplist
16
17 print 'I will sort my list now'
18 shoplist.sort()
19 print 'Sorted shopping list is', shoplist
20
21 print 'The first item I will buy is', shoplist[0]
22 olditem = shoplist[0]
23 del shoplist[0]
24 print 'I bought the', olditem
25 print 'My shopping list is now', shoplist
```

- 使用元组

```
1  #!/usr/bin/env python
2  #Filename: using_tuple.py
3
4  zoo = ('wolf', 'elephant', 'penguin')
5  print 'Number of animals in the zoo is', len(zoo)
6
7  new_zoo = ('monkey', 'dolphin', zoo)
8  print 'Number of animals in the new zoo is', len(new_zoo)
9  print 'All animals in new zoo are', new_zoo
10 print 'Animals brought from old zoo are', new_zoo[2]
11 print 'Last animal brought from old zoo is', new_zoo[2][2]
```

- 打印元组

```
1  #!/usr/bin/env python
2  #Filename: print_tuple.py
3
4  age = 22
5  name = 'Swaroop'
6
7  print '%s is %d years old' % (name, age)
8  print 'Why is %s playing with that python?' % name
```

- 使用字典

```
1  #!/usr/bin/env python
2  #Filename: using_dict.py
3
4  # 'ab' is short for 'a'ddress'b'ook
5
6  ab = {       'Swaroop'   : 'swaroopch@byteofpython.info',
7               'Larry'     : 'larry@wall.org',
8               'Matsumoto' : 'matz@ruby-lang.org',
9               'Spammer'   : 'spammer@hotmail.com'
10     }
11
```

```
print "Swaroop's address is %s" % ab['Swaroop']

# Adding a key/value pair
ab['Guido'] = 'guido@python.org'

# Deleting a key/value pair
del ab['Spammer']

print '\nThere are %d contacts in the address-book\n' % len(ab)
for name, address in ab.items():
    print 'Contact %s at %s' % (name, address)

if 'Guido' in ab: # OR ab.has_key('Guido')
    print "\nGuido's address is %s" % ab['Guido']
```

- 使用序列

```
#!/usr/bin/env python
#Filename: seq.py

shoplist = ['apple', 'mango', 'carrot', 'banana']

# Indexing or 'Subscription' operation
print 'Item 0 is', shoplist[0]
print 'Item 1 is', shoplist[1]
print 'Item 2 is', shoplist[2]
print 'Item 3 is', shoplist[3]
print 'Item -1 is', shoplist[-1]
print 'Item -2 is', shoplist[-2]

# Slicing on a list
print 'Item 1 to 3 is', shoplist[1:3]
print 'Item 2 to end is', shoplist[2:]
print 'Item 1 to -1 is', shoplist[1:-1]
print 'Item start to end is', shoplist[:]

# Slicing on a string
name = 'swaroop'
print 'characters 1 to 3 is', name[1:3]
print 'characters 2 to end is', name[2:]
print 'characters 1 to -1 is', name[1:-1]
print 'characters start to end is', name[:]
```

- 对象与参考

```
#!/usr/bin/env python
#Filename: reference.py

print 'Simple Assignment'
shoplist = ['apple', 'mango', 'carrot', 'banana']
# mylist is just another name pointing to the same object!
mylist = shoplist

del shoplist[0]

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that both shoplist and mylist both print the same
# list without the 'apple' confirming that they point to
# the same object

print 'Copy by making a full slice'
mylist = shoplist[:] # make a copy by doing a full slice
del mylist[0] # remove first item

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that now the two lists are different
```

- 更多字符串方法

```
#!/usr/bin/env python
#Filename: str_methods.py
```

```
3
4  name = 'Swaroop' # This is a string object
5
6  if name.startswith('Swa'):
7      print 'Yes, the string starts with "Swa"'
8
9  if 'a' in name:
10     print 'Yes, it contains the string "a"'
11
12 if name.find('war') != -1:
13     print 'Yes, it contains the string "war"'
14
15 delimiter = '_*_'
16 mylist = ['Brazil', 'Russia', 'India', 'China']
17 print delimiter.join(mylist)
```

- 创建类

```
1  #!/usr/bin/env python
2  #Filename: simplestclass.py
3
4  class Person:
5    pass # An empty block
6
7  p=Person()
8  print p
```

- 使用对象的方法

```
1  #!/usr/bin/env python
2  # Filename: method.py
3
4  class Person:
5    def sayHi(self):
6      print 'Hello, how are you?'
7
8  p=Person()
9  p.sayHi()
10
11 # This short example can also be written as Person().sayHi()
```

- 使用__init__方法

```
1  #!/usr/bin/env python
2  #Filename: class_init.py
3
4  class Person:
5    def __init__(self,name):
6      self.name=name
7    def sayHi(self):
8      print 'Hello, my name is',self.name
9
10 p=Person('Swaroop')
11 p.sayHi()
12
13 # This short example can also be written
14 # as Person('Swaroop').sayHi()
```

- 使用类与对象的变量

```
1  #!/usr/bin/env python
2  #Filename: objvar.py
3
4  class Person:
5    '''Represents a person.'''
6    population=0
7
8    def __init__(self,name):
9      '''Initializes the person's data.'''
10     self.name=name
11     print '(Initializing %s)' %self.name
```

```
12
13      #When this person is created, he/she adds to the population
14      Person.population+=1
15
16    def __del__(self):
17      '''I am dying.'''
18      print '%s says bye.' %self.name
19
20      Person.population-=1
21
22      if Person.population==0:
23        print 'I am the last one.'
24      else:
25        print 'There are still %d people left.' %Person.population
26
27    def sayHi(self):
28      '''Greeting by the person.
29
30      Really, that's all it does.'''
31      print 'Hi, my name is %s.' %self.name
32
33    def howMany(self):
34      '''Prints the current population.'''
35      if Person.population==1:
36        print 'I am the only person here.'
37      else:
38        print 'We have %d persons here.' %Person.population
39
40  swaroop=Person('Swaroop')
41  swaroop.sayHi()
42  swaroop.howMany()
43
44  kalam=Person('Abdul Kalam')
45  kalam.sayHi()
46  kalam.howMany()
47
48  swaroop.sayHi()
49  swaroop.howMany()
```

- 使用继承

```
1   #!/usr/bin/env python
2   #Filename: inherit.py
3
4   class SchoolMember:
5     '''Represents any school member.'''
6     def __init__(self,name,age):
7       self.name=name
8       self.age=age
9       print '(Initialized SchoolMember: %s)' %self.name
10
11    def tell(self):
12      '''Tell my details.'''
13      print 'Name:"%s" Age:"%s"' %(self.name,self.age),
14
15  class Teacher(SchoolMember):
16    '''Represents a teacher.'''
17    def __init__(self,name,age,salary):
18      SchoolMember.__init__(self,name,age)
19      self.salary=salary
20      print '(Initialized Teacher: %s)' %self.name
21
22    def tell(self):
23      SchoolMember.tell(self)
24      print 'Salary: "%d"' %self.salary
25
26  class Student(SchoolMember):
27    '''Represents a student.'''
28    def __init__(self,name,age,marks):
29      SchoolMember.__init__(self,name,age)
30      self.marks=marks
31      print '(Initialized Student: %s)' %self.name
32
33    def tell(self):
34      SchoolMember.tell(self)
35      print 'Marks: "%d"' %self.marks
```

```
36
37 t=Teacher('Mrs. Shrividya',40,30000)
38 s=Student('Swaroop',22,75)
39
40 print # prints a blank line
41
42 members=[t,s]
43 for member in members:
44   member.tell() # works for both Teachers and Students
```

- 使用文件

```
1  #!/usr/bin/env python
2  #Filename: using_file.py
3
4  poem='''\
5  Programming is fun
6  When the work is done
7  if you wanna make your work also fun:
8    use Python!
9  '''
10
11 f=file('poem.txt','w') # open for 'w'riting
12 f.write(poem) # write text to file
13 f.close() # close the file
14
15 f=file('poem.txt')
16 # if no mode is specified, 'r'ead mode is assumed by default
17 while True:
18   line=f.readline()
19   if len(line)==0: # Zero length indicates EOF
20     break
21   print line,
22   # Notice comma to avoid automatic newline added by Python
23 f.close() # close the file
```

- 储存与取储存

```
1  #!/usr/bin/env python
2  #Filename: pickling.py
3
4  import cPickle as p
5  #import pickle as p
6
7  shoplistfile='shoplist.data'
8  # the name of the file where we will store the object
9
10 shoplist=['apple','mango','carrot']
11
12 # Write to the file
13 f=file(shoplistfile,'w')
14 p.dump(shoplist,f) # dump the object to a file
15 f.close()
16
17 del shoplist # remove the shoplist
18
19 # Read back from the storage
20 f=file(shoplistfile)
21 storedlist=p.load(f)
22 print storedlist
```

- 处理异常

```
1  #!/usr/bin/env python
2  #Filename: try_except.py
3
4  import sys
5
6  try:
7    s=raw_input('Enter something --> ')
8  except EOFError:
9    print '\nWhy did you do an EOF on me?'
10   sys.exit() # exit the program
```

```
except:
  print '\nSome error/exception occurred.'
  # here, we are not exiting the program

print 'Done'
```

- 引发异常

```
#!/usr/bin/env python
#Filename: raising.py

class ShortInputException(Exception):
  '''A user-defined exception class.'''
  def __init__(self,length,atleast):
    Exception.__init__(self)
    self.length=length
    self.atleast=atleast

try:
  s=raw_input('Enter something --> ')
  if len(s)<3:
    raise ShortInputException(len(s),3)
  # Other work can continue as usual here
except EOFError:
  print '\nWhy did you do an EOF on me?'
except ShortInputException,x:
  print 'ShortInputException: The input was of length %d, \
was expecting at least %d' %(x.length,x.atleast)
else:
  print 'No exception was raised.'
```

- 使用 sys.argv

```
#!/usr/bin/env python
#Filename: cat.py

import sys

def readfile(filename):
  '''Print a file to the standard output.'''
  f=file(filename)
  while True:
    line=f.readline()
    if len(line)==0:
      break
    print line, # notice comma
  f.close()

# Script starts from here
if len(sys.argv)<2:
  print 'No action specified.'
  sys.exit()

if sys.argv[1].startswith('--'):
  option=sys.argv[1][2:]
  # fetch sys.argv[1] but without the first two characters
  if option=='version':
    print 'Version 1.2'
  elif option=='help':
    print '''\
This program prints files to the standard output.
Any number of files can be specified.
Options include:
  --version : Prints the version number
  --help    : Display this help'''
    else:
    print 'Unknown option.'
  sys.exit()
else:
  for filename in sys.argv[1:]:
    readfile(filename)
```

- 使用列表综合

```
#!/usr/bin/env python
#Filename: list_comprehension.py

listone=[2,3,4]
listtwo=[2*i for i in listone if i>2]
print listtwo
```

- 使用 lambda

```
#!/usr/bin/env python
#Filename: lambda.py

def make_repeater(n):
  return lambda s: s*n

twice=make_repeater(2)

print twice('word')
print twice(5)
```

- 三个备份脚本

```
#!/usr/bin/env python
#Filename: backup_ver1.py

import os
import time

# 1. The files and directories to be backed up are specified in a
# list.
source=['/home/swaroop/byte','/home/swaroop/bin']
# If you are using Windows, use source=[r'C:\Documents',r'D:\Work']
# or something like that

# 2. The backup must be stored in a main backup directory
#Remember to change this to what you will be using
target_dir='/mnt/e/backup/'

# 3. The files are backed up into a zip file
# 4. The name of the zip archive is the current date and time
target=target_dir+time.strftime('%Y%m%d%H%M%S')+'.zip'

# 5. We use the zip command (in Unix/Linux) to put the files in a
# zip archive
zip_command="zip -qr '%s' %s" %(target,' '.join(source))

# Run the backup
if os.system(zip_command)==0:
  print 'Successful backup to',target
else:
  print 'Backup FAILED'
```

```
#!/usr/bin/env python
#Filename: backup_ver2.py

import os
import time

# 1. The files and directories to be backed up are specified in a
# list.
source=['/home/swaroop/byte','/home/swaroop/bin']
# If you are using Windows, use source=[r'C:\Documents',r'D:\Work']
# or something like that

# 2. The backup must be stored in a main backup directory
#Remember to change this to what you will be using
target_dir='/mnt/e/backup/'

# 3. The files are backed up into a zip file
# 4. The current day is the name of the subdirectory in the main
# directory
today=target_dir+time.strftime('%Y%m%d')
```

```
# The current time is the name of the zip archive
now=time.strftime('%H%M%S')

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
  os.mkdir(today) # make directory
  print 'Successfully created directory',today

# The name of the zip file
target=today+os.sep+now+'.zip'

# 5. We use the zip command (in Unix/Linux) to put the files in a
# zip archive
zip_command="zip -qr '%s' %s" %(target,' '.join(source))

# Run the backup
if os.system(zip_command)==0:
  print 'Successful backup to',target
else:
  print 'Backup FAILED'
```

```
#!/usr/bin/env python
#Filename: backup_ver3.py

import os
import time

# 1. The files and directories to be backed up are specified in a
# list.
source=['/home/swaroop/byte','/home/swaroop/bin']
# If you are using Windows, use source=[r'C:\Documents',r'D:\Work']
# or something like that

# 2. The backup must be stored in a main backup directory
# Remember to change this to what you will be using
target_dir='/mnt/e/backup/'

# 3. The files are backed up into a zip file
# 4. The current day is the name of the subdirectory in the main
# directory
today=target_dir+time.strftime('%Y%m%d')
# The current time is the name of the zip archive
now=time.strftime('%H%M%S')

# Take a comment from the user to create the name of the zip file
comment=raw_input('Enter a comment --> ')
if len(comment)==0: # check if a comment was entered
  target=today+os.sep+now+'.zip'
else:
  target=today+os.sep+now+'_'+\
      comment.replace(' ','_')+'.zip'
  # Notice the backslash!

# Create the subdirectory if it isn't already there
if not os.path.exists(today):
  os.mkdir(today) # make directory
  print 'Successfully created directory',today

# 5. We use the zip command (in Unix/Linux) to put the files in a
# zip archive
zip_command="zip -qr '%s' %s" %(target,' '.join(source))

# Run the backup
if os.system(zip_command)==0:
  print 'Successful backup to',target
else:
  print 'Backup FAILED'
```

The End of Appendix E: Python.