GEOG 432/832: Programming, Scripting, and Automation for GIS

Week 01.02: Intro to Python

Dr. Bitterman

Today's schedule

- Open discussion
- Python basics and practice

Anything to discuss?

Programming in Python

- What have you used before?
- We're going to talk about (and use) 3 main interpreters:
  - IDLE
  - Spyder
  - Jupyter notebooks (both in and out of ArcPro)
- First, open ArcGIS Pro
  - You'll have to setup the license server the first time you login to a machine
  - Find instructions on the Canvas page
- Find IDLE in your start menu, open it (we'll talk about it)
- Next, launch Anaconda --> Jupyter notebooks (we'll talk about it too)
- Finally, launch Spyder (in Anaconda) (we'll talk some more)

What is a variable?

Time to practice. We'll start by looking at variables.

- From algebra: a letter can represent any number, like in `x + 2`
- In computer science, variables represent *values* or *objects* we want the computer to store in its memory for later use
- Variables don't only represent numbers
    - ...but also text and Boolean values ('true' or 'false')
    - input from the program's user
    - to store values returned from another program
    - to represent constants
    - etc.

Why use variables?

- Variables make your code readable and flexible
- Hard-coded values (also called "magic numbers"), mean your code is useful only in one particular sceanario
- You can manually change the values in your code to fit a different scenario
    - tedious
    - greater risk of making a mistake
- Variables allow your code to be useful in many scenarios and are easy to *parameterize*, meaning you can let users change the values

Let's write some (basic) code

- **Open** Jupyter Notebook in Anaconda
- Create (or find) a working directory
- Create a new notebook, name it something sensible

In the first cell, type the following, then press *Enter:*

```
x = 5
```

What happened? ...anything?

...let's try **Shift + Enter**

What happened?

You've declared a variable *x* and set its value to 5
In some programming languages (e.g., Java) you're required to tell the interpreter what the *type* of the variable is. Not Python.

When you execute a command, nothing *appears* to happen, but the program now has stored this variable in memory. To prove this, type:

```
x + 3
```

What happened?

You can also print. Try:

```
print(x + 3)
```

We'll use the print function a lot when we're testing code

Some strings

Variables can also represent words, or *strings*. Try this:

```
ourTeam = "Huskers" #or "Hawkeyes", or whatever your team is :)
print(ourTeam)
```

- Here, the quotation marks tell Python that you are declaring a string variable
- Python is a powerful language for working with strings
- A simple example of string manipulation is to add, or concatenate, two strings:

```
string1 = "Go "
string2 = "Big "
string3 = "Red!"
print(string1 + string2 + string3)
```

TRY IT!

Did you get the spacing right?

More strings

- You **can** include a number in a string variable by putting it in quotes...
- but you must thereafter treat it like a *string*, NOT a number
- For example, this results in an error:

```
myValue = "5"
print (myValue + 3)
```

Try it, what happens?

Tips for naming variables

- Variable names are case sensitive. myVariable is a different variable than MyVariable.
- Variable names cannot contain spaces
- Variable names cannot begin with a number
- Recommended practice for Python variables
    - name the variable beginning with a lower-case letter
    - then begin each subsequent word with a capital letter
    - called "camel case"
    - For example: myVariable, mySecondVariable, roadsTable, bufferField1, etc.
- Variables cannot be reserved words such as "import" or "print"
- *Make variable names meaningful so that others can easily read your code. This will also help you read your code and avoid making mistakes*

What can variables represent?

- number and string variables we worked with above represent data types that are built into Python
- Variables can also represent other things:
  - such as GIS datasets
  - tables
  - rows
  - geoprocessor that can run tools
- All of these things are objects that you use when you work with ArcGIS in Python

(Most) everything is an object

In Python, everything is an object. All objects have:

- a unique ID, or location in the computer's memory
- a set of properties that describe the object
- a set of methods, or things that the object can do

Using objects: Let's pretend

Task

We're writing a program to make a peanut butter and jelly sandwich. If we were to write the program in a procedural language, it would flow something like this:

1. Go to the refrigerator and get the jelly and bread
2. Go to the cupboard and get the peanut butter
3. Take out two slices of bread
4. Open the jars
5. Get a knife
6. Put some peanut butter on the knife
7. Spread the peanut butter on one piece of bread
8. etc.

In an object-oriented (o-o) language:

Something like this:

```
mySandwich = Sandwich.Make
mySandwich.Bread = Wheat
mySandwich.Add(PeanutButter)
mySandwich.Add(Jelly)
```

The sandwich object "knows how" to build itself, given just a few pieces of information.

You can define the properties of the sandwich (like the bread type) and perform *methods* (or actions) on the sandwich, such as adding the peanut butter and jelly

Classes

Why so easy?

- Someone did the work!
  - Defined what a sandwich is
  - What you can do with it

They did this using a *class*

A *class* defines:

- how to create an object
- properties available to that object
- methods available to that object
- how the properties are set and used
- what each method does

A class is a blueprint for creating objects

The blueprint determines what properties and methods an object of that class will have

A common analogy is that of a car factory

- Produces thousands of cars of the same model, built using the same blueprint
- A class produces objects that have the same predefined properties and methods

Classes in Python

- Classes are grouped together into *modules*
- You import modules into your code to tell your program what objects you'll be working with
- You can write modules yourself, but most likely you'll bring them in from other parties or software packages. For example, the first line of most scripts you write in this course will be:

```
import arcpy
```

- import keyword to tells your script that you'll be working with the arcpy module, which is provided as part of ArcGIS
- After importing this module, you can create objects that leverage ArcGIS in your scripts

Other modules

- **os**: allows you to work with the operating system
- **random**: allows for generation of random numbers
- **csv**: allows for reading and writing of comma-separated value files (CSVs)
- **math**: allows you to work with advanced math operations

Above are included with Python, but they aren't imported by default.

- Only import what you need for that particular script
- For example, although it might not cause any errors in your script, you wouldn't include import arcpy in a script not requiring any ArcGIS functions.

Python syntax

Every programming language has rules

- capitalization
- white space
- how to set apart lines of code and procedures, and so on

Basic syntactical rules to remember

- Python is *case-sensitive* both in variable names and reserved words
    - Important whether you use upper or lower-case
    - All lower-case "print" is a reserved word in Python that will print a value, while "Print" is unrecognized and will return an error
    - arcpy is *very* sensitive about case and will return an error if you try to run a tool without capitalizing the tool name
- You end a Python statement by pressing **Enter** and literally beginning a new line. It's okay to add empty lines to divide your code into logical sections
- If you have a long statement that you want to display on multiple lines for readability, you need to use a line continuation character, which in Python is a backslash `\`

Basic rules continued

- Indentation is required to logically group together certain lines, or blocks, of code
- Indent your code four spaces inside loops, if/then statements, and try/except statements
- In most programming languages developers are encouraged to use indentation to logically group together blocks of code; however, in Python, indentation of these language constructs is not only encouraged, but **required**
- Seems like a pain, but results in greater readability

More rules!

Add a comment to your code by beginning the line with a `#` character

- Comments are lines that you include to explain what the code is doing
- Comments are ignored by Python when it runs the script, so use them at any place in your code
- Comments help others
- Comments help you! (and especially *FUTURE* you!)
- The intended audience for comments is developers

Adding documentation

- Add documentation to your code by surrounding a block of text with `"""` (3 consecutive double quotes) on either side
- Text surrounded by `"""` is ignored when Python runs the script.
- Unlike comments, which are sprinkled throughout scripts, documentation strings are generally found at the beginning of scripts. Their intended audience is end users of the script (non-developers)

A first example

```
# Opens a feature class from a geodatabase and prints the spatial reference

import arcpy

featureClass = "C:/Data/USA/USA.gdb/Boundaries"

# Describe the feature class and get its spatial reference
desc = arcpy.Describe(featureClass)
spatialRef = desc.spatialReference

# Print the spatial reference name
print (spatialRef.Name)
```

What are some concepts that we've talked about are demonstrated in this code block?

For next class

- Chapters 1 & 2 are relevant (if you haven't read already)
- Practice on your own. Install Anaconda or another *free* Python interpreter
  - Pycharm is good too
- This week, Friday is NOT a workday - sort of. We'll be doing more exercises in class