Python for ArcGIS

Day 1 - Focus on Python

Presented by

Matthew Collier

president of

Attribute Q, LLC

Agenda – Day 1

- Introduction
- Variables
- Iteration
- Conditionals
- User Defined Functions
- Python Modules
- Scripting with Python

Beginning Concepts 1/5

- Compiled Languages
  - C, C++, Fortran,…
  - Compiler translates into machine language executable which may be VERY fast.
- Interpreted Languages
  - Python, Perl, Ruby,…
  - Interpreter goes through slowly, line-by-line.
- Python
  - Complete Language
  - Used to glue other code together.
  - Why choose Python? (see link below)

resources.arcgis.com/en/help/main/10.1/index.html#//002z00000001000000

Beginning Concepts 2/5

- Syntax
  - How something is written (grammar)
  - Computer languages are persnickety!
  - Often easy for computer assisted error finding
  - Script often won’t even run so you know it’s there.
- Semantics
  - What you intend for the computer to do
  - Semantic errors often show up in the output
  - Logic and domain knowledge often needed to debug semantic errors

Beginning Concepts 3/5

- Accessing the Interpreter
  - From ArcGIS in Start Menu choose "Python 2.x | IDLE (Python GUI)"
  - Type commands after ">>> " prompt
  - Don't be afraid to make mistakes :-)

Beginning Concepts 4/5

- Character Encoding

```
>>> “This character string fails!”
>>> "This character string flies!"
```

- Continuation

```
>>> "This line, which is very long,
continues on this line, but fails"

>>> "This line, which is very long, flies \
just fine with the continuation character."
```

- Commenting

```
>>> # Everything after the first '#' is a comment.

>>> """Also, beginning and ending a text block with three
  quotes makes an interpretable comment across lines"""
```

Beginning Concepts 5/5

- Tips
  - Choose “Options | Configure IDLE” and adjust the color schemes and font for great visibility.
  - Use “File | Save As…” to remember command history as a *text* file.
  - Click on a line in the history above and hit enter. That will bring the line from history down to the command prompt. You can edit, and reissue the command.

BREAK

Python 101: Variables 1/8

- Assignment, Dynamic typing, and Display

```
>>> subGroup = 2        # an integer
>>> subgroup = 3        # Watch the C(c)ase!
>>> print subGroup      # note color coding

>>> anInteger = 2147483647
>>> type(anInteger)     # note color coding
>>> aLong = anInteger + 1
>>> type(aLong)
>>> aLong
```

Python 101: Variables 2/8

- Assignment, Dynamic typing, and Display

```
>>> approxiPi = 3.14        # a float
>>> approxiPi
>>> print approxiPi

>>> fcName = "roads"        # a string
>>> fcName
>>> print fcName
```

Python 101: Variables 3/8

- Casting with built-in functions

```
>>> int(approxiPi)
>>> int(fcName)          # generates error
>>> fpm = "5280"
>>> int(fpm)             # no error.
```

- Experiment with int( ), long( ), float( ), and str( ). What works? Why?

Python 101: Variables 4/8

- The variables we've defined are objects
  - Placing a '.' after the variable name, and hitting <tab> should reveal any members of that object
  - Use up/down arrows to highlight a member.
  - Or, start typing and the menu will respond.
  - Begin with a left parenthesis after function name
  - Place arguments, close parentheses, and hit enter to execute.
- Class Example: fpm.center( )

Python 101: Variables 5/8

- Lists - A true Python workhorse!
  - Indexed, with zero-based counting
  - Mutable

```
>>> fcList = ["lakes", "rivers", "trails"]
>>> print fcList[2]
>>> fcList[0] = "roads"

>>> nums = [[1,2,3],[4,5,6]]
>>> nums[1]
>>> nums[1][2]
```

Python 101: Variables 6/8

- Tuples
  - Indexed, with zero-based counting
  - Immutable
  - Strings are special cases of a tuple

```
>>> code = ("A", "AA", "AAA")
>>> code[0] = "B"   # produces error
>>> mix = (1, "C", 1.67) # also with lists
```

Python 101: Variables 7/8

- Dictionaries
  - Key:Value pairs to enable lookup
  - No index (you provide the key)
  - CAUTION: No ordering should be assumed!
  - Mutable values

```
>>> subTypeD = {6:"Okay",7:"Bad",8:"Ugly"}
>>> print subTypeD[0]    # produces error
>>> print subTypeD[7]
>>> subTypeD[6] = "Good"
```

Python 101: Exercises 8/8

- Experiment with sort( ), reverse( ), and other interesting members of the fcList object. Are these reversible operations?
- Try indices with a string. (i.e. fcName[3])
- How do the methods of tuples compare with those of lists?
- How do methods of dictionaries compare with those of lists?

BREAK

String Reprise: Slicing 1/6

- Try these examples with me:

```
>>> fcName = "roads.shp"
>>> fcName[1:6]  # count between chars
>>> fcName[1:]
>>> fcName[:6]
>>> fcName[:]
>>> fcName[:-4]  # Why use negatives?
>>> fcName[-3:]
```

String Reprise: Concatenation 2/6

- Adding and multiplying strings

```
>>> old = "lakes.shp"
>>> new1 = old[:-4].upper() + old[-4:]
>>> new2 = old[:-4] + ".csv"

>>> old[:-3]*3
>>> print old
>>> print '-'*len(old)
```

String Reprise: Formatting 3/6

- Passing integers into strings

```
>>> k = 5
>>> message = "Feature count: %i" % k
>>> print message

>>> print "k=%i" % k
>>> print "k=%4i" % k
>>> print "k=%04i" % k

>>> print "k=%i, k+2=%i" % (k, k+2)
```

String Reprise: Formatting 4/6

- Passing floating points into strings:

```
>>> z = -1.618
>>> "z = %.2f meters" % z
>>> print "z = %7.2f meters" % z
```

- Passing strings into strings:

```
>>> s = "Agricultural"
>>> "Landuse is %s." % (s,)
>>> print "Landuse is %20s." % (s,)
>>> print "Landuse is %-20s." % (s,)
```

String Reprise: Exercises 6/6

- Can you divide or subtract strings?
- Try printing the value of k with the formatting conversion specification %+04i. Now set k = -5, and print again.
- Does adding the + work the same for floating point number conversion?
- Do a triple conversion into a message string that uses the variables k, z, and s.

BREAK

Python 102: Control Structures 1/6

- Iteration with the for loop
  - Works with iterables like lists and strings
  - Consistently indent commands within code blocks
  - 3 spaces != 4 spaces != a single tab
  - Let's dissect a loop that uses range( )

```
>>> help(range)   # learn how range works

>>> for k in range(5):
...     square = k * k
...     print square
```

Python 102: Control Structures 2/6

- Iteration with the for loop
  - Another way: Implicitly pick up the elements

```
>>> count = 0
>>> fcOutList = []
>>> for item in fcList:
...     count = count + 1
...     fcOutList.append(item + "_out")

>>> print "Files converted: %i" % (count, )
>>> print "Names are:", fcOutList
```

Python 102: Control Structures 3/6

- if, elif, else conditionals
  - Comparisons made with: >, >=, <, <=, !=, <>, ==

```
>>> if  k < 10:
...       k = k + 1

>>> if  k < 10:  k = k + 1

>>> if k == 10:
...      k = k - 1
... else:
...      k = k + 1
```

Python 102: Control Structures 4/6

- if, elif, else conditionals

```
>>> if k <= 4:
...     print "Too low."
... elif k == 5:
...     print "Bingo!"
... elif (k >= 6) and (k <=10):
...     print "A little too large."
... elif k > 10:
...     print "Way too big!"
... else:
...     print "Oops:  I lost count."
```

Python 102: Control Structures 5/6

- Iteration with the while loop
  - works by evaluating a condition
  - for or while? Often it's just programmer's choice.

```
>>> big = bool(1)   # set big as True
>>> count = 10      # initialize counter
>>> while big:      # loop while True
...     print count
...     count = count - 1# decrement count
...     if count < 5:
...         big = bool(0) # set big to False
...         print "Countdown aborted!"
```

Python 102: Exercises 6/6

- Is it possible to implicitly for loop across a dictionary and print out the values?
- Think of a way to use an if-elif-else construction to make choices based on values from an attribute table.
- Try the following, describe what happens, and how to recover.

```
>>>  while True: pass
```

BREAK

Python 102: User Functions 1/4

- The user-defined function
  - Your first step beyond code snippets! Yay!
  - Useful for cleaning up complex or repeating operations inside your scripts.
  - But first, it's getting unweildy to do multi-line structures on the interpreter commandline
- Introducing: The IDLE editor window
  - Go to "File | New Window" menu item in IDLE
  - Then, go to "File | Save As..." and save it to your working directory as "functions.py"

Python 102: User Functions   2/4

- Type the following in the editor window
  - Watch for correct indentation
  - Note the colorization in your editor

```
def f2c(f): # Fahrenheit to Celsius
   c = 5.0*( f-32.0 )/9.0
   return c

celcius = f2c(212)
print celcius
```

Python 102: User Functions 3/4

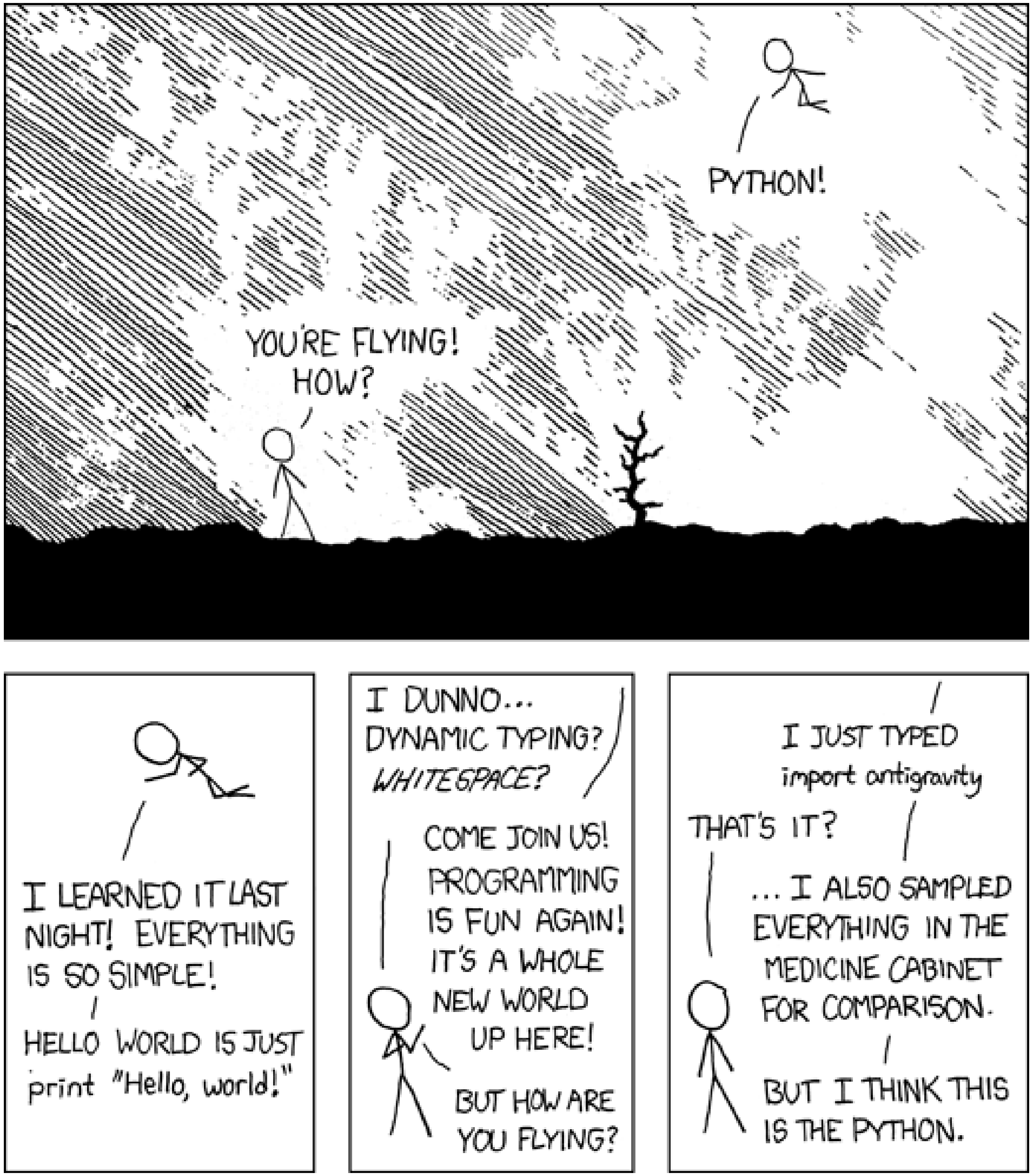
- Now do the following
  - In the editor choose "Run | Run Module"
  - Save the script when the option appears
  - Look in the Python Shell for any results
  - The function is now in your Python interpreter session's memory.
  - Try calling it from the Shell commandline as follows and insert a few different values in for the Fahrenheit argument:

```
>>> f2c(32)
```

Python 102: User Functions 4/4

- Examples, Examples, Examples...
  - We've almost arrived at some productivity!
  - But you need more experience...
  - LOTS of examples...
  - We'll work examples in the editor window.
  - Follow along interactively! Slow me down if I go too fast!
  - After we get each function working, save the entire editor window out as functions.py so you can have a copy for your records.

BREAK


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

From http://xkcd.com/353/

Python 201: math Module 2/7

- It's in the standard library
- See the following for details:
  - http://docs.python.org/release/2.7.2/library/math.html
- Contains members for: angular conversion, trigonometry, powers, logarithms, constants, and others...
- Provides acces to C standard math functions

Python 201: math Module 3/7

- examples using the module

```
>>> sqrt(2)                  # produces error
>>> import math              # basic import
>>> math.sqrt(2)

>>> import math as m         # variation 1
>>> m.sqrt(2)

>>> from math import *       # variation 2
>>> sqrt(2)                  # now it works
```

Python 201: os Module 4/7

- It's in the standard library
- See the following for details:
  - http://docs.python.org/release/2.7.2/library/os.html
- Wraps OS interaction into a standard Python form regardless of platform.
- Some useful methods include: getcwd( ), chdir( ), listdir( ), mkdir( ), rename( )

Python 201: sys Module 5/7

- Also in the standard library
- See the following for details:
  - http://docs.python.org/release/2.7.2/library/sys.html
- Variables kept or used by Python interpreter
- Also contains functions interacting strongly with Python interpreter.
- Some useful methods include: sys.path, sys.path.append( ), with many others.

Python 201: Modules 6/7

- The Standard Modules
  - math, os, sys are only a few of many!
  - http://docs.python.org/library/index.html

- There are many other third party modules!
  - Search on “python”, “module” and other keywords
  - Review http://pypi.python.org/pypi/

Python 201: Modules 7/7

- You've already made your own module called “functions”!
- How does Python know where to find it?
- How would you import it?
- Making an executable module...

```
# ...normally indented functions above

if __name__ == "__main__":

    # indented code follows...
```

BREAK

Python 201: Scripting 1/1

- We'll end the day with scripting practice by doing one or both of the following exercises as a class:
  - Code read-throughs. These are an excellent way to learn new techniques, and find solutions!
  - Building a driver. Illustrates building more complex applications from code encapsulated in functions and modules.

BIG BREAK

(See you tomorrow)

Resources to keep you up tonight

- The standard Python documentation
  - http://www.python.org/doc/
  - http://docs.python.org/release/2.7.6/
  - You can play with that last number in the URL above to find your specific version of Python.