第三章 Python基础

1. 语句和语法
    - 分号;允许你将多个语句写在同一行上，语句之间用分号隔开
    - 冒号:将代码的头和体分开
    - 反斜杠\用于将一行过长的语句分解成多行
    - Python使用缩进来分隔代码组(缩进相同的一组语句，多个语句)
        - 缩进四个空格宽度，避免使用Tab
    - 每一个脚本文件都可以当成是一个模块，模块可以包含直接运行的代码块、类、函数或者这些的组合。
2. 变量赋值
    - **在Python中，对象是通过引用传递的**。在赋值时，不管这对象是新创建的还是已经存在的，都是将该对象的引用(并不是值)赋值给变量。
        - Python的赋值语句不会返回值。
        - 链式赋值

```python
>>>  x = 1
>>> y = x = x + 1
>>> x,y
(2,2)
```

- 增量赋值+=,-=,*=等与赋值运算符结合的
- Python不支持自增运算
- 多重赋值

```
>>> x = y = z = 1
>>> x
1
>>> y
1
>>> z
1
```

一个值为1的对象被创建，然后将该对象的同一个引用赋值给x,y,z。如果用id(object)将x,y,z的地址打印出来，地址值会是一样的。

- 多元赋值(可以通过元组实现)

```python
    (x,y,z) = (1,2,'a string')
```

1. 标识符(与C语言类似)

- 关键字构成Python语言的标识符(可以用keyword库中的kwlist属性把所有的关键字列出来)
- build-in标识符
- Python不支持重载标识符，任何时刻都只有一个名字绑定
- __buildins__可以看作全局变量
- 下划线的特殊用法
  - _xxx 不用'from module import * '导入
  - __xxx__ 系统定义的名字
  - __xxx 类中的私有变量名

3. 基本风格指南

- 注释 #开始
- 文档 使用ojb.**doc**来获取,obj为模块、类、函数的名字
- 缩进 缩进四个空格宽度
- 模块布局

1. 起始行(Unix)
2. 模块文档–介绍模块的功能和全局变量的含义,模块外使用module.__doc__来访问
3. 模块导入
4. 变量定义–为全局变量(尽量使用局部变量代替全局变量)
5. 类定义–当模块被导入时class语句会被执行，类也就会被定义，类的文档变量是class.__doc__
6. 函数定义–可以通过module.function()在外部被访问到，当模块被导入时def语句会被执行，函数也就都会定义好，函数的文档变量是function.__doc__
7. 主程序–main()

Examples:

```
#/usr/bin/env python      #(1) Startup line(Unix)


"this is a test module"   #(2) Module document

import sys                #(3) Module imports
import os

debug = True              #(4) (Global)Variable declarations

class FooClass(object):   #(5) Class declarations
    'Fooclass'

def test():               #(6) function declarations
    "test function"
    foo = FooClass()
    if debug:
        print 'ran test()'

if __name__ == '__main__': #(7) main body</td>
    test()
```

- 总之，包含主程序的模块会被直接执行。
- 最高级别的 Python 语句–也就是说，那些没有缩进的代码行在模块被导入时 就会执行，不管是不是真的需要执行。
- **通常只有主程序模块中有大量的顶级可执行代码，所有其他的被导入的模块只应该有很少的顶级可执行代码，所有的功能代码都应该封装在函数或者类中。**

- 通过__name__系统变量判断一个模块是被直接运行还是被导入的
  如果模块是被导入， __name__ 的值为模块名字
  如果模块是被直接执行， __name__ 的值为 '**main**'
- 在主程序中放置测试代码是测试的简单方法。

5. 内存管理
   由Python解释器承担内存管理的任务，
   - 引用计数–在Python内部记录着所有使用中的对象各有多少引用。对象被创建时就创建一个引用计数，当这个对象的引用计数为0是，它就被回收。
   - 引用计数的增加(同一个对象)–该对象新增一个新的引用
   - **对象被创建**并将其引用赋值给变量时，引用计数设置为1
   - 赋值给其他变量
   - 作为参数传递给函数或者方法或类实例
   - 成为容器对象的一个元素
   - 赋值为窗口对象的成员
   - 引用计数的减少–当对象的引用被销毁时，引用计数会减小。
   - 当引用离开其作用范围时(函数运行结束时)，所有局部变量都会被自动凶啊会，对象的引用计数会减少
   - 对象被显式地销毁 del x
   - 对象的别名被赋值另外一个对象 x = 123
   - 对象从一个窗口对象中移除 `list.remove(x)`
   - 窗口对象本身被销毁 `del list`

任何追踪或调试程序会给一个对象增加一个额外的引用， 这会推迟该
对象被回收的时间。

- 不再被使用的内存会被一种称为垃圾收集的机制释放。解释器跟踪对象
  的引用计数，垃圾收集器负责释放内存。垃圾收集器寻找引用计数为0的对象，然后释放内存。

6. 第一个Python程序

```
# writeText.py--提示用户输入每一行文本，然后写入到文件中
import os

ls = os.linesep
# get filename
filename = 'd:\\test.txt'
# get file content lines
allText = []
print "\nEnter lines('.'by itself to quit)."

while True:
    entry = raw_input('> ')
    if entry == '.':
        break
    else:
        allText.append(entry)

# 逐行写入文件
# write lines to file with proper line-ending
fobj = open(filename,'w')
# (x, ls)表示每一行及其行结束符
fobj.writelines(['%s%s' % (x,ls) for x in allText]) #列表解析
fobj.close()
print 'Done!'
```

```
# readText.py--从指定的文件中读取内容，并显示
""" readText.py---read and display text file """

filename = "d:\\test.txt"
try:
    fobj = open(filename,'r')
except IOError,e:
    print "file open error:",e
else:
    # display contents to the screen
    for eachLine in fobj:
        print eachLine
    fobj.close()
```

使用局部变量替换模块变量。
类似 os.linesep 这样的名字需要解释器做两次查询：
（1）查找 os 以确认它是一个模块，
（2）在这个模块中查找 linesep 变量。

os.path.exists() 和异常处理：
异常处理最适用的场合，是在没有合适的函数处理异常状况的时候。

7. 开发工具
   Profilers: profile,hotshot,cProfile
   历史上，因为不同的人们为了满足不同的需求重复实现了很多性能测试器

   - Python profile 模块是 Python 写成的，用来测试函数的执行时间，及每次脚本执行的总时间

- Python2.2中新增了hotshot,用 C 语言写成,性能提高了。hotshot
  重点解决了性能测试过载的问题，但却需要更多的时间来生成结果。
- Python2.5新增cProfile,C写的，修复了hotshot 模块的一个关于时间计量的严重 bug。**缺点**是它需要花较长时间从日志文件中载入分析结果， 不支持子函数状态细节及某些结果不准