[初学入门]ArcGIS 中 Python 脚本的使用

By：飞天小猪

目录

写在前面的话 2

前言 2

一、PYTHON 语言基础 3

1 数学运算符 3
2 字符串操作 4
3 模块的使用（MODULES） 5
4 使用 DEF 构建函数 6
5 流程控制结构：IF，WHILE，FOR 7
6 简单输入和输出 9

二、ARCGIS&PYTHON 10

1 如何创建地理处理对象（GEOPROCESSOR OBJECT） 10
2 获取地理处理帮助 11
2.1 举例：如何使用 Geoprocessor Programming Model 中的 Lists 11
3 使用地理处理工具——TOOLBOXES 和 ALIASES 12
4 在建模中使用脚本（SCRIPTS IN MODELBUILDER） 13
5 在 PYTHONWIN 里调试地理处理脚本 19
5.1 调试选择和消息 20
5.2PythonWin 的调试工具 21
5.3 地理处理工具举例 22
6 使用描述（DESCRIBE）和存在（EXISTS）获取数据信息 22
6.1 描述 23
6.2 存在（Exists） 24
6.3 在循环中使用描述和存在 24
7 在 PYTHON 脚本中使用地图代数（MAP ALGEBRA） 27
8 数据管理和指针（DATA MANAGEMENT AND CURSORS） 28
8.1 数据管理（Data Management） 28
8.2 指针（Cursors） 29

附录 1：地理处理脚本中输入&输出方法指南 32

附录 2：其他 33

写在前面的话

一直想学习 ArcGIS 中的 Python 脚本，大四下半学期终于有了时间，可是想找到这么一本好的教材不容易。茫茫互联网，终于找到了旧金山州立大学 Jerry Davis 教授的个人主页，对其中《Geoprocessing Scripts With Python》如获至宝，独乐乐不如众乐乐，现在将其教程翻译并结合自己的学习情况给出总结。希望能够给更多想学习 Python 的同学一个参考。

另外，在我刚开始接触 Python 时，是看了台湾辅仁大学一位老师的视频课件，在此致谢。

我想从两个大部分总结：一、Python 语言基础；二、ArcGIS&Python。其中第一部分参考了《Python 精要参考（第二版）》、《Python 编程金典（读书笔记）》等书籍文献。对于多数读者来说，可能或多或少有一些编程基础，所以理解起来应该不成问题。

文中多数数据来自 Jerry Davis 教授的主页，放在“C:\prog”目录下，为了直观，我将运算结果一并编辑，方便参考。

值得一提的是 ArcGIS 的在线帮助文档，一个实时更新的 GIS 宝库，很多专业性知识都可以找到答案，点击链接 ArcGIS10 中文帮助、ArcGIS9.3.1 或 9.3 英文帮助。获取更过脚本例子来学习 ： ESRI 的地理处理模型和脚本工具库是个不错的选择。

由于我也是初次接触，翻译或者心得难免有纰漏之处，希望同仁们可以多多交流！

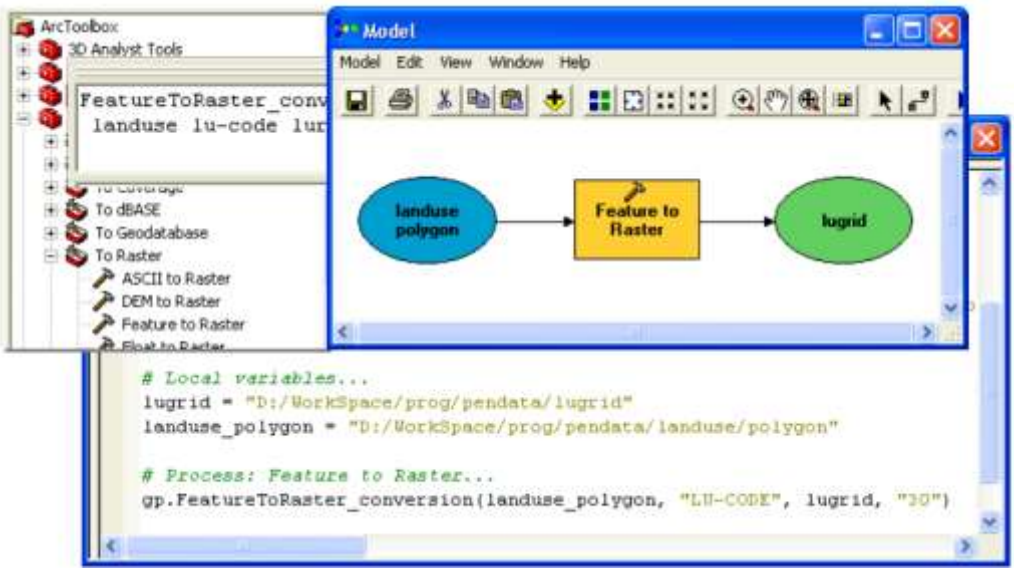
前言

在 GIS 建模或 GIS 数据管理中，你可能经常需要处理一系列步骤才可以完成的工作；你可能有一个工作目录下的数据需要重投影、裁剪到研究区域，或者用某种方法组合成期望的结果；我们也经常需要根据不同情形用不同方法处理数据，因此我们需要作出选择，而高质量的决策需要考虑很多低水平的决策，这可以通过脚本程序模型辅助完成。

脚本编程的主要目的是使枯燥的处理数据工作自动化，通过逻辑来指挥处理过程。我想自动化和逻辑是关键，它们区别于我们多数使用计算机时的交互活动。我们发 E-mail，写文章或者设计地图，都需要和计算机交互，而处理一系列数据，我们需要自动化和利用逻辑来指导自动化。

在地理处理脚本逻辑中，我们需要在允许我们做的事情中作出决定，比如，处理栅格数据不同于矢量数据，或为没投影的数据设置投影，或处理仅在特定时间搜集的数据集。对于重要的 GIS 工作来说，脚本以及其他形式的程序是必需的，而非可有可无。

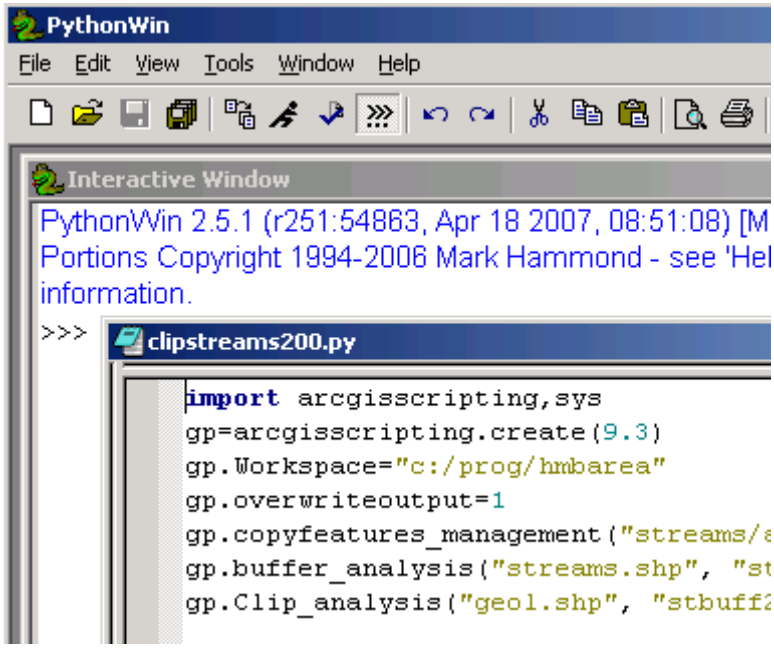
在接下来的联系中，我们会探索 Python 的使用以及创建脚本来使用 ArcGIS 里众多的地理处理工具。所有你能在 ArcToolbox 或 Model 中使用的工具都能够用在 Python 脚本中，这些脚本可以生成脚本工具，像其他地理处理工具一样使用。



一、Python 语言基础

安装 PythonWin，在...\ArcGisDesktop9.3.iso\Desktop\PythonWin 目录下可以找到 PythonWin 的安装程序，默认是不安装的，

。同时会安装 win32com 以及允许任何脚本在基于 Dispatch 的地理处理过程中工作。ArcGIS10 中引入了全新的 Python Window 来增强内嵌的 Python 体验。



警告：不要尝试更新随 ArcGIS 安装的 Python 到一个新的版本！

下面介绍 Python 的一些简单语法和规则。

1 数学运算符

Python 提供了多样化的通用数学运算符——多数编程语言的特征，以及许多通过 import 的 modules 提供的符号。常用的有+,-,*,/,**(幂),%(取模，即除后的余数)。

下面的表格显示了整型（Integer）和浮点型（Float）各种组合运算的结果，记住一条规则，只要参与运算的有浮点型，则结果为浮点型；全为整型时，结果才为整型。

输入表达式	结果	Notes
2+3	5	整型结果
2.+3	5.0	2.是浮点型，结果浮点型
2-3	-1	
2*3	6	整型结果
2.*3	6.0	浮点型
5/2	2	整型
5./2	2.5	
5%2	1	取模
Az=270 Newaz=az+180 Print newaz%360	90	取模的用途之一——方位角加 180 后逆转方向
5**2	25	
25**0.5	5.0	没有 sqrt()功能，除非添加 math 模块

2 字符串操作

注：使用 Python 帮助：有超过 30 种内置方法来处理字符，请到 Sequence Types 下的 String Methods 寻找帮助！

字符串是一串字母，比如'San Francisco'，字符串下标从 0 开始。学习字符串语法的最好方法是自己动手尝试，下标展示之：

输入	结果	Notes
print 'zhulj'.capitalize()	Zhulj	**s.capitalize()**即将 capitalize()方法用于 s
s='zhulj' print s.capitalize()		
print s[0]	z	Strings 可以像一个字母列表一样处理，第一个字母下标为 0，某个字符段可以用 1:3 来格式化：从第 1 个的开头到第 3 个的开头，不包括下标为 3 的字母；s[-1]表示倒过来第一个，相当于 s[len(s)-1]
s1=s[1] print s1	h	
print s[-2:]	lj	
print s[2:3]	u	
print s[2:4]	ul	
print s[2:],s[:5]	ulj zhulj	
s2=s.upper() print s2	ZHULJ	我们可以将字符串方法的结果赋给新的变量
s3=s+s2 print s3	zhuljZHULJ	字符串组合用“+”
print s*3	zhuljzhuljzhulj	字符串重复用“*”，后为重复次数
selstr='"elev">1000' print selstr	"elev">1000	字符串可以使用单引号或双引号，跨行时用双引号。
othersel="'elev'>1000" print othersel	'elev'>1000	
print s.isupper()	False	一些方法返回值为布尔型（True 或 False），一些返回索引值（下标值）
print s2.isupper()	True	

p='d:/work/lu.shp' print p.find('.')	10	
print p.find('/')	2	
plist=p.split('/') print plist	['d:', 'work', 'lu.shp']	你可以用 split()方法解析出不同的字符串片段，并创建一个列表（List），我们可以使用其中不同的元素
print plist[0]	d:	
print plist[1]	work	
p2='d:\\work\\soil.shp' print p2	d:\work\soil.shp	反斜线“\”和某些字母一起有特殊用法，如\n为换行，“\”为转义字符，如“\\”则表示“\”
print 'Jerry\'s Kids'	Jerry's Kids	
print 'Jerry\'s\nKids'	Jerry's Kids	
p3=r'd:\work\soil.shp' print p3	d:\work\soil.shp	字符串前加“r”则强制“\”代表其本身，而非转义字符，这对于文件路径的操作很方便

3 模块的使用（Modules）

Python 提供了一系列内置的方法（大量依赖于模块）用于通用编程。Python 安装时自带了大量 Modules，最常用的有 math,sys,random,array 以及 os.path。

当然还有好多 Modules 可以下载，比如数字处理（Numeric）——numpy，可在 www.python.org 或 www.google.com 里搜索。www.python.org/moin/NumericAndScientific 页面中列举了一些。

使用 Module 前，必须 import 之。通常我们会将一行 **import <Module 名>**放在程序顶部，比如：

```
import arcgisscripting
```

当然，这不必成为你程序的第一行，但必须在使用它里面方法之前。

当要引用多个模块是，中间用逗号分隔，比如：

```
import arcgisscripting,sys,string,os,math
```

我们也可以自己为频繁使用的方法创建 Module，下面，我们开始体验内置的 Modules。

math 和 random 模块

很多常用的数学计算功能都可以通过 math 找到，比如三角计算或对数计算，如果要使用复杂数字，就使用 cmath 模块。

和之前一样，通过以下表格来体现模块的使用：

输入	结果	Notes
import math print math.log10(100)	2.0	以 10 为底的对数
print math.log(100)	4.60517018599	自然对数
print math.pi	3.14159265359	π 是一个静态常量，所以不需要括弧

pi=math.pi print pi	3.14159265359	如果不想总是输入“math.pi”可以将其赋给一变量
pi	3.1415926535897931	不需要 print 即可查看变量值
print math.sin(radians) print math.cos(radians) print math.tan(radians)		三角函数的计算是弧度制
degrad=pi/180 45*degrad	0.78539816339744828	度转化为弧度
sin=math.sin sin(45*degrad) sin(90*degrad)	0.70710678118654746 1.0	即使功能函数（像 sin）都可以赋给一个变量
math.e	2.7182818284590451	
math.hypot(3,4)	5.0	此方法是求三角形的斜边
x1=520382;y1=4152373 x2=520475;y2=4152963		不同赋值语句间用“;”分隔
xr=x2-x1 yr=y2-y1 math.hypot(xr,yr) math.sqrt(xr**2+yr**2) (xr*xr+yr*yr)**0.5	597.28468923956189	不同的方式，相同的结果
import random random.random()	0.27281588185756478	random()方法，每次结果都不同，值域为[0.0,1.0)
rnd=random.random rnd()	0.4456393835072503	
mu=50 s=10 print random.gauss(mu,s)	46.528202194	

4 使用 def 构建函数

有点像 Module，但更简单，函数是一个自己定义功能，用在之后的代码中，并且提供任何你想要使用的参数。这个函数从此可像变量那样在程序中使用，结合例子更容易理解。接下来的代码定义了一个将度转换为弧度的简单函数，同时也定义了一个弧度转换为度的函数，它们和 Excel 内置的函数类似。

```
import math
def radians(angdeg):
    return angdeg*math.pi/180
def degrees(angrad):
    return angrad*180/math.pi
print math.sin(radians(45))
print degrees(math.acos(0.5))
```

运行之，得到结果：0.707106781187　　60.0

5 流程控制结构：If，While，For

任何脚本或编程语言的一个重要特征就是执行一系列不同情形语句的能力。

你想要创建一系列山影栅格来代表夏天、冬天和春秋分。山影（hillshade）工具需要有太阳高度角和方位角作为输入参数。

重要日期	太阳倾角
夏至（6 月 21 日）	23.44
春秋分（3 月 21 日，9 月 21 日）	0
冬至（12 月 21 日）	-23.44

接下来是一段相当简单的代码，通过太阳倾角（太阳光线正午垂直照射的纬度）获取太阳角和方位角以及纬度。输入两个参数：**lat**（研究区域的纬度，南半球为负）和 **decl**（太阳倾角），由此得到 **sunangle** 和 **azimuth**：

```
lat=30
decl=20
sunangle=90-lat+decl
azimuth=180
if sunangle>90:
    sunangle=180-sunangle
    azimuth=0
    print sunangle,azimuth
```

上面的例子中 **lat** 和 **decl** 强制赋了值。

有三种流程控制操作：

if　　仅在一个特定情形下才执行语句；

while　　当一种情形存在下，持续执行语句

for　　遍历一系列值

这些语法和 def 有些相似：初始语句后加顿号、需要执行的语句块有缩进。

这三个结构的一些重要的公共特征：

①if、while、for 语句均以冒号结尾，接下来是缩进的代码块，用于 if、while、for 定义的情形。在脚本编写窗口，你会发现，你在一行末尾打上冒号后，下一行自动缩进，在接下来的一行按下退格键取消缩进。

②如果你只需做一件事情，你可以在冒号后面同一行添加简短的语句，比如：

```
if x>0: print ‘x 比 0 大’
print ‘下一行不要缩进了。。。’
```

if（continued）

接下来，我们会探索一下另一个方便的模块：os.path:

开始之前，在 d:/下创建一个“testfolder”文件夹，然后新建一个“test.txt”文件；

尝试以下代码段，确保 print 语句前有缩进。

```
import os.path
if os.path.exists("d:/testfolder/test.txt"):
    print "测试文件夹存在"
    print "txt 文件存在"
```

```
elif os.path.exists("d:/testfolder"):
    print "测试文件夹存在"
    print "测试文件夹存在，但 txt 文件不存在"
else:
    print "两者都不存在"
```

可选探索示例

接下来的例子做的事情对 GIS 非常重要，但是实际上不用任何地理处理代码。USGS7.5 米分辨率 DEM（数字高程模型）是文本文件（USGSDEM 文件），投影为 UTM，UTM 北向和东向单位是米，但是高程单位可能是英尺（feet）或米（meters）。因此在获取垂直或水平距离信息时会有问题，比如坡度可以通过垂直距离/水平距离获得。如果你不在使用 Z 值之前设置为 0.3048，将会出现错误结果。但是不幸的是，你可能不知道 DEM 文本文件的垂直单位是英尺还是米。这些信息保存在第 539 个字符里，“1”代表英尺，“2”代表米，所以可以通过读取这个文件判断。

下面的脚本演示了上述内容：

```
import fileinput
infile=r"c:\prog\pendata\woodside.dem"
firstline=fileinput.input(infile)[0]
unitchar=firstline[539]
unit="(unknown:not a 7.5' DEM?)"
if unitchar=="1":unit="feet"
if unitchar=="2":unit="meters"
print "\nElevation in"+" "+unit
fileinput.close()
```

输出结果：Elevation in feet

while（continued）

- 运行下面的代码，说明了一种 while 循环：

```
x=1
while x<10:
    print x
    x=x+1
```

屏幕依次输出 1~9

- 下面说明一下“==”（等于）的概念：

x=5 z=x==4 print z 输出 False	x=5 z=x==5 print z 输出 True

“==”是逻辑运算符之一，其他有“<”（小于）、“>”（大于）、“>=”（大于或等于）、“<=”（小于或等于）、“<>”（不等于）。

使用逻辑运算符计算得到结果为布尔型：true（1）和 false（0）。

下面例子简单体会一下布尔型表达式：

```
x=1
while x<10:
```

```
print x
x=x+1
```

表达式“x<10”结果是 true 或 false，所以这样允许我们在计算完一种情况时运行一系列代码。许多情况下我们需要使用条件代码。

while 循环的一个优点是允许我们跳过整个部分，如果条件不满足初始情况。由于 while 提供一种容易结束循环的方法，我们甚至用它代替 if 语句。当循环一个数据集时（GIS 中很常用的工作）while 循环很有用。在后面地理处理中我们会接触一些例子。

for

- 尝试下面代码，演示了 for 循环：

```
for x in [1,2,3,4]:（注：[1,2,3,4]用 range(1,5)代替是一样的）
    msg="hello world"
    print str(x)+" "+msg（注：当我们希望一个数字 x 和字符串连接时，必须先对数字进行格式转换即 str(x)，否则系统报错）
```

- 下面的代码创建并输出指定文件夹内 shp 文件名列表（每个都以‘.shp’结尾）

```
import os
ws="c:/prog/hmbarea"
ilist=os.listdir(ws)#创建一个列表保存工作文件夹内的文件
fcs=[]#创建一个空列表，保存结尾为‘.shp’的文件
for i in ilist:
    if i.endswith(".shp"):
        fcs.append(i)
for fc in fcs:
print fc（输出结果如右图所示）
```

```
geol.shp
geolstr200.shp
landuse.shp
stbuff200.shp
streams.shp
water.shp
```

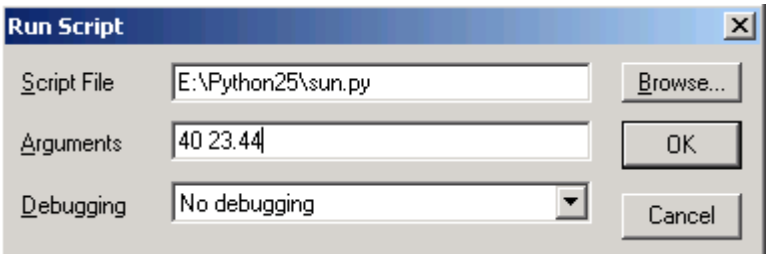
- 下面这个例子的循环较多次数。变量 mu 是算术平均数，s 是标准差——这两个是 random.gauss()用到的参数，可以尝试改变 n 的值查看结果！

```
import random
mu=50
s=10
z5=mu-s*1.96
z95=mu+s*1.96
count=0
n=100000
for i in range(n):
    x=random.gauss(mu,s)
    if x<z5 or x>z95:count=count+1
print  float(count)/n（每次运行的结果都不同，但都在 0.05 左右，即统计结果在 5%左右）
```

6 简单输入和输出

我们现在利用前面计算太阳角代码的片段，之前是直接指定参数值，现在我们有很多种

方法提供输入参数，现在我们用 sys 方法。尝试下面的代码，点击  （run 运行）之后，在对话框中函数自变量（Arguments）一栏填入：40 23.44，如下图：




```
import sys
lat=float(sys.argv[1])
decl=float(sys.argv[2])#使用 arguments（argv）方法从“Arguments”一栏中获取输入参数，并指定一个浮点型转换将字符型输入值传递给 lat 和 decl
sunangle=90-lat+decl
azimuth=180
if sunangle>90:
    sunangle=180-sunangle
    azimuth=0
print "正午太阳角="+str(sunangle)
print "方位角="+str(azimuth)（结果：正午太阳角=73.44    方位角=180）
```

二、ArcGIS&Python

1 如何创建地理处理对象（geoprocessor object）

所有 geoprocessing 的 Python 脚本都可以通过 import arcgisscripting 或者 win32com 去穿件 geoprocessor object。下面的例子显示二者区别：arcgisscripting module 需要在 Python2.5.1 版本上创建并且需要此版本创建 geoprocessor；通过 win32com 创建的 geoprocessor 可以在不同的 Python 版本上运行。

```
#9.3
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.workspace=”c:/Tongass”
gp.clip_analysis(“standb4”,”clipcov”,”standb4_clip”,”POLY”.”1.25”)

#Dispatch
import win32com.client
gp=win32com.client.Dispatch(“esriGeoprocessing.GpDispatch.1”)
gp.workspace=”c:/Tongass”
gp.clip_analysis(“standb4”,”clipcov”,”standb4_clip”,”POLY”.”1.25”)
```

✧ 理解 ArcGIS 中数据多样性格式对我们理解地理处理工具很有帮助。比如，特征数据可能是①单个 shp 文件；②geodatabase（地理数据库，我们可能指定地理数据库为

工作空间）；③多边形、弧或点要素的 coverage 数据。当我们想遍历工作空间里的 coverage 文件时，应使用 ListDatasets 而不是 ListFeatureClasses。

2 获取地理处理帮助

如果你基本熟悉了 Python 的语法，便可以开始熟悉 ArcGIS 的 Geoprocessing 啦，获取这些方法帮助的途径有两个：

① ArcGIS 帮助系统，Go To Geoprocessing/Automating your work with scripts/Scripting object:Properites and Methods.

这里你会看到 Geoprocessor Object,这个能让你很方便地获得所有对你有用的条目、设置和其他操作文档。比如，你想得到特定类型的文件，就找到 listfeatureclasses 方法：fcList=gp.ListFeatureClasses("w*","polygon")

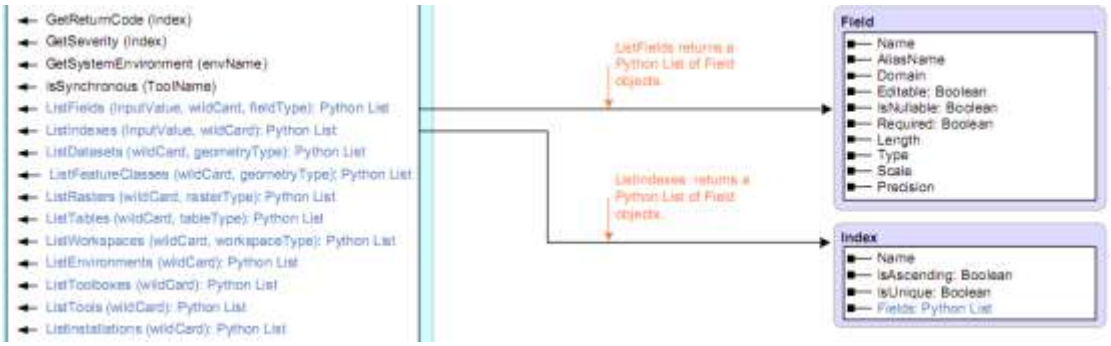
注：此方法的语法为：object.ListFeatureClasses({wildCard} As String, {FeatureType} As String) As Python List

{} = optional　wildcard 为通配符，和星号（*）组合使用，如果没有通配符则返回工作目录下的所有 feature classes。

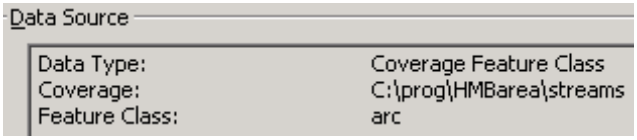
② Geoprocessor Programming Model（PDF），包含方法（左箭头表示）、属性（可读写的表示为杠铃形，只读的表示为部分杠铃形）

2.1 举例：如何使用 Geoprocessor Programming Model 中的 Lists

Lists（列表）及其属性和方法在图表中用紫色标出，如下：



现在我们试着编写一段脚本列举出属性表中所有属性值（Fields）（以 hmbarea 栅格土地利用为例，文件存在 c:\prog\hmbarea 下）




```
import arcgisscripting, sys
gp = arcgisscripting.create(9.3)
 gp.workspace = " c:/prog/hmbarea"
fieldList = gp.ListFields("landuse", "*", "all")
dsc=gp.describe("landuse")
print "landuse"+" "+dsc.DataType+":"
for field in fieldList:
...     print field.Name, field.Type (此时输出结果如右图)
```

```
>>> execfile('showFieldsDesc.py')
landuse RasterDataset:
Rowid OID
VALUE Integer
COUNT Integer
LU-DESC String
```

3 使用地理处理工具——Toolboxes 和 Aliases

总所周知，地理处理工具在脚本中的使用和 ArcToolbox 中相同，但是需要提供工具名称来使用它们。但是记住一个名称可能有好几个工具，比如，裁剪工具（Clip）在 Coverage->Analysis->Extract 工具集里，另一个是在 Data Management Tools 下的 Raster 工具集下。区别是每个工具适用不同的数据类型，但是我们如何在脚本中区分它们呢？

在 ArcToolbox 中，我们可以随意选择要使用的工具，但在脚本里就必须在某些方面加以区分。因此我们使用 Aliases（别名）——已经成为工具名称的一部分。

每一个工具都有自己的别名，我们可以通过右键->属性来查看：

Aliase	Toolbox		“conversion”	Conversion
“3d”	3D Analyst		“geocoding”	Geocoding
“analysis”	Analysis		“ga”	Geostatistical Analyst
“arc”	Coverage		“lr”	Linear Referencing
“management”	Data Management		“sa”	Spatial Analyst
“cartography”	Cartography		“stats”	Spatial Statistics

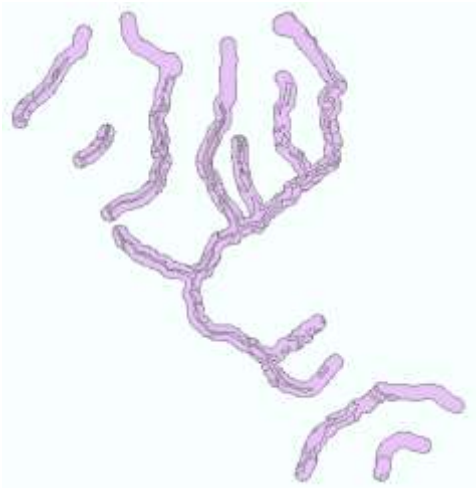
现在我们用一段脚本来解释：

```
import arcgisscripting,sys
gp=arcgisscripting.create(9.3)
gp.Workspace=”c:/prog/hmbarea”
gp.overwriteoutput=1 #OverWriteOutput:Boolean,为 1 表示允许覆盖已存在文件
# 将 streams/arc 转换为 shp 文件
gp.copyfeatures_management("streams/arc", "streams.shp")
#利用转换后的 shp 文件，做 200 米的缓冲
gp.buffer_analysis("streams.shp", "stbuff200.shp", 200)
# 用做过缓冲的 shp 裁剪
gp.Clip_analysis("geol.shp", "stbuff200.shp", "geolstr200.shp")
```

注：上面脚本用“#”注释的中文内容不要出现在脚本文件中，否则会出现错误

```
SyntaxError: Non-ASCII character '\xce' in file clipstreams200.py on line 5,
but no encoding declared; see http://www.python.org/peps/pep-0263.html for
details (clipstreams200.py, line 5)
```

结果截图：

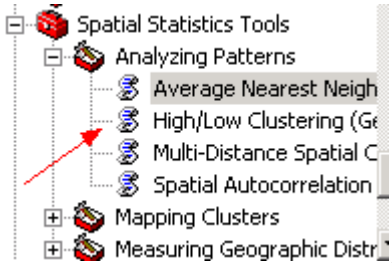


如果你一次使用一个工具集中的若干工具，可以通过环境设置省下一些文字：

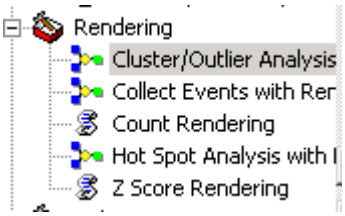
```
gp.Toolbox = "Analysis"
gp.Buffer("streams.shp", "stbuff200.shp", 200)
gp.Clip("geol.shp", "stbuff200.shp", "geolstr200.shp")
```

4 在建模中使用脚本（Scripts in ModelBuilder）

首先，需要记住的很重要的一点是，ArcToolbox 里相当数量的工具实际上都是脚本。脚本都有一个 Script 图标。比如，空间统计分析工具（Spatial Staistics tools）



几乎都是 Python 脚本（一些是模型中使用了脚本



）。实际上，我们可以复制并编辑这些脚本作为其他用途。

为了在 ModelBuilder 中使用脚本或将脚本当做 ArcToolbox 中工具使用，我们需要考虑如何给它输入值以及让其设置输出值。仍然以太阳角计算代码为例，我们给其加上两句引用，四句输入输出语句，就可以用作 Modelbuilder 中的一个步骤了。

编辑下面的脚本，不过不要再 PythonWin 中运行之，因为 getparameterastext 和 setparameterastext 只能用在脚本工具（ArcToolbox 或 Modelbuilder）中。

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
```

```
lat=float(gp.getparameterastext(0))
decl=float(gp.getparameterastext(1))
sunangle=90-lat+decl
azimuth=180
if sunangle>90:
    sunangle=180-sunangle
    azimuth=0
gp.setparameterastext(2,str(sunangle))
gp.setparameterastext(3,str(azimuth))
```

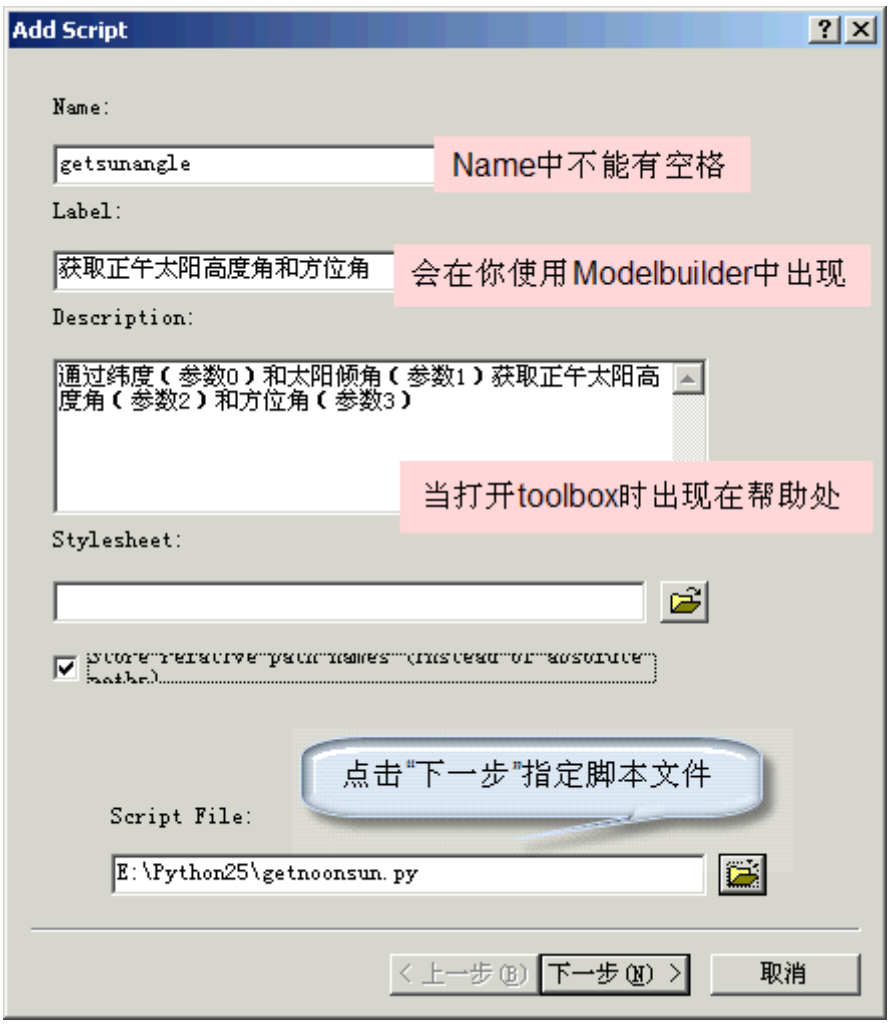
这段代码中的“新面孔”：

getparameterastext：是 Python 传递给 geoprocessor（我们称之为 gp）的一个方法，允许工具提供输入参数。每次你运行这个工具时，都会看到一个对话框，提示输入参数，这个方法允许你在接下来的程序中使用。索引 0 和 1 指第一个和第二个参数。

setparameterastext：和 getparameterastext 相反，它传递第二个条目（比如 str(sunangle)的值）给指定的输出参数。前两个参数索引为 0 和 1，所以接下来输出参数的索引为 2 和 3。

然后，我们需要将脚本加进工具（Making a script into a tool），那样才能在 ArcToolbox 或 ModelBuilder 或 Command line 中使用。如前面提到的那样，这个脚本只能用于工具，包括输入/输出方法是 PythonWin 不能处理的，但这些是多数工具必需的。

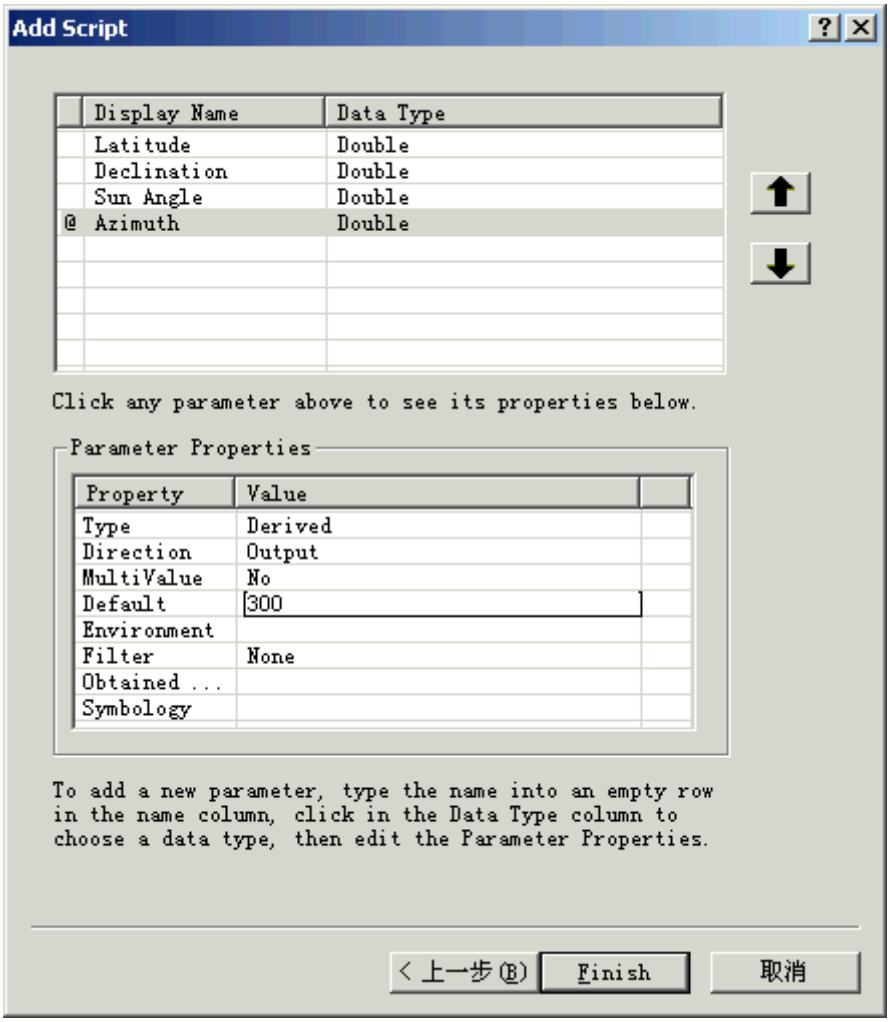
- 在 ArcCatalog 里，定位到 Python 脚本保存的文件夹，最好和数据在一个盘里，右键->新建 toolbox。当然也可以使用之前创建的 toolbox。
- 在 ArcToolbox 里，右键 toolbox，选添加->scripts，填写如下图：



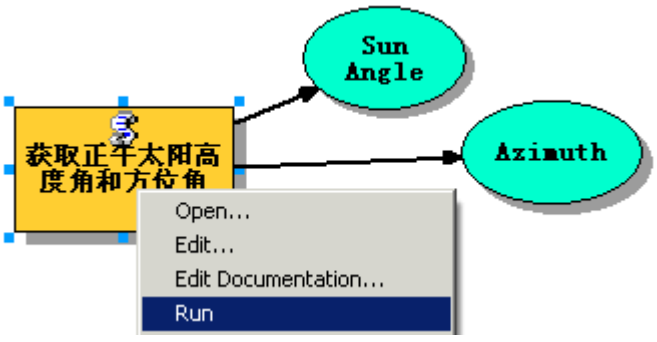
注意：脚本文件是一个脚本工具的参考！非常重要的一点，你使用脚本创建了一个脚本工具，但是脚本本身并没有和工具一起保存，脚本工具作为 toolbox 的一部分保存在“*.tbx”文件中。你也可以右键脚本工具点击“编辑”，出现 PythonWin 或其他任何 IDE 窗口，这看起来好像是脚本存在于工具中。所以，记得移动时将脚本工具文件和脚本本身一起拷贝。

“下一步”后是参数配置页面，如下图设置各参数如表格所示：

Display Name	Data type	Direction	Default
Latitude	Double	input	0
Declination	Double	input	0
Sun Angle	Double	output	35
Azimuth	Double	output	300

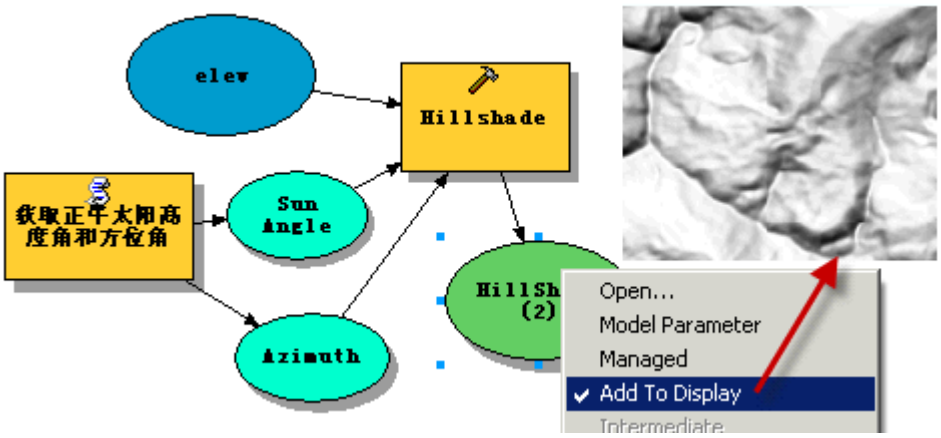


- 现在可以运行此 toolbox 了，对话框提示你输入 latitude 和 declination。工具是否正确运行呢？如果是的话，你应该可以看到“Completed”，你可能会看到有一黑色窗体一闪而过，不用担心。那么，它是干什么的呢？
  还记得结果是输出两个数字参数，那么，这些数字哪去了呢？
  很好的问题，这仅能说明你能创建一个工具，但是不能想 ArcToolbox 那样运行。比如输出一种数据，栅格或特征数据（.shp）之类的。
- 但是它能在作为 Modelbuilder 工具正常运行，下面我们将使用它的输出参数创建一个 hillshade。
- 在你的 toolbox 中新建一个 model，将刚才创建的脚本工具（script tool）拖进来。
- 双击工具，输入参数，初始化之。打开“Sun Angle”和“Azimuth”发现它们还是默认值，说明此脚本工具还没有运行。右键单击工具，选择 Run，然后发现两个输出参数已经改变！



需要注意的是：latitude 范围是-90~90，declination 范围是-23.44~23.44。尝试输入 latitude -70，declination 23.5，你会得到什么？为什么？

- 确保你已经得到值域内的太阳角和方位角，它们将是构建 hillshade 的输入参数。首先，添加 hillshade 工具，双击指定一个 elevation 栅格数据（这里我选择了 marbles 文件夹下的 elevation），用下拉条指定 azimuth 和 altitude 值为 azimuth 和 sun angle。运行，然后右键单击输出文件，选“Add to Display”在 ArcMap 里查看结果。



✧ **探索 1：**

如果你想通过日期获取太阳倾角，该怎么做呢？尝试下面代码，保存为 getnoonsunfromdata.py，现在有五个参数，如下表进行正确设置：

Display Name	Data type	Type	Direction	Default
Month	Long	Required	input	1
Date	Long	Required	input	1
Latitude	Double	Required	input	0
Sun Angle	Double	Derived	output	35
Azimuth	Double	Derived	output	300

```
import win32com.client,sys,math
gp=win32com.client.Dispatch("esriGeoprocessing.GpDispatch.1")
#构建两个函数，首先判断输入月/日的合法性，然后获取是一年中的第几天
def jdate(im, id):
    # 通过年月日起返回一年中的第几天
    lastD = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    jd = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]
    if ((im > 0) and (im < 13)):
        if ((id > 0) and (id <= lastD[im - 1])):
            return jd[im - 1] + id
        else:
            print "Date not in month"
            return 0
    else:
        print "Month should be between 1 and 12 inclusive"
        return 0
#计算倾角的函数
def declin(day):
```

```
    rad = math.pi / 180
    xlong = 279.164 + 0.985647 * day
    xanom = 356.381 + 0.985600 * day
    g = xanom * rad
    elong = xlong + 1.915 * math.sin(g) + 0.02 * math.sin(2 * g)
    elong = elong * rad
    oblecl = 23.44 * rad
    return math.asin(math.sin(oblecl) * math.sin(elong)) / rad

#主程序
month = int(sys.argv[1])
date = int(sys.argv[2])
lat = float(sys.argv[3])
decl = declin(jdate(month, date))
sunangle = 90 - lat + decl
azimuth = 180
if sunangle > 90:
    sunangle = 180-sunangle
    azimuth = 0
gp.setparameterastext(3,str(sunangle))
gp.setparameterastext(4,str(azimuth))
```

我们在思考远一点，如何根据一天内的任何时间而不是正午时间，获取太阳角和方位角呢？

感兴趣的可以从本文开头的链接中下砸相应代码学习。

✧ **探索 2：**

如何在 PythonWin 里运行这个脚本？首先我们得明确几个点：①我们将把 hillshade 作为脚本的一部分使用，并为其提供输入参数：一个高程栅格（elevation raster）；②GetParameterastext 仅在用作工具时起作用。

下表为脚本工具的参数设置：

Display name	Data type	Type	Direction	Default
Month	Long	Required	Input	
Date	Long	Required	Input	
Latitude	Double	Required	Input	
Workspace	Workspace	Required	Input	
Elevation	Raster Dataset	Required	Input	
Hillshade	String	Required	Input	

代码如下：

```
import win32com.client, sys, math
gp = win32com.client.Dispatch("esriGeoprocessing.GPDispatch.1")
#函数定义，同上，下面仅给出函数名称：
def jdate(im, id):
def declin(day):
#  主程序，使用 sys.argv[]代替 getparameterastext()
month = int(sys.argv[1])
```

```
    date = int(sys.argv[2])
    lat = float(sys.argv[3])
    gp.Workspace = sys.argv[4]#输入时注意，路径应为反斜杠“\”
    elev = sys.argv[5]
    hillsh = sys.argv[6]#给输出 hillshade 指定文件名
    decl = declin(jdate(month, date))
    sunangle = 90 - lat + decl
    azimuth = 180
    if sunangle > 90:
        sunangle = 180-sunangle
        azimuth = 0
    gp.addmessage("about to try")
    try:
        gp.OverwriteOutput = 1
        gp.addmessage("after overwriteoutputs setting, " + gp.workspace + "/" +
hillsh)
        gp.CheckOutExtension("spatial")
        gp.addmessage(gp.workspace + "/" + hillsh)
        gp.HillShade_sa (elev, gp.workspace + "/" + str(hillsh), azimuth, sunangle)
        gp.addmessage("done with hillshade")
        gp.CheckInExtension("spatial")
    except:
    ##      print gp.getmessages()
        gp.addmessage(gp.getmessages())
        gp.CheckInExtension("spatial")
```

阅读代码发现，第一个输入参数不适用的 getparameterastext(0)，而是 sys.argv[1]，这是因为 getparameterastext()方法只在工具中起作用，而 sys.argv[]有同样的效果，索引从 1 而不是 0 开始，当然，这需要首先引用 sys 模块。这里我们直接指定输出文件在输入数据文件夹内，省去了 setparameterastext()方法，当然这个方法在 PythonWin 中也无法运行。

尝试输入参数如下图，得到右下结果：

5 在 PythonWin 里调试地理处理脚本

既然我们已经认真地学会了从 Python 中创建并运行地理统计工具，那么现在需要考虑如何调试我们的程序了。我们需要经常在 Python 和添加的地理处理系统引用之间调试程序。当一个地理处理工具运行失败后，我们需要从 Pythonwin 中得到一个丰富的消息，而不是“未知错误”。

5.1 调试选择和消息

Python 优于 AML 的优点之一是它有更好的调试方法，调试程序有很多选择，这里不能一一列举。

- **打印语句（Print statements）**

一开始就养成良好的调试方法是：将变量的当前值或脚本的处理过程打印在屏幕上。

比如，对之前的一段脚本加以修改：

```
import arcgisscripting,sys
gp=arcgisscripting.create(9.3)
gp.Workspace="c:/prog/hmbarea"
gp.overwriteoutput=1
gp.copyfeatures_management("streams/arc", "streams.shp")
gp.Toolbox="Analysis"
gp.buffer("streams.shp", "stbuff200.shp", 200)
print "Finished Buffer.Now Cliping..."
gp.Clip("geol.shp", "stbuff200.shp", "geolstr200.shp")
```

```
>>> Finished Buffer.Now Cliping...
Script 'E:\Python25\clipstreams200.py' returned exit code 0
```

可以看出成功运行脚本！

然而当在工具中运行时，print 语句不会产生错误，但也不会输出任何东西，因此，我们用 gp.addmessage("Finished Buffer.Now Cliping...")代替 print 语句。那么，如果想无论在工具中或 Pythonwin 中都可以显示消息，就可以这两句都写上。我喜欢的做法是定义一个‘sendmsg’函数来输出消息：

```
def sendmsg(msg):
    print msg
    gp.addmessage(msg)
....
sendmsg("Finished Buffer.Now Cliping...")
```

- **获取工具消息和（try:…except:）**

上面的方法很有用，但当我们运行我们并不了解很多信息的地理处理工具时就显得无能为力了。如果在 Pythonwin 中运行 Buffer 时出现错误，比如输入文件不存在等，只能看到“未知错误”的提示，这就引出了 GetMessage 上的方便！

现在我们要体验两种调试技巧：①GetMessages 地理处理方法；②Python 语言的错误处理程序：try…except。

① 现在我们添加 GetMessages 查看错误信息：

还是上面的代码，把“streams.shp”修改为“stream.shp”，查看错误信息：

```
gp.Toolbox="Analysis"
gp.buffer("stream.shp", "stbuff200.shp", 200)
gp.Clip("geol.shp", "stbuff200.shp", "geolstr200.shp")
```

运行之，查看错误信息，然后修改代码如下：

```
try:
    gp.Toolbox="Analysis"
```

```
        gp.buffer("stream.shp", "stbuff200.shp", 200)
        gp.Clip("geol.shp", "stbuff200.shp", "geolstr200.shp")
    except:
        print gp.GetMessages()
```

GetMessages():所以信息
GetMessages(0):只显示消息
GetMessages(1):只显示警告
GetMessages(2):只显示错误

运行可以看到以下错误提示：

顺便说一下 GetMessages 的集中形式和代表的含义

```
Start Time: Mon Mar 28 13:58:28 2011
Failed to execute. Parameters are not vali
ERROR 000732: Input Features: Dataset stre
supported
WARNING 000725: Output Feature Class: Data
already exists.
Failed to execute (Buffer).
End Time: Mon Mar 28 13:58:28 2011 (Elapse
```

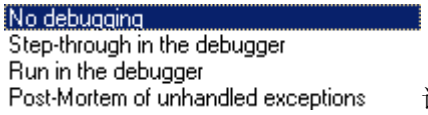
② 运行下面简单的代码，并改变 x 的值为非零，查看结果：

```
x=0.
y=15.
try:
        print y/x
 except:
        print "错误！0 不能是被除数！"
```

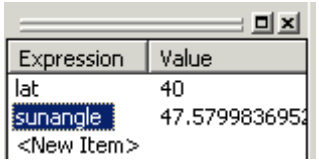
5.2PythonWin 的调试工具

PythonWin 提供了一些工具：①单步执行代码②插入断点③观测变量，或其他事情。这些都可以在 PythonWin 的帮助系统里找到相关示例教程。这里我们演示单步执行代码的过程。

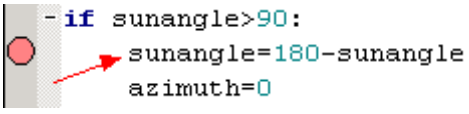
以前面用过的太阳角计算脚本为例，点击运行之后，在 Debugging 下拉框里看到



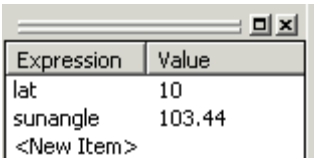
调试选项。选择“Step-through in the debugger”。然后，一系列调试工具出现  ；一个黄色三角出现在第一行代码前。按下  （Step over）按钮进入下一行，继续执行；点击  （Watch）按钮，出现，在 Expression 下<New Item>里输入 lat 等可以查看当前变量值



；当你遍历完程序，点击  关闭调试；尝试添加一个断点，当定位到某一行时，在



点  ，然后运行此脚本在“Run in the debugger”下，发现没有任何影响，但是修改第一行为“lat=10”，运

行发现在断点处停止， 

，点击  按钮继续程序。得到结果“Noon sun angle=76.56 Azimuth=0”。

5.3 地理处理工具举例

- **转换脚本（conversion script）**

既然我们已经会在帮助系统里寻找答案，也掌握了几种程序调试方法，现在让我们建立一个脚本。

① 添加引用

```
import arcgisscripting
```

② 创建“sendmsg”函数

```
def sendmsg(msg):
    print msg
    gp.addmessage(msg)
```

③ gp.OverwriteOutput=1

④ 使用 Data Management toolbox 中的 Workspace 工具集中的 CreateFolder 工具在“c:/prog”中创建新文件夹“woodside”

```
gp.CreateFolder_management("c:\prog","woodside")
```

⑤ 设置 gp 的工作文件夹为 woodside

```
gp.workspace="c:\prog\woodside"
```

⑥ 使用 Conversion 里的 To Raster 工具集中的 DEMtoRaster 工具将 c:/prog/pendata 下的 woodside.dem 转换为 woodelev 栅格数据，不要提供任何可选参数值

```
gp.DEMtoRaster_conversion('c:\prog\pendata\woodside.dem','woodelev')
```

⑦ 在第一行前添加 try:，最后一行后添加 except:，最后加上，sendmsg(gp.GetMessages()).

- 要素转换为栅格（feature to raster conversion）

相信有了前面的联系，本例很简单了，将 d:/prog/pendata/landuse(polygon)根据“LU-CODE”字段转换为“lugrid”栅格数据，分辨率为 30。

```
try:
    import arcgisscripting
    def sendmsg(msg):
        print msg
        gp.addmessage(msg)
    gp.OverwriteOutput=1
    gp.workspace="c:\prog\pendata"
    gp.FeatureToRaster_conversion('landuse/polygon','LU-CODE','lugrid','30')
except:
sendmsg(gp.messages())
```

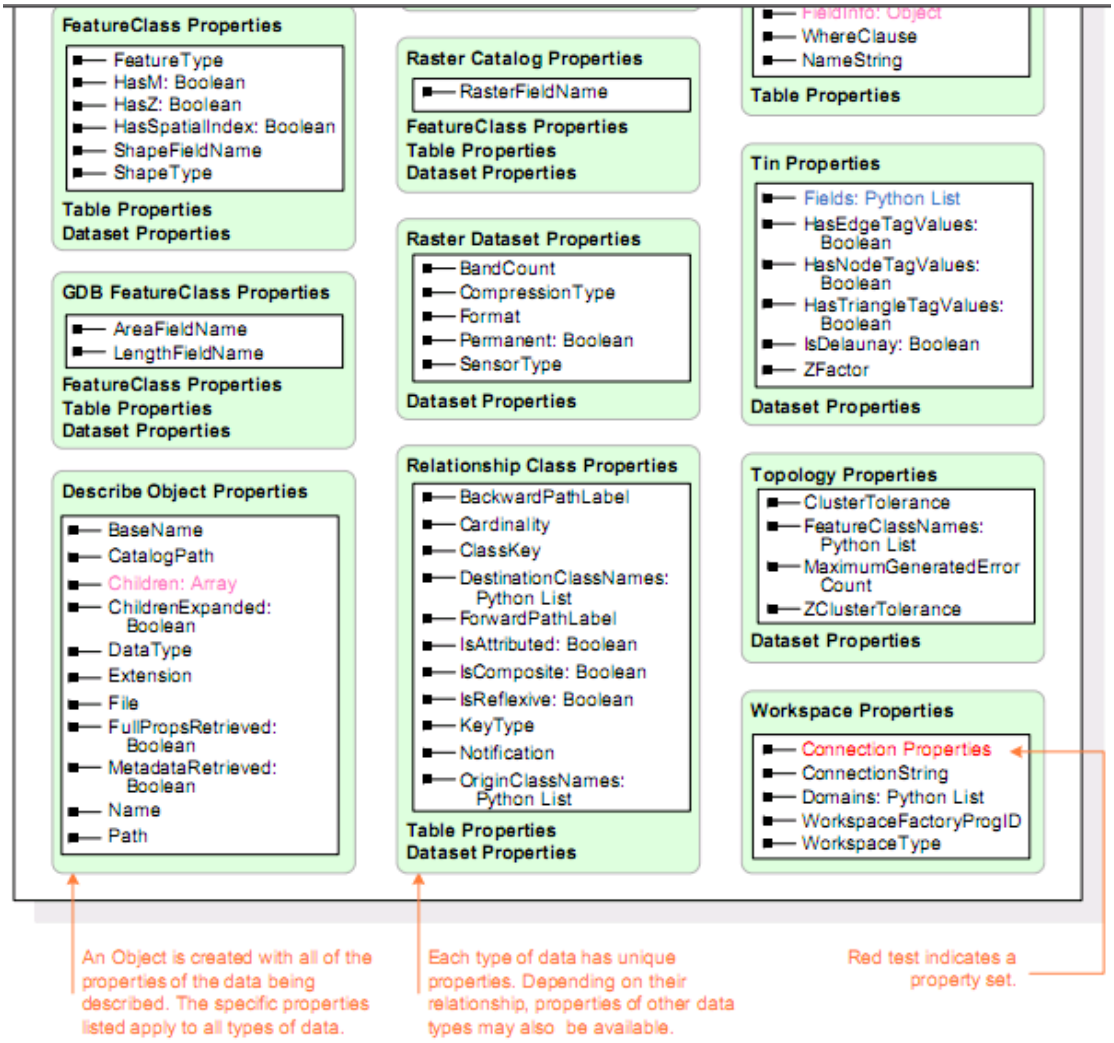
6 使用描述（Describe）和存在（Exists）获取数据信息

有很多情况下我们需要使用某个 GIS 数据的特征去处理其他数据。比如，在栅格运算中，

我们可能想用一个数据的边界去界定另个数据集，很像裁剪操作，或者检查一个数据的拓扑错误。

6.1 描述

gp.Describe 方法适用于生成一个数据集的若干属性。Geoprocessor Programming Model 中的绿色区域部分，有按等级划分的很多属性：所有数据都有一个数据类型（DataType）和目录路径（CatalogPath）。FeatureClass 属性包括所有表个属性和数据集属性；栅格数据属性包括栅格波段属性和数据集属性，所以数据集属性在一定程度上是共有的（Feature Class、Rasters、Coverages）。



- 利用描述系统用一个数据边界裁剪另一个栅格数据，将裁剪后的土地利用图存在 woodside 文件夹下。

```
import arcgisscripting
gp = arcgisscripting.create(9.3)
gp.workspace="c:\prog\woodside"
dscRas=gp.Describe("woodelev")
#print dscRas.Extent
envel=str(dscRas.Extent.XMin)+''+str(dscRas.Extent.YMin)+''+str(dscRas.Extent.XMax)+' '+str(dscRas.Extent.YMax)
```

#print envel 因为从 9.3 版本之后，Extent 是一个 Object，所以直接使用 dscRas.Extent 在 clip 语句中会出现错误，而 clip 语句的第二个参数 Rectangle 是一个 Envelope 类型数据，所以这里我使用了 envel 接收 Extent 的四个值。

```
try:
    gp.clip_management("c:\prog\pendata\lugrid",envel,"luwood")
except:
print gp.GetMessages()
```

- 编写一个脚本输出“c:/prog/hmbarea/tmcomp.bil”的波段个数，命名为 countBands.py。在栅格数据集属性中寻找这个方法：

```
import arcgisscripting, sys
gp = arcgisscripting.create(9.3)
gp.workspace = "c:/prog/hmbarea"
dta = "tmcomp.bil"
dsc = gp.Describe(dta)
print dta + " " + dsc.DataType + ": " + str(dsc.BandCount) + " bands."
```

运行结果：“**tmcomp.bil RasterDataset: 3 bands.**”

6.2 存在（Exists）

关于一个数据集的一个更基本的信息是其是否存在。测试一个数据是否存在也可帮助我们避免错误。一个非常有用的技术是通过 gp.Exists 判断一个特定的数据是否存在。

gp.Exists 能用在各种类型数据上，比如在使用数据之前添加一个判断是否存在的语句：

```
if gp.Exists(“streams.shp”)
    gp.Buffer(“streams.shp”,”stbuff200.shp”,200)
```

在下面的代码中我们将使用这种方法判断数据是否存在，其可以成为 overwriteoutput 设置的替代语句。

- 判断一个字段属性是否存在。上面的方法无法测试一个数据的字段是否存在，但线面的 listFields 枚举是个技巧：(listFields 返回一个 Python 属性对象列表)

```
if not gp.listFields(“streams.shp”,”st-code”):
```

我们将在后面使用这句判断字段是否存在。

6.3 在循环中使用描述和存在

这会让你感到脚本在处理多种数据方面的强大作用。我们将裁剪一整系列的栅格数据，并存在新文件夹内。在这个处理过程中，我们也会在创建输出数据之前用“存在工具”中一个很方面的方法检查其是否已经存在。在本例中，我们使用了 ArcInfo Workspace，并把格网栅格存在这里，当然你也可以将他们存在 Geodatabase 中。

代码如下：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
#gp.checkextension("spatial")
gp.workspace="c:/prog/hmbarea"
if not gp.Exists(gp.workspace+"/cities_polygon.shp"):

gp.FeatureclassToShapefile_conversion("c:/prog/pendata/cities/polygon",gp.workspace)
#因为没有找到 citypoly.shp 和 city 栅格数据，所以我使用 pendata 文件夹下的 cities 里的 polygon 转换的 shp 文件取代。
else:
    print gp.workspace+"/cities_polygon.shp exists"
if not gp.Exists(gp.workspace+"/cities"):

gp.PolygonToRaster_conversion("cities_polygon.shp","CITY_CODE","cities","","","60")
else:
    print gp.workspace+"/cities exists"
if not gp.Exists(gp.workspace+"hmbcity"):
    gp.createArcInfoWorkspace(gp.workspace,"hmbcity")
else:
    print gp.workspace+"/hmbcity exists"
try:
    gp.mask="cities"
    gp.checkoutextension("spatial")#检查有 spatial 的许可
    raslist=gp.listrasters()
    for ras in raslist:
        outputname=gp.workspace+"/hmbcity/"+ras
        print outputname
        if  not gp.exists(outputname):
            try:
                gp.ExtractByMask_sa(ras,gp.mask,outputname)#用掩膜裁剪
            except:
                gp.getmessages()
except:
    gp.getmessages()
```

探索 1：selSanMateoSel.py

用 Select_Analysis 创建 San Mateo（`city_code = 26`）城市的 shp 文件。创建一个小的工作空间命名为“Sanmateo”，然后用 SanMateo.shp 裁剪 pendata 文件夹下文件名以‘g’开头（“g*”）的 coverage 文件的 polygon，存入此空间。建议：你需要使用 gp.listdatasets(“g*”)代替 gp.listfeatureclasses，因为你需要寻找 coverage，在你处理列表里成员时，可以将成员名称赋给变量 f，然后用 gp.exists(f+”/polygon”)查找多边形要素集。你也会发现 gp.CreateFolder_management 很有用，你也可以裁剪所有数据时设置通配符为“*”，或者不适用通配符。

代码如下：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.overwriteoutput=1
gp.workspace="c:/prog/pendata"
workspaceLoc="c:/prog"
try:
    gp.Select_Analysis("cities/polygon","San_Mateo.shp",'"CITY-CODE"=26')
    wsName="San_Mateo"
    newWS=workspaceLoc+"/"+wsName
    if not gp.exists(newWS):
        gp.CreateFolder_management(workspaceLoc,wsName)
    fList=gp.listdatasets("g*")
    for f in fList:
        if gp.exists(f+"/polygon"):
            gp.Clip_analysis(f+"/polygon","San_Mateo.shp",newWS+"/"+f)
except:
    print gp.getmessages()
```

探索 2：multiFeatureRas.py

创建两个 Python 列表：

```
cov=[“geology”,”landuse”,”publands”,”flood”,”streams”,”roads”,”vegetation”]

fld=[“TYPE-ID”,”LU-CODE”,”PUB-CODE”,”FLOOD-CODE”,”ST-CODE”,”ROAD-CODE”,”VEG-CODE”]
```

将每个 coverage 根据对应的字段转换为栅格，分辨率设置为 60。

代码如下：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.workspace="c:/prog/pendata"
cov=["geology","landuse","publands","flood","streams","roads","vegetation"]
fld=["TYPE-ID","LU-CODE","PUB-CODE","FLOOD-CODE","ST-CODE","ROAD-CODE","VEG-CODE"]
gp.overwriteoutput=1
if not gp.Exists("multiFeatRas"):#新建一个文件夹保存处理后数据
    gp.CreateFolder_management(gp.workspace,"multiFeatRas")
for i in range(len(cov)):#遍历 coverage 数据
    if gp.Exists(cov[i]+"/polygon"):
        covTop=cov[i]+"/polygon"
    elif gp.Exists(cov[i]+"/arc"):
        covTop=cov[i]+"/arc"
    gp.FeatureToRaster_conversion(covTop,fld[i],"multiFeatRas/"+cov[i]+"g","60")
```

7 在 Python 脚本中使用地图代数（Map Algebra）

将 AML 程序移植到 Python 中唯一令我犹豫的是它不能像在 AML 中那样编写地图代数语句来处理格网数据。但是，有一种方法可以很好地解决这个问题——简单修改一个定义好的函数。Spatial Analyst 工具下的 MultiOutputMapAlgebra 工具允许使用完整的地图代数赋值语句，包括嵌套结构（!）。唯一的遗憾是这工具名字太长，所以我已经把它嵌入一个自定义函数中，命名为“ma”，当然函数中包括开始和结束“Spatial”扩展功能。下面的例子说明了这种用法，虽然只有一点语句，但包括了在处理多工具中的两个棘手的问题。

usingMapAlgebra.py

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
def ma(expression):
    gp.checkoutextension("spatial")
    gp.MultiOutputMapAlgebra_sa(expression)
    gp.checkinextension("spatial")
gp.workspace="c:\prog\hmbarea"
gp.overwriteoutput=1
if not gp.exists("streamsg"):
    gp.FeatureToRaster_conversion("streams/arc","st-code","streamsg","60")
gp.CellSize=60
ma("slopeg=slope(elev,0.3048)")
ma("stream2=con(isnull(streamsg),0,streamsg)")
 ma("histreams=( elev > 1000) and stream2 ")
```

另外一个方法是直接将方法赋值给变量，这样通过下面简单的一句话就可以实现前面所讲的自定义函数了：

```
ma=gp.MultiOutputMapAlgebra_sa
```

但也要确保在初始化 ma 变量之前检查“Spatial”扩展功能，安全的编程习惯是你在 Except: 后添加 gp.CheckInExtension(“spatial”)，这是为了避免在你尝试 CheckOutExtension 时，其已经被检查过了，如下所示：

```
gp.checkoutextension("spatial")
ma=gp.MultiOutputMapAlgebra_sa
ma("slopeg=slope(elev,0.3048)")
ma("stream2=con(isnull(streamsg),0,streamsg)")
ma("histreams=( elev > 1000) and stream2 ")
gp.checkinextension("spatial")
```

8 数据管理和指针（Data Management and Cursors）

8.1 数据管理（Data Management）

处理表格数据、创建字段来储存这些数据或者其他数据操作时 GIS 中非常重要的工作。的那个你需要做一连串数据管理和分析时，脚本是一个很好的选择。在一些情况下，我们需要处理数据字段时有一系列工具可以使用，比如，“Add Fields”、“Calculate values for fields”、“delete fields”以及通过一个公共字段链接表格获取附加数据字段。其中有些工具会利用不同的统计方法将输入数据输出为新的表格。其他情况下，我们需要处理行数据，比如栅格数据中单个要素或栅格数据的属性值，接下来我们学习使用指针（Cursor）逐个处理行数据。

首先，我们看一下一些能够在处理表格数据（主要是包括属性字段）时用到的工具。有些工具根据属性选择记录，或复制或删除选中记录。

Toolbox	Tool	做什么？	Output
Analysis	frequency	频率统计	Table
	statistics	总结统计	Table
	select	利用 where 子句选择要素	新要素集
	table_select	用 SQL 语句选择并提取选中的属性	Table
Data Management Alias:*management*	addField	添加字段	表格中字段
	calculateField	通过表达式为字段赋值	已存在字段的值
	deleteField	删除字段	
	addXY	为点要素添加 X&Y 值	X&Y 值
	selectLayer By Attribute	对图层或表格通过属性查询选择、更新或移除选择	
	copyFeatures	复制选中的要素	新要素集
	deleteFeatures	删除选中的要素	
	addJoin	根据一个共同的字段将一表格连接到图层（或表格）	连接关系
	removeJoin	移除已存在连接	
Coverage Alias:*Arc*	select(reselect)	根据逻辑表达式提取 Coverage 地图要素	输出 Coverage（包含属性表）
	additem	为信息表添加属性字段	
	dropitem	从信息表中移除属性字段	

	joinitem	根据相关属性融合数据表	Table
	addxy	为 point、label、node 表格添加 x、y 字段	字段添加到已存在表格

- 创建新脚本（addCalcField.py），添加字段“elevm”到“contours.shp”（surf_bld 文件夹内），并计算其值为“[elev]*0.3048”。

代码如下：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.workspace="c:/prog/surf_bld"
gp.toolbox="management"#设置 gp.toolbox 为“management”。
try:
    if not gp.listfields("contour.shp","elevm"):#判断此字段是否已经存在
gp.addField("contour.shp","elevm","float")

gp.calculateField("contour.shp","elevm","[elev]*0.3048")
    except:
    print gp.GetMessages(2)
```

```
>>> 100 30.4799995422
110 33.5279998779
120 36.5760002136
130 39.6240005493
140 42.672000885
90 27.4319992065
```

- 创建新脚本（addXY.py）使用 management 里的 addxy 工具为 Marbes 文件夹下的“samples.shp”添加 x&y 值。

代码如下：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.overwriteoutput=1
gp.workspace="c:/prog/marbles"
gp.addxy_management("samples.shp")
```

8.2 指针（Cursors）

指针给了一种访问数据值的通道，允许遍历数据表格中的所有记录。由于记录（行数据）可能是一个矢量要素或一栅格值，所以，指针在处理数据是十分强大。

指针有三种类型：

- SearchCursor：读取一行中的值
- InsertCursor：插入新的行
- UpdateCursor：改变行中值以及删除行

- 尝试下面的代码，使用了 SearchCursor 来显示 addCalcField.py 计算的结果：

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
gp.workspace="c:/prog/surf_bld"
cur=gp.SearchCursor("contour.shp")
row=cur.Next()
while row:
    print row.elev,row.elevm
```

```
row=cur.Next()
```

- 尝试一下代码，介绍了读取和显示“stream.shp”内所有顶点，不要忘记 while 循环内 Next()语句，否则会出现无限循环！也展示了几何要素的用法！

```
import arcgisscripting
gp=arcgisscripting.create(9.3)
rows=gp.SearchCursor("c:/prog/surf_bld/stream.shp")
row=rows.Next()
while row:
    feat=row.shape
    a=0
    while a<feat.PartCount:
        stArray=feat.GetPart(a)
        stArray.Reset
        pnt=stArray.Next()
        while pnt:
            print str(pnt.id)+";"+str(pnt.x)+";"+str(pnt.y)
            pnt=stArray.Next()
        a=a+1
row=rows.Next()
```

- 尝试下面代码，展示了 InsertCursor 的用法，同样的基本方法可用来读取外部数据，比如文本文件。

```
import win32com.client, sys, math, string
gp = win32com.client.Dispatch("esriGeoprocessing.GPDispatch.1")
#import arcgisscripting
#gp=arcgisscripting.create(9.3)   也可以这样引用
gp.workspace = "c:/prog/Marbles"
try:
    cur = gp.InsertCursor("mvalley_pts.shp")
    feat = cur.NewRow()
    feat.id = 12
    feat.Name = "Sky High Lake Camp"
    pnt = gp.CreateObject("Point")
    pnt.x = 485339
    pnt.y = 4600001
    feat.shape = pnt
    cur.InsertRow(feat)
    del cur
except:
    print gp.getmessages()
if cur: del cur
```

挑战 1：

下面的代码可以“打开”、“写入”、“关闭”一个文本文件，修改上面的脚本输出节点值信息。(对“stream.shp”的操作)

```
import arcgisscripting,os,sys
gp=arcgisscripting.create(9.3)
wspath="c:/prog/surf_bld"
txtfile="strout.txt"
fpath=wspath+"/"+txtfile
if gp.exists(fpath):os.remove(fpath)
txtfile=open(fpath,"w")
gp.workspace=wspath
rows=gp.SearchCursor("stream.shp")
row=rows.Next()
while row:
    feat=row.shape
    a=0
    while a<feat.partCount:
        geomArray=feat.getpart(a)
        fid=row.GetValue("ID")
        geomArray.Reset
        pnt=geomArray.Next()
        while pnt:
            txtfile.write(str(fid) + "; " + str(pnt.x) + "; " + str(pnt.y)+"\n")
            pnt=geomArray.Next()
        a=a+1
    row=rows.next()
txtfile.close()
```

挑战 2：

写一个可以读取刚才输出的文本文件并且创建一个新的 shp 文件的脚本。

如果你使用的是 ArcGIS10，这里有最新的帮助，如果使用的是 ArcGIS9.3.1 或 9.3，这里有相应帮助。

```
import arcgisscripting,sys,fileinput
gp=arcgisscripting.create(9.3)
inFile="c:/prog/surf_bld/strout.txt"
gp.workspace="c:/prog/surf_bld"
fcName="newsamp.shp"
gp.overwriteoutput=1
gp.CreateFeatureClass_management(gp.workspace,fcName,"point")
try:
    cur=gp.InsertCursor(fcName)
    for line in fileinput.input(inFile):
        values=line.split(";")
        id=int(values[0])
        x=float(values[1])
        y=float(values[2])
        feat=cur.NewRow()
```

```
            pnt=gp.CreateObject("Point")
            pnt.x=x
            pnt.y=y
            feat.shape=pnt
            feat.id=id
            cur.InsertRow(feat)
            print id,x,y
        del cur
        fileinput.close()
    except:
        print gp.getmessages()
        if cur:
            del cur
```

接下来该做什么呢？

我们已经做了很多，但是大多数都是创建处理特定事情的部分，比如遍历一系列数据。但是到目前为止我们并没有花太多时间在这些循环上。

另一间我们没有花太多时间处理的事情是，让脚本工具更好地工作。输入和输出是这个处理过程中有挑战性的部分，所以在这方面下手可能会更适合你。我已经发现这是最令人困惑的部分，所以我将能发现的尽可能多的输入/输出情况做了总结，请看附录 1。

最后一个建议是在模型中使用你的脚本工具，这是一个非常值得和有用的工作。

附录 1：地理处理脚本中输入&输出方法指南

1、 输入数据集（Datasets）
   - 脚本：使用 GetParameterAsText 或 sys.argv
   - 工具参数：（要素类、栅格数据或图层等）必需的输入
2、 输出数据集（Datasets）（如，Spatial Statistics/测量地理分布里的平均中心）
   - 脚本：使用 GetParameterAsText 或 sys.argv
   - 工具参数：（要素类、栅格数据等）必需的输入
3、 输出值和字段（如，Spatial Statistics/Utilities 下的计算面积）
   - 脚本：输入和输出的数据集、复制输入到输出
   - 工具参数：输入要素类、输出要素类
4、 输出单个值（如，Spatial Statistics/Analyzing Patterns 下的空间自相关）
   - 脚本：SetParameterAsText
   - 工具参数：（Long、Double 等）获取的输出
5、 输入和输出表格或要素类（如，Management/Fields 下的 Transpose Time fields）
   - 脚本：GetParameterAsText
   - 工具参数：输入—表格视图，必需的输入；输出—表格，必需的输出
6、 修改输入到输出，没有找到简单的例子，除了 Mosaic（Samples/Conversion/Raster）
   - 脚本：输出用 GetParameterAsText ；类似的 SetParameterAsText

- 工具参数：输出是必须的输入，然后作为获取的输出，设置从原始处获取

7、 批处理（如，Management/Projections 和 Transformations/Feature 下的 Batch Project）
- 脚本：输入和输出用 GetParameterAsText，获取的输出不需要
- 工具参数：输入地理数据集，必需的输入，允许多值；输出工作空间，工作空间或要素数据集，必需的输入；输出空间的单独的获取的输出也是一样

8、 如果成功完成则返回 True（如，Management/Raster 下的 Batch Calculate Statistics）
- 脚本：nothing
- 工具参数：布尔型参数，获取的输出默认为 True

什么时候用 Derived（获取的，衍生的）？如果工具更新输入到工具里，或者如果你需要一个输出并用在一个模型里，但是你不想让输出参数在脚本对话框中出现。

附录 2：其他

1、 工作空间和临时工作空间设置

支持“临时工作空间”环境设置的工具可将指定的位置用作输出数据集的默认工作空间。“临时工作空间”专门用于存放不愿保留的输出数据。

“临时工作空间”环境的主要用途是供“模型构建器”使用。“模型构建器”需要使用一个工作空间来写入中间数据集（模型运行后便不再使用）。尽管它主要服务于“模型构建器”，但有时可能也需要为各工具对话框设置临时工作空间。

使用工具对话框时，输出数据集名称会按照当前工作空间环境和临时工作空间环境设置自动生成。

2、 Python 和 PythonWin 的版本

不要急于升级你的 Python 或 PythonWin 版本。最好是使用随 ArcGIS 安装的版本，而且 PythonWin 不是自动安装的，在前文中有介绍。