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

Week 01.02: Programming Exercise

Dr. Bitterman

Thursdays are typically our work days

The purpose of today's work

- Practice, practice, practice
- Don't worry about completing all the tasks in class
- These are here to stimulate your brain
- ...and to prime you for the rest of the semester

there will be hiccups along the way, don't worry (too much) about them

Anything to discuss?

(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! They 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 the first half of 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 (often non-developers)

Comments and octothorpes

99 PI: https://99percentinvisible.org/episode/octothorpe/

Comments are very important in your programs. They are used to tell you what something does in English, and they also are used to disable parts of your program if you need to remove them temporarily. Here's how you use comments in Python:

```
# A comment, this is so you can read your program later.
# Anything after the # is ignored by python.

print("I could have code like this.") # and the comment after is ignored

# You can also use a comment to "disable" or comment out a piece of code:
# print "This won't run."

print("This will run.")
```

Comment-out a print command and run again - what happens differently?

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 of the concepts we've talked about that are demonstrated in this code block?

More practice

Python the hard way

- A sort of "classic" resource for learning Python *by doing*, even if you don't know **what** you're doing
- But it's in Python 2.7, while we're using Python 3
- It's unfortunately been monetized: https://learncodethehardway.org/python/
- But some resources (like I'm co-opting here) are available on the web: http://cglab.ca/~morin/teaching/1405/lpthw/book/ex1.html

Numbers and math

Every programming language has some kind of way of doing numbers and math.

- `+` plus
- `-` minus
- `/` slash
- `*` asterisk
- `%` percent (modulo)
- `<` less-than
- `>` greater-than
- `<=` less-than-or-equal
- `>=` greater-than-or-equal

Paired programming exercises

1. Find a buddy

2. Open a Jupyter notebook (wherever)

Numbers and math part 2

Write one at a time and STOP TO THINK/CALCULATE what the output will be prior to running each command:

```
print("I will now count my chickens:")

print("Hens", 25 + 30 / 6)
print("Roosters", 100 - 25 * 3 % 4)

print("Now I will count the eggs:")
print(3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6)

print("Is it true that 3 + 2 < 5 - 7?")
print(3 + 2 < 5 - 7)

print("What is 3 + 2?", 3 + 2)
print("What is 5 - 7?", 5 - 7)

print("Oh, that's why it's False.")

print("How about some more.")

print("Is it greater?", 5 > -2)
print("Is it greater or equal?", 5 >= -2)
print("Is it less or equal?", 5 <= -2)
```

Variables and characters

1. Write a comment above each line explaining to yourself what it does in English
2. Explain to your neighbor what the commands will do

```
cars = 100
space_in_a_car = 4.0
drivers = 30
passengers = 90
cars_not_driven = cars - drivers
cars_driven = drivers
carpool_capacity = cars_driven * space_in_a_car
average_passengers_per_car = passengers / cars_driven

print("There are", cars, "cars available.")
print("There are only", drivers, "drivers available.")
print("There will be", cars_not_driven, "empty cars today.")
print("We can transport", carpool_capacity, "people today.")
print("We have", passengers, "to carpool today.")
print("We need to put about", average_passengers_per_car, "in each car.")
```

Now do it yourself (but let's start together)

Say hello

Create a string variable called `x` and assign it the value "Hello". Display the contents of the x variable in the Console.

Concatenate two strings

Create a string variable called `first` and assign to it your first name. Likewise, create a string variable called `last` and assign to it your last name. Concatenate (merge) the two strings together, making sure to also include a space between them.

Do some easy math and report results

Create a variable `score1` and assign it a value of 60. Then create a variable `score2` and assign it a value of 40. Compute the sum and average of these values and output to the Console the following messages, filling in the blanks:
The sum of these scores is _____.
Their average is _____.

tips:

In constructing your messages, you're probably going to want concatenate text strings with numbers. This requires converting the numeric values into strings to avoid a syntax error. The str() function can be used to do this conversion. **For example:**

```
print('Score 1 is ' + str(score1))
```

For next week

Topic: Geoprocessing in ArcPro and Python fundamentals

- Read: chapters 3 & 4 in textbook
- Lab 1 starts next week
- Practice on your own. Install Anaconda or another *free* Python interpreter and see what you can do!
    - Pycharm and DataSpell are good too (and what I tend to use)