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

Week 03.01: Geoprocessing in Python (with ArcGIS Pro too)

Dr. Bitterman

Today's schedule

- Open discussion
- Lab check-in
- Discussion and exercises
- For next class

Open discussion

How is lab 1 going? Any challenges? Thoughts? Questions?

Unit schedule

- Today: Discussion and exercises
- Next class: Continuation and project introduction

Next week

- Next Monday: More, more, more (Python geoprocessing)
- Next Wednesday: Discussion of assigned reading (see Canvas) + leftovers
- Next Friday: Lab 02 starts

Geoprocessing in Python (+ arcpy)

ArcPy

- The ArcPy package exposes ESRI ArcGIS Pro functions to Python env.
- Includes:
  - modules
  - classes
  - functions
- So we can use all the geoprocessing tools in ArcGIS Pro in a Python script

ArcPy and other editors

- It **is** possible to use the ArcGIS Pro python environment in other editors (e.g., Spyder, Python)
    - *(See your book and ESRI documentation for examples and tutorials)*
- It's a bit trickier with Jupyter/Anaconda - especially given the permissions restrictions in a university lab... :(
- For this work, we're going to use the built-in Jupyter notebook functionality in ArcGIS Pro

Download the in-class data

- Start a new ArcGIS Pro project
- Download `week03inclass.zip` from GitHub repository
- Extract it to the same folder as your project

Importing ArcPy

- Working with ArcPy starts with importing the package
- A typical geoprocessing script therefore starts with the following line of code:

```
import arcpy
```

- Gives access to toolboxes installed with ArcGIS Pro

Not necessary if using Python from ArcGIS Pro!

ArcPy structure

Several modules

- module for working with data `(arcpy.da)`
- map scripting module `(arcpy.mp)`
- module for image analysis and interpretation `(arcpy.ia)`
- module for map algebra and raster analysis `(arcpy.sa)`

Once imported, you can start use modules, functions, and classes

During the import...

Two important checks:

1. **is ArcPy available?** requires:

    i. that ArcPy is installed

    ii. that the Python editor used to run the script is configured to use ArcPy.

    The default installation of other IDEs requires additional steps. If ArcPy is not available, the import will generate the following error message: `ImportError: No module named arcpy`

2. **is an ArcPy license is available?** ArcPy is *closed source*. If ArcPy is installed, but no license is available, the line of code import arcpy will generate the following error message: `RuntimeError: NotInitialized`

Setup reminder

- When writing code in a Python notebook or window in ArcGIS Pro, there is no need to import arcpy or check for licensing - you're already setup
- Any Python script that runs outside ArcGIS Pro—for example, using IDLE or PyCharm, must use `import arcpy`

Data storage

Discussion question:

How do we store spatial data? What methods, structures, or systems are you familiar with?

Geodatabases

Esri has developed multiple ways of storing spatial data

- You are encouraged to put your data in **geodatabases:**
    - organizational structures for storing datasets and defining relationships between those datasets.
- Different "flavors" of geodatabase are offered for storing different magnitudes of data

1. **File geodatabases**:
    i. store data locally
    ii. using an ESRI-proprietary format
    iii. more functionality than shapefiles

ArcPro has more...

2. **ArcSDE geodatabases** or "enterprise geodatabases":
    i. store data on a central server in a relational database management system (RDBMS) such as SQL Server, Oracle, or PostgreSQL. 2. ArcSDE is "middleware" that allows you to configure and read your datasets without touching the RDBMS software

- can also pull data directly out of an RDBMS using SQL queries, with no ArcSDE involved, through **query layers**
- A single vector dataset within a geodatabase is called a **feature class**
    - Feature classes can be optionally organized in **feature datasets**
    - Raster datasets can also be stored in geodatabases

Shapefiles

Discussion: what's a shapefile?

A shapefile is...

- A "standalone" **vector** data format
- Actually consists of several files that work together to store vector geometries and attributes
- The files all have the same root name, but use different extensions

In the Esri file browsers in ArcGIS Pro (and ArcMap, and QGIS), the shapefiles appear as a single file

GDBs vs. shapefiles

- When should we use a GDB or a shapefile?
- Largely personal preference, but...
  - GDB is a useful "container"
  - provides additional structural advantages over shapefiles
  - custom point, line and polygon features
  - Networks, subtypes, etc.
- Downsides:
  - Is ESRI-specific... locks you in (there is an API, but is limited)

ArcGIS Pro by DEFAULT creates a .gdb for each project

Paths

- Often in a script, you'll need to provide the path to a dataset
- Syntax for specifying the path is sometimes tricky because of the many different ways of storing data listed above. **Why?**
    - Navigate to a Geodatabase on your computer. What do you see?
    - What about a shapefile?

So what to do? **(until you get the hang of it - and sometimes even after)**

1. Check ArcGISPro's Catalog View and browse to the dataset
2. The location box along the top indicates the directory or geodatabase whose contents are being viewed. Clicking the breadcrumbs or dropdown arrow displays the full path

- Alternatively, you could right-click any feature class from either the Catalog View or Catalog Pane, go to Properties, then click the Source tab to access its path

Path syntax reminder

- You can't use a single backward slash `(\)` for paths. Why?
- because Python views it as an escape character
- Other correct notations are `r"C:\Data"` and `"C:\\Data"`
- A path is a *string variable*

Workspaces

- The Esri geoprocessing framework uses the notion of a **workspace** to denote the folder or geodatabase where you're working
- When you specify a workspace, you don't have to list the full path to every dataset
- When you run a tool, the geoprocessor sees the feature class name and *assumes that it resides in the workspace you specified*

Other utility

- Workspaces are especially useful for batch processing, when you perform the same action on many datasets in the workspace
- For example, you may want to clip all the feature classes in a folder to the boundary of your county

Setting a workspace

A workspace provides a default location for the files you will be working with, such as inputs and outputs of geoprocessing tools.

For example, here is how to set the current workspace to `C:\Data`:

```
import arcpy
arcpy.env.workspace = "C:/Data"
# or arcpy.env.workspace = "C:\\Data"
```

Complicating factors

- A Python script also has a current working directory, which by default is the location of the script
- You can get the current working directory using `os.getcwd()`, and change it using `os.chdir()`

However...!!!

However, these Python-only options are **not enough** to work with geospatial datasets

- not all valid workspaces in ArcGIS Pro are recognized by the Windows operating system
- so set the workspace in a geoprocessing script using `arcpy.env.workspace`

Let's actually do some work!

Start simple

setup and list some files

1. Start a new project in ArcGIS Pro
2. Put the "week 03 data" in that directory
   (your workspace will be different than mine)

BEFORE RUNNING THE CODE, WHAT IS GOING TO HAPPEN?

```
# set the workspace to the path where your data are
arcpy.env.workspace = "C:\\Users\\pjbitterman\\Dropbox\\GEOG432\\week03\\week03data"

myFiles = arcpy.ListFiles()

myFeatureClasses = arcpy.ListFeatureClasses()

myFiles

myFeatureClasses
```

Feature class properties

What's your expectations?

```
# find one spatial ref

myFc = "Municipal_Boundaries.shp"
desc = arcpy.Describe(myFc)
spatialRef = desc.SpatialReference
print(spatialRef.Name)
```

What *did* happen?

Let's batch it a bit

What will this code do?

```
# find spatial ref and VCS

for fc in myFeatureClasses:
    desc = arcpy.Describe(fc)
    spatialRef = desc.SpatialReference
    print(fc, spatialRef.Name, spatialRef.VCS)
```

What did it print? How do you know? *(and if not, how would you find out?)*

Accessing fields (refresher)

- Vector features in ArcGIS feature classes are stored in a table with records (rows) and attributes (columns)
- Fields in the table store the geometry **and** attribute information for the features
- There are two fields in the table that you cannot delete
    i. The geometry of the feature (usually called Shape), stored in binary format
    ii. object ID field (OBJECTID or FID). This contains a unique number, or identifier for each record that is used by ArcGIS to keep track of features
- The rest of the fields contain attribute information that describe the feature. These attributes are usually stored as numbers or text

Accessing fields (the code)

We always break down the code before we run it...

```
# fields

parksFc = "State_Park_Locations.shp"
fieldList = arcpy.ListFields(parksFc)

# Loop through each field in the list and print the name
for field in fieldList:
    print (field.name)
```

And we always compare what happened to our expectations...

More about fields

What to note?:

- OBJECTID, which holds the unique identifying number for each record, and Shape, which holds the geometry for the record
- Other fields?

arcpy treats the field as an object

- field has properties that describe it (like `field.name`)
- The ESRI help reference topic: Using fields and indexes lists all the properties that you can read from a field
- Field properties are read-only
- To change field properties (e.g., scale, precision), add a new field

Let's try a function

What does the Clip function do?

Let's break down this code:

```
arcpy.Clip_analysis("State_Park_Locations.shp", "lancaster_county.shp", "myFirstOutput.shp")
```

Run it - what happens?

A function in a loop

Break it down first... expectations?

```
## clip with loop

import arcpy # unnecessary if already done above

featureClassList = arcpy.ListFeatureClasses()
clipFeature = "lancaster_county.shp"

if not os.path.exists("tempoutput"):
        os.makedirs("tempoutput")

for featureClass in featureClassList:
    arcpy.Clip_analysis(featureClass, clipFeature, "\\tempoutput\\" + "lc_clip_" + featureClass)
```

What happened? What issues did it create? And how might you fix them?

For next class

- Readings (Chapter 5 if you haven't already)
- Practice!
- Lab 01 is due on 2/11