Python笔记

调用Python解释器

Python解释器通常安装在目标机器的/usr/local/bin/目录下，将/usr/local/bin目录放进shell搜索路径里，确保它可以通过输入Python启动。

在 Windows 机器中, Python通常安装在C:\Python32目录。

输入一个文件结束符(UNIX上是Control-D，Windows上是Control-Z ) 解释器会以0值退出。若没有起作用，可以输入以下命令退出：quit()。

解释器的行编辑功能并不复杂, 装在 UNIX 上的解释器可能需要GNU readline库支持，这样就可以额外得到精巧的交互编辑和历史记录功能。确认命令行编辑器支持能力最方便的方式可能是在主提示符下输入Control-P，如果有嘟嘟声(计算机扬声器)，说明你可以使用命令行编辑功能(在交互式输入编辑及历史替代在快捷键的介绍)。如果什么也没有发声, 或者显示^P，说明命令行编辑功能不可用；你将只能用退格键删除当前行的字符。

解释器的操作有些像 UNIX Shell：使用终端设备作为标准输入来调用它时，解释器交互地解读和执行命令，通过文件名参数或以文件作为标准输入时，它从文件中解读并执行脚本。

启动解释器的第二个方法是python -c command [arg] ...，这种方法会执行command中的语句，等同于Shell的-c选项。因为 Python语句中通常会包括空格之类的对shell有特殊含义的字符，所以最好把整个command用单引号包起来。

有一些Python模块也可以当作脚本使用。它们可以通过 python- m module[arg]...调用，这如同在命令行中给出其完整文件名来运行一样。

使用脚本文件时，经常会运行脚本然后进入交互模式。这也可以通过在脚本之前加上-i参数来实现。(如果脚本来自标准输入，就不能这样执行，与前述提及原因一样。)

参数传递

调用解释器时，脚本名和附加参数传入一个名为sys.argv的字符串列表。

没有给定脚本和参数时，它至少有一个元素：sys.argv[0]，此时它是一个空字符串，

脚本名指定为'-'(表示标准输入)时，sys.argv[0]被设为'-'

使用-c命令时，sys.argv[0]被设定为'-c'

使用-m模块时，sys.argv[0]被设定为模块的全名

-c command或-m module之后的参数不会被Python解释器的选项处理机制所截获，而是留在sys.argv中，供命令或模块操作。

交互模式

从tty读取命令时，称解释器工作于交互模式(interactive mode)。这种模式下它通过主提示符(primary prompt)提示下一条命令，主提示符通常为三个大于号(>>>)；而通过从提示符由(三个点标识...组成)提示一条命令的续行。在第一条命令之前，解释器会打印欢迎信息，版本号和授权

输入多行结构时就需要从属提示符

解释器及其环境

错误处理

有错误发生时，解释器会输出错误信息和栈跟踪。交互模式下，它返回到主提示符，如果从文件输入执行，它在打印栈跟踪后以非零状态退出。(在try语句中抛出并被except从句处理的异常不是这里所讲的错误)。一些非常致命的错误会导致非零状态下退出，这通常由内部问题或内存溢出造成，所有的错误信息都写入标准错误流；命令中执行的普通输出写入标准输出。

在主提示符或从属提示符后输入中断符(通常是Control-C 或者 DEL)就会取消当前输入,回到主提示符。执行命令时输入一个中断符会抛出一个KeyboardInterrupt异常，它可以被try语句截获。

可执行的Python脚本

Python脚本可以像shell脚本那样直接执行，只要在脚本文件开头加一行文本来声明模式：

#! /usr/bin/env python3.2

(要先确认Python解释器存在于用户的PATH环境变量中)。#! 这两个字符必须是文件的头两个字符。在某些平台上，第一行必须以UNIX风格的行结束符('\n')结束，不能用Windows ('\r\n')的行结束符。注意，#用于Python一行注释的开始。

脚本可以用chmod命令指定可执行模式或权限：$ chmod +x myscript.py

在 Windows 系统下, 没有“可持行模式(executable mode)”的概念。Python安装器会自动地把.py后缀的文件与python.exe 绑定, 因此双击一个Python文件，就可以把它作为脚本来运行。扩展名也可以是.pyw，这时工作台窗口会隐藏不被打开。

源程序编码

默认情况下,Python源码文件以UTF-8编码. 在这种编码下, 世界上 大多数语言的字符都可以用于,字符串常量,标识符,以及注释——尽管标准库遵循一个所有可移植代码都应遵守的约定：仅使用ASCII字符作为标识符。要正确地显示所有这些字符，编辑器一定要有能力辨认出是UTF-8编码，还要使用一个支持所有文件中字符的字体。

也可以为源码文件指定不同的编码，要在#!行后面指定一个特殊的注释行，以定义源码文件的编码：

# -\*- coding: encoding -\*-

有了这样的声明，源文件中的所有字符都会被以encoding的编码来解读，而非UTF-8。在Python库参考的codecs一节可以找到所有可用的编码。

例如，如果使用的编辑器不支UTF-编码，但是支持另一种称为Windows-1252的编码，可以在源码中写上：

# -\*- coding: cp-1252 -\*-

这样就可以在源码文件中使用Windows-1252字符集。这个特殊的编码注释必须在代码文件的第一或第二行。

交互式启动文件

交互式地使用Python解释器时，可能需要在每次启动时执行一些命令。设置一个名为PYTHONSTARTUP的变量，指向包含启动命令的文件，类似于Unix Shell的.profile文件。

这个文件只在交互式会话中才被读取，当Python从脚本中读取命令或显式地以/dev/tty作为命令源时(尽管行为很像是处在交互会话期)则不会如此。它与解释器执行的命令处在同一个命名空间，所以由它定义或引用的一切可以在解释器中不受限制的使用。也可以在这个文件中改变sys.ps1 and sys.ps2的值。

如果要在当前目录中执行额外的启动文件，可以在全局启动文件中加入类似以下的代码：

ifos.path.isfile('.pythonrc.py'):exec(open('. pythonrc.py').read()).

如果要在某个脚本中使用启动文件，须要在脚本中写入这样的语句：

import os

filename = os.environ.get('PYTHONSTARTUP')

if filename and os.path.isfile(filename):

exec(open(filename).read())

定制模块

Pytho提供两个钩子(hook)来定制交互环境：sitecustomize和usercustomize。要知道它如何工作，先找到user site-package 目录的位置，打开Python并运行这段代码：

>>> import site

>>>site.getusersitepackages() '/home/user/.local/lib/python3.2/site-packages'

现在可以在那个目录下创建一个名为usercustomize.py的文件，并在里面放置任何想放的东西。它将影响到每一次Python的调用，除非使用了-s选项来禁用了自动导入功能。

sitecustomize以同样的方式工作，但通常由该计算机的管理员在全局site-packages目录下创建，并且在usercustomize之前被导入。