



Lesson 2: Fundamentals of Python

Basic Programming in ArcGIS with Python
Workshop

RS/GIS Lab, Utah State University
With Material from ESRI



Learning Objectives:

- Learn basics of Python syntax
- Learn about importing modules in Python
- Learn about the ArcObject concept
- Understand the concepts of Properties and Methods
- Learn where to find syntax for ArcToolbox tools
- Understand why ArcToolbox aliases are important
- To be able to edit/write a very simple Python script



Lesson 2a: The Basics of Python Syntax

- Comment: non-executable line of code
 - One or two number signs (# or ##)

```
# Set the current workspace  
gp.workspace = "C:/john/Lesson2"  
## Set so output can be overwritten  
gp.Overwriteoutput = 1
```
- Comment and uncomment blocks of code
 - Right-click the line of code > Source Code > Comment out region
 - Useful for debugging code



Variable: a named entity that holds information

- Can hold different types of data
 - Strings, numbers, lists, tuples, dictionaries, files

```
# Local variables...
```

```
fg = "C:/john/Lesson2/fl_grd"
```

- No declaration key word or type assignment, Python smart enough to know `fg` holds a string
- Variables are case sensitive (below are *two different* variables):

```
Num = 1000
```

```
num = 2000
```



Strings: text or characters

- Strings surrounded by double (") or single (') quotes
- Pathnames use two back (\\) or one forward (/) slash

```
fg = "C:/john/Lesson2/fl_grd"
```

```
fg = "C:\\john\\Lesson2\\fl_grd"
```

- Strings can be concatenated (combined together) with + character

```
fgPath = "C:/john/Lesson2/"
```

```
fg = "fl_grd"
```

```
fullPath = fgPath + "/" + fg
```

- Strings are indexed (zero-based) you can "fetch" characters/items

```
fl = "Flowline.shp"
```

```
fl[0] # fetches the first item (i.e. "F")
```

```
fl[1:3] # fetches from 1 up to, but not including, 3 (i.e. "lo")
```

```
fl[:-4] # fetches from 1 up to, but not including the last  
# 4 items ( i.e. "Flowline")
```



Numbers and Lists:

- Variables can hold numbers and expressions

```
num1 = 2.3
```

```
num2 = 2 + 3
```

- Variables can hold lists

```
numList = [1, 2, 3, 4, 5]
```

```
grdList = ["elev30m", "elev30m10", "elev30m100", "elev30m250"]
```

- Lists are indexed (zero-based)

```
# create a list
```

```
grdList = ["elev30m", "elev30m10", "elev30m100", "elev30m250"]
```

```
# fetch the first item (i.e. "elev30m")
```

```
grd1 = grdList[0]
```

```
# fetch the second item (i.e. "elev30m10")
```

```
grd2 = grdList[1]
```

```
# fetch from first item up to second item
```

```
grd3 = grdList[0:1]
```

```
# fetch from 1 up to, but not including the last item
```

```
grd4 = grdList[0:-1]
```

```
# fetch third through the end
```

```
grd5 = grdList[2:]
```



Variable Naming Conventions (ESRI):

- Use combination of lower and upper case, beginning with lowercase
- Acronym at the beginning, use lowercase, followed by descriptive part with uppercase

```
fcFlowline = "flowline.shp"
```

```
grdElev = sys.arg[3]
```

- Avoid special characters (e.g. / \ & * !)
- Examples of not-so-good variable names

```
- Var1, a, temp-var, b
```



Line Continuation:

- Line continuation characters
 - Backslash \
 - Parentheses () and brackets []
- Indentation is automatic

```
# Process: Polyline to Raster...
```

```
gp.PolylineToRaster_conversion(flowline, "FID", fl_grd, \  
                                "MAXIMUM_LENGTH", "NONE", "30")
```

```
# Create a list
```

```
fcList = ["streams", "rivers", "ownership", "soils",  
          "roads", "springs", "hazards"]
```



Built-in Python Functions:

- `len()` – returns the length of a string or a list

```
fc = "flowline.shp"
```

```
len(fc) → returns 12
```

```
grdList = ["elev30m", "elev30m10", "elev30m100"]
```

```
Len(grdList) → returns 3
```

- `round()` – returns a rounded number

```
xCoord = 435690.435676
```

```
round(xCoord) → returns 435690
```

- `str()` – converts an integer object to a string

```
zone = 1
```

```
zonestr = str(zone)
```

```
print "slice"+zonestr
```



Importing Modules for Additional Functions:

- The math module:

```
import math  
math.sqrt(49) → returns 7  
math.pow(10,2) → returns 100
```

- The string module:

```
import string  
string.split("4 5 9 13") → returns ['4', '5', '9', '13']  
string.upper("c:\\john") → returns 'C:\\JOHN'
```

- Other Python modules:
 - sys, os, arcgisscripting, gdal



Python Statements:

- `import`—imports a module

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

- `print`—prints to the Interactive Window

```
print "buffer done!"
print "dataset does not exist"
```

- Other statements:

```
if...elif...else
while
for...in
try...except
```



Decision Making Syntax:

- Testing conditions

```
if y == 1:  
    print "y is 1"  
elif y == 2:  
    print "y is 2"  
else:  
    print "y is neither 1 nor 2"
```

- Notes:

- Colons used at end of each condition
- Indention defines what executes for each condition
- Python automatically indents
- One equal sign (=) for assignment, two (==) for conditions

```
y = 48          # assignment  
If y == 48:    # testing a condition
```



Looping Syntax:

- While loops:

```
y = 1
while y < 15:
    print y
    y = y + 1
```

- Counted loops:

```
for y in range (1,15):
    print y
```

- List loops

```
numList = [1, 2, 3, 4, 5]
for num in numList:
    print num
```

- Notes:

- Colons at end of each condition (implies “do this”)
- Indentation important for proper execution

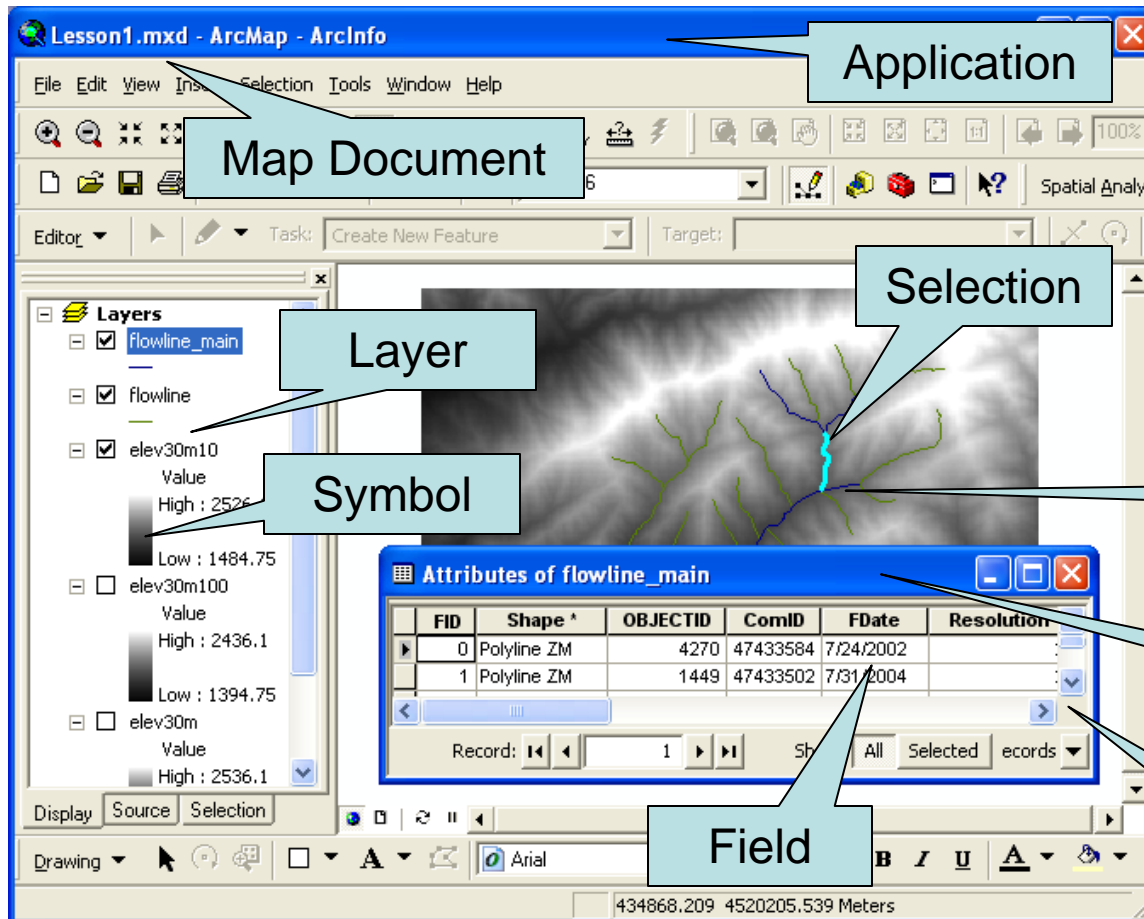


Class Activity: Learning Python Syntax

- As a class we'll experiment with what we've learned about Python syntax.
- Start in the Interactive Window, then move to writing simple scripts in the Script Window



Lesson 2b: Accessing Tools and Environment Settings



- ArcGIS is built with hundreds of ArcObjects



Interacting with ArcObjects (Object Oriented Programming)

- Each ArcObject has Properties and Methods
 - Property: Characteristic of an object
 - Method: Something the object knows how to do (action)
- We interact with objects through properties and methods

Map Document
Properties
-Layer count
-Name
-Spatial reference
-Map scale
-Extent
Methods
-Add layer
-Clear selection
-Select feature

Feature Class
Properties
-Shape type
-Spatial reference
-Extent
Methods
-Create feature
-Remove feature

The Geoprocessor ArcObject:

- Most geoprocessing “sits on” one ArcObject called the Geoprocessor Object
- The geoprocessor object has many properties and methods
- The geoprocessing object allows us to do geoprocessing

Geoprocessor	
Properties	
-Current workspace	}
-Map extent	
-Cell size	
Methods	
-Buffer	}
-Clip	
-Select	
-Copy features	
-Select feature	

Env.
Settings

Tools



Accessing the Geoprocessor with Python:

- The gp object can be used in many languages
 - Perl, VBScript, Jscript, Python, VBA, VB, etc.
 - Any COM (Component Object Model) compliant language
- In Python, objects are accessed through dot notation

```
# Import the arcgisscripting module
```

```
import arcgisscripting
```

```
# Create the Geoprocessor object (assign gp variable name)
```

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

- Thereafter all geoprocessing functionality is accessed through the **gp** object (or gp variable)

```
# Check out the spatial analyst extension license
```

```
gp.CheckOutExtension("spatial")
```



Syntax for properties and methods:

- To assign a value to a property:

- Object.Property = Value

```
gp.Workspace = "C:/john/Lesson2"
```

- To get a value of a property:

- Object.Property

```
Print "The workspace is" + gp.Workspace
```

- To use a method:

- Object.Method(arg, arg, ...)

- All methods require parentheses even if there are no arguments

- Arguments are separated by commas

```
gp.Thin_sa("ingrd", "thin_grd", "ZERO", "NO_FILTER",  
"ROUND", "30")
```

Finding Scripting Syntax for Environment Settings or Tools:

- ArcGIS Desktop Help:
 - Contents > Geoprocessing tool reference
 - Index and Search Tabs
- In ArcToolbox (my preferred method):
 - Right-click tool, then choose Help
- Usage in Python (in Interactive Window), type:

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

```
gp.Usage("Thin_sa")
```

```
'Usage: Thin_sa <in_raster> <out_raster> {ZERO | NODATA}  
      {NO_FILTER | FILTER} {ROUND | SHARP} {maximum_thickness}'
```



Toolbox Aliases:

- Many tools in ArcToolbox with the same name
 - Coverage Tools > Analysis > Clip
 - Analysis Tools > Extract > Clip
- Always suffix the tool with the toolbox alias name
- Always assign alias names to custom toolboxes

Toolbox	Alias Name
3D Analyst	3d
Analysis Tools	analysis
Cartography Tools	cartography
Conversion Tools	conversion
Coverage Tools	arc
Data Management Tools	management
Geocoding Tools	geocoding
Linear Referencing Tools	lr
Spatial Analyst Tools	sa
Spatial Statistics	stats

```
Usage: Thin_sa <in_raster> <out_raster> {ZERO | NODATA}  
{NO_FILTER | FILTER} {ROUND | SHARP} {maximum_thickness}
```



Example: The Select Tool

- Syntax:

Select_analysis (in_features, out_feature_class, where_clause)

- Example:

```
# gp.select_analysis("nfroads.shp", "paved.shp", '  
"ROAD_CLASS" = \'PAVED\')
```

- Notes:

In the where_clause for the select tool, fields are double quoted (") and text values are single quoted ('). In Python strings are identified by single or double quotes. This can become problematic with Python, so we use the escape character \ to tell Python to ignore the single quotes used by the where_clause.



Example: The Polyline to Raster Tool

- Syntax:

`PolylineToRaster_conversion (in_features, value_field, out_raster_dataset,
cell_assignment, priority_field, cellsize)`

- Example:

```
gp.PolylineToRaster_conversion(InFeatures, "myField", OutRaster,  
"MAXIMUM_LENGTH", "", "")
```

- Notes:

Can use "" to skip an optional argument



Example: TheThin Tool

- Syntax:

Thin_sa (in_raster, out_raster, background_value, filter, corners, maximum_thickness)

- Example:

```
gp.Thin_sa(inRaster, outRaster, "ZERO", "NO_FILTER", "ROUND",  
"30")
```

Or

```
gp.Toolbox = "sa"
```

```
gp.Thin (inRaster, outRaster, "ZERO", "NO_FILTER", "ROUND",  
"30")
```

- Notes:

Can use `gp.Toolbox = "sa"` instead of toolbox alias



Assignment 2b: Write a simple Python Script

- Write a script that performs the following geoprocessing steps
 - Selects a stream line from a vector shapefile.
 - Buffers the stream by 100 feet.
 - Rasterizes the buffer area.
- Follow the step-by-step instructions in the handout. Think about what you're doing. If you have questions, ask.



Assignment 2c: Write a Python Script with a Loop

- Start with the code provided in lesson2c.py. Some comments are provided to you. Fill in the header comments and add any additional comments you think are important.
- Use a List Loop with the Slice tool to create 3 elevation zone grids for 5, 10 and 15 zones.
- Hint 1: You'll need to check out the spatial analyst license to use Slice.

```
# check out spatial analyst license  
gp.CheckoutExtension("spatial")
```
- Hint 2: the items in the list loop will look like this [5,10,15]. However this presents a challenge when you create the output, because each of the output rasters should have a new name (e.g. slice5, slice10, slice15). The items in the loop are integers and you can't concatenate an integer to a string, so you'll have to convert the integers to strings. You'll need to import the string module, then list the functions in the module. From there you'll have to use the function that converts an object (e.g. an integer object) to a string object.