Click here to watch the Lesson 3A, Part 1 (of 2) video

Listing and iterating spatial data

The arcpy List methods

Running tools in a loop

Review of Describe

Putting it together

PSA Book Sections

6.4 – 6.7: Listing

6.3: Describe

NR426

Programming for GIS I

Lesson 3 (a, b)

Instructor:

Elizabeth Tulanowski | ESS Department

What does Lesson 3 cover?

Two class sessions: Lessons 3a, 3b

Using the various arcpy List methods

Describing data to control flow within a loop

Implementing larger workflows with arcpy

Batch-run Toolbox tools through Python

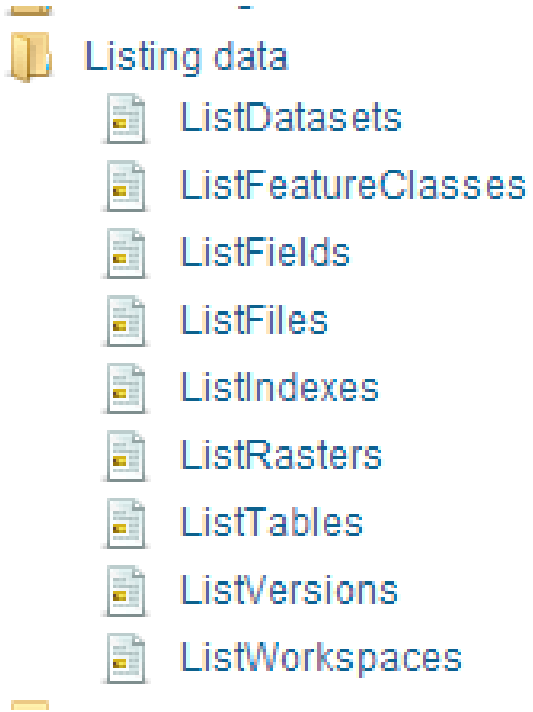
Lesson 3a

The arcpy List methods

Running tools in a loop

Listing geographic data

- Several list methods are available to loop through feature classes, raster, folders, fields, and more.
- Each list method returns a Python list, which can be looped through with a `for` loop
- List methods available for Data Tools/Toolboxes
- The workspace MUST be set – the list will be created for the data in the workspace



PSA Book Ch. 6.4 – 6.7, p. 198-217

List examples

ListFeatureClasses help

```
fcList = arcpy.ListFeatureClasses()
```
→ returns all FCs in workspace

```
fcList = ListFeatureClasses("*", "all")
```
→ returns all FCs in workspace

Wildcards can be used to limit which datasets go into list:

```
fcList = ListFeatureClasses("*", "polygon")
```
→ Only return polygon FCs

```
RasterList = arcpy.ListRasters("C*")
```
→ All rasters that start with C

Fields need more: returns list of *field objects*, must retrieve property of each field

```
FldList = arcpy.ListFields(layer)
```
→ FldList contains field objects (not friendly)

```
for fld in FldList:
    print (fld.name)
```
→ Loop through list and print a given field property

List methods

Function	Description
ListFields(dataset, wild_card, field_type)	Returns a list of fields found in the input table
ListIndexes(dataset, wild_card)	Returns a list of attribute indexes found in the input value
ListDatasets(wild_card, feature_type)	Returns the datasets in the current workspace (Datasets = Feature datasets, raster datasets, Network datasets, etc… Does not include feature classes or GDBs)
ListFeatureClasses(wild_card, feature_type, feature_dataset)	Returns the feature classes in the current workspace (Vector only, GDB FCs, shapefiles only. Does not include KML/KMZ, GeoJSON, or geopackage)
ListFiles(wild_card)	Returns the files in the current workspace. (Includes files and folders, does not dig into subfolders)
ListRasters(wild_card, raster_type)	Returns a list of rasters found in the current workspace
ListTables(wild_card, table_type)	Returns a list of standalone tables found in the current workspace (Does not include attribute tables of vector or raster layers)
ListWorkspaces(wild_card, workspace_type)	Returns a list of workspaces found in the current workspace (Workspace = location in which data are stored: Folders or Geodatabases)
ListVersions(sde_workspace)	Returns a list of versions the connected user has permission to use (For Enterprise GDB only)

* **Bold** indicates most common list methods

From Create Lists of Data help page

A raster example

"List raster sample_TNC….py"

What is this code making a list of?

Rasters? Yes.

Be more specific…

```
import arcpy, os
from arcpy.sa import *

maindir = "L:\\Projects_active\\TNC_Riparian\\Gunnison\\2015\\"
smsdir = maindir + "SegmentMeanShift\\"
isodir = maindir + "ISO\\"
ecddir = isodir + "ECDfiles\\"
print(ecddir)
arcpy.env.workspace = maindir
# create segmented image for each tile
for tif in arcpy.ListRasters("*", "TIF"):
    inRast = tif
    spectral = "20"
    spatial = "20"
    min_seg_size = "1"
    bands = "4 1 2"
    if not os.path.exists(smsdir + "seg_" + str(tif)):
        seg_raster = SegmentMeanShift(inRast, spectral, spatial, min_seg_size, bands
        seg_raster.save(smsdir + "seg_" + str(tif))
    else:
        print("SMS complete")
arcpy.env.workspace = smsdir
for tif in arcpy.ListRasters("*", "TIF"):
    print(tif)
    inSegRast = tif
    maxClasses = "15"
    out_def = ecddir + str(tif) + "_iso.ecd"
    maxIter = "20"
    minNumSamples = "10"
    skipFactor = "2"
    attributes = "COLOR;MEAN"
    if not os.path.exists(out_def):
        TrainIsoClusterClassifier(inSegRast, maxClasses, out_def, "", maxIter, minNu
    else:
        print("ecd complete")
```

Demo

Watch this demo as a video (Lesson 3A, Part 2 of 2)

- Review lists
- Review running tools
- Finding help on the List methods
- Put it together
  - Create simple script
  - Modify to run on a list
  - Modify to automate from a workspace
  - Use starter code on the N:\ drive: ListDemoStarterCode.py

```
# Import any necessary modules
import arcpy, sys

# Set any variables
# Set the workspace to the FtCollins.gdb
arcpy.env.workspace = r"C:\path\to\FtCollins.gdb"
# Set a variable to a feature class to work with
fc = "city“

#  Does the dataset exist?
if arcpy.Exists(fc):
    print(f"Yes, the {fc} dataset is there")
else:
    print("That dataset does not exist, need to quit script")
    sys.exit()

# Run a tool on one dataset

# Generic syntax for running any tool is: arcpy.Toolname_alias(arguments)
# Buffer the city boundary
arcpy.Buffer_analysis(fc, "city_buf", "1000 feet")

# Print out a list of everything in the geodatabase
fclist = arcpy.ListFeatureClasses()
for x in fclist:
    print(x)   # x is a variable whose value changes with every iteration

# Create (and print out) a list of only point feature classes
ptlist = arcpy.ListFeatureClasses("*", "Point")
for y in ptlist:
    print(y)

# Print out a statement with the number of point FCs in the GDB
numpts = len(ptlist) # len gets the number of items in a list
print(f"There are {numpts} point feature classes in the GDB")
```

PSA Book Ch. 6, end of 6.5

Getting into subfolders

- Use arcpy.da.Walk (Link to help page)
  - `Walk (workspace, {topdown}, {onerror}, {followlinks}, {datatype}, {type})`
- Provide the top-level folder or workspace
- Returns a tuple containing workspace, folder names, file names
  - To view results, iterate over the elements of the tuple
- Similar to os.walk but arcpy.da.Walk recognizes spatial formats

```
# Set variables
toplevel = r"C:\Data\Lesson3LabData"
arcpy.env.workspace = toplevel
feature_classes = []
walk = arcpy.da.Walk(arcpy.env.workspace, datatype="FeatureClass", type= "point")
for dirpath, dirnames, filenames in walk:
    for filename in filenames:
        feature_classes.append(os.path.join(dirpath, filename))
print (feature_classes)
```

[C:\Data\Lesson3LabData\\CityOfSanAntonio.gdb\\Crimes',
C:\Data\Lesson3LabData\\CityOfSanAntonio.gdb\\Burglary',
C:\Data\Lesson3LabData\\UnionCity.gdb\\ParkCenters,
C:\Data\Lesson3LabData\\UnionCity.gdb\\Schools,
C:\Data\Lesson3LabData\\FtCollins.gdb\\Schools]

In-class activity: Complete the code and practice with the help

- List all the polygon feature classes in the workspace:

```
gdbList = arcpy.List_____(_____)
```

- List all the integer fields that start with "N":

```
fieldList = arcpy.List_____("Roads", _____)
```

- List all the rasters that are TIFs:

- List all the tools from the analysis toolbox:

```
toolList = arcpy.
```

Lab 3A – List methods

- Write the code to loop through a geodatabase
- For only the polygon layers:
  - Print the name and number of features
  - Create readable well commented code with variables and environments set
  - Create code that can be easily reused on different data

Click here to watch Lesson 3B, Part 1 as a video

Lesson 3B – All about Describe

Using Describe to check data properties

Accessing fields and spatial reference

Describing data with arcpy.Describe()

- Retrieve data properties
- Describe help doc
- All data have access to the Describe object properties
- More general Describing Data help doc

```
import arcpy, os

#Set the workspace
path = r"C:\Student\MtnCampus_map\MtnCampus_map.gdb"
arcpy.env.workspace = path
lyr = "MtnCampus_streams"

#Create a describe object
dsc = arcpy.Describe(lyr)

#Call on properties
print ("-Shape type / geometry type for the layer:")
print (dsc.shapetype)

print ("-Data type for the layer:")
print (dsc.datatype)

print ("-Dataset type for the layer:")
print (dsc.datasettype)

print ("-File path for the layer:")
print (dsc.catalogpath)
```

```
-Shape type / geometry type for the layer:
Polyline
-Data type for the layer:
FeatureLayer
-Dataset type for the layer:
FeatureClass
-File path for the layer:
C:\Student\MtnCampus_map\MtnCampus_map.gdb\MtnCampus_streams
```

PSA Book Ch. 6.3, p. 195

Describe examples

```
dsc = arcpy.Describe("roads.shp")
print (dsc.name)                          → roads.shp
print (dsc.spatialreference.name)   →
                                  NAD_1983_StatePlane_Colorado_Feet
print (dsc.datasettype)             → FeatureClass
print (dsc.datatype)                → ShapeFile
print (dsc.catalogpath)             → Path to roads.shp
print (dsc.basename)                → roads
print (dsc.file)                    → roads.shp
print (dsc.shapetype)               → Polygon
```

Using Describe

- Another way to describe datasets, all in one line instead of 2:

```
arcpy.Describe("Boulder.gdb").WorkspaceType
```

- Additional useful properties:
  - `dsc.WorkspaceType` → FileSystem, LocalDatabase, RemoteDatabase
  - `dsc.Fields` → Returns a list of fields for a table/feature class. To get the field name, use `print (field.name)`
  - For rasters, `dsc.Format` → GRID, TIF, JPG etc…

PSA Book Ch. 6.3

Use Describe to access fields

```
arcpy.env.workspace = r"D:\Lesson3LabData\UnionCity.gdb"

fcList = arcpy.ListFeatureClasses()
for fc in fcList:
    print (fc)

layer = "Watersheds"

print ("\nLet's use Describe to return all the field names")
dsc = arcpy.Describe(layer)
lyrFlds = dsc.Fields
for fld in lyrFlds:
    print (fld.name)

print ("\nNow let's use List Fields and only grab the fields that start with M")
lyrFlds2 = arcpy.ListFields(layer, "M*")
for f in lyrFlds2:
    print (f.name)
```

```
Let's use Describe to return all
the field names
OBJECTID
Shape
INDEX
WATERSHED
ABBREVIATI
TOTAL_AREA
ATLANTA
COLLEGE_PA
EAST_POINT
ALPHARETTA
FAIRBURN
MOUNTAIN_P
PALMETTO
ROSWELL
UNION_CITY
OUTSIDE_FU
FULTON_COU
N_OR_S
NEW_AREA
AREA
RECNO
MINX
MINY
MAXX
MAXY
TOTAL_WATE
TOTAL_WAT_
TOTAL_PHOT
SWM_DISTRI
Shape_Length
Shape_Area
```

```
Now let's use List Fields and only grab the fields that start with M
MOUNTAIN_P
MINX
MINY
MAXX
MAXY
```

PSA Book Ch. 6.3

Describing data with arcpy.Describe()

From Lesson 2B… for your reference

- Examples:

```
# Create a describe object from a feature class

dsc = arcpy.Describe (r"C:\Data\Trails.shp")
#Print some properties

print ("Feature type: " + dsc.featureType)
print ("Shape type: " + dsc.shapeType)
print ("Spatial Index: " + dsc.hasSpatialIndex)
print ("Format: " + dsc.format)  #would throw an
                                 error for a vector layer
```

Use `if hasattr()` to only call a property if it is available

```
# Create describe object from another feature class

dsc = arcpy.Describe (r"C:\Data\WUI_area.shp")
#Print some properties

if hasattr (dsc, "name"):
    print ("Feature type: " + dsc.name)
if hasattr (dsc, "dataType"):
    print ("Shape type: " + dsc.dataType)
if hasattr (dsc, "format"):
    print ("Format: " + dsc.format)
```

Output to text file

From Lesson 2B… for your reference

```
import arcpy
## ********************************* ##
# Set some variables #  C H A N G E   T H E S E :
txtfile = r"C:\Documents\NR426-427\describing.txt"       # Full path to txt file you want to create
arcpy.env.workspace = r"C:\Data\GIS_Data"  # Full path to workspace
data = "NorthwestRoadTrip.gdb"                # Name only of data to describe
## ********************************* ##
# Create and open output text file
outFile = open(txtfile, "w")

# Describe the data
dsc = arcpy.Describe(data)

#Write information out to the text file
outFile.write("***** Describe information for "+data+" *****\n\n")
try:
    outFile.write(data+ " has " +dsc.shapetype+ " geometry\n")
except:
    outFile.write(data+ " is a "+dsc.datatype+" and h
```

describing - Notepad

File Edit Format View Help

```
***** Describe information for NorthwestRoadTrip.gdb *****

NorthwestRoadTrip.gdb is a Workspace and has no shapetype
The full path is C:\Data\Beth_Data\NorthwestRoadTrip.gdb
```

Describe to access the projection information

- SpatialReference is an object that contains all the spatial reference (projection) info
- Create it new, point to a .prj file, or grab it from a dataset with Describe

Learn more:
PSA Book Ch. 5.9-5.10
And the **arcpy Help**

```
import arcpy

# Method #1 - Point to a prj file
prjfile = r"D:\NR426-427\NR426 Materials\Lesson 2 - arcpy\Lesson2LabData\parks.prj"
spatialref = arcpy.SpatialReference(prjfile)
print(spatialref.name)

# Method #2 - Use describe
data = r"D:\NR426-427\NR426 Materials\Lesson 2 - arcpy\Lesson2LabData\parks.shp"
spatref = arcpy.Describe(data).SpatialReference
print (spatref.name)
```

Other SR properties to know:

- Name
- Type
- Factory Code
- LinearUnitName
- PCSName / GCSName
- DatumName

Accessing other properties

Learn more:
PSA Book Ch. 10: 10.4, 10.5

- Bounding Box (coordinates of extent, Minx, MinY, MaxX, MaxY)
  - For entire layer: Use Dataset describe properties `dsc.Extent` property
  - Individual features: Use Search Cursor, access Shape field `(ie row[0].extent.Xmin)`
- Raster information
  - From Raster Band describe properties or Raster object
    - **Cell size**: Mean Cell Height or Width
    - **Integer or Float**: isInteger
    - **Pixel Type** (aka Pixel Depth): pixelType
    - **Number of rows/columns**: height, width
  - From getRasterInfo
    - **Cell size**: getCellSize

```
rasobj = arcpy.Raster(inputraster)
print (rasobj.isInteger)
print (rasobj.extent.XMin)
```

```
rasobj = arcpy.Raster(inputraster)
print (rasobj.getRasterInfo().getCellSize())
```

Accessing properties sample

```
import arcpy
gdb = r"C:\Lesson2LabData\FtCollins.gdb"
arcpy.env.workspace = gdb
ras = r"FtCollinsWest_DEM"
fc = "parks"

print ("Getting values from Describe")
dsc = arcpy.Describe(ras) # Describes the raster
print(f"Mean cell height: {dsc.MeanCellHeight}") # Print cell size
print (f"X Min: {dsc.Extent.XMin}") # Prints the minimum X coordinate

print("Getting values from Raster object")
rasobj = arcpy.Raster(ras)  #Creates a raster object from the raster
print (rasobj.isInteger) # Prints if raster is an integer (not float)
print (rasobj.extent.XMin) # Prints the minimum X coordinate
print (rasobj.catalogPath) # Prints the folder path
print (rasobj.spatialReference.name) # Prints name of spatial reference
print ("\nGetting raster info")
print (rasobj.getRasterInfo().getCellSize()) #Creates raster info
                                              object and returns the cell size
#rinfo = rasobj.getRasterInfo()
# print (rinfo.getCellSize())
# print (rinfo.getExtent().XMin)

print ("\n*** Getting values for a FEATURE CLASS ***\n")
fdsc = arcpy.Describe(fc) # Describe the parks layer
print (fdsc.extent.XMin)  # Prints the minimum X coordinate of Parks
```

Demo

Click here to watch the Lessonb3B Demo video – Describing data with arcpy

- Listing other types of objects rasters, workspaces, fields
  - Listing fields returns the *fields* object. It cannot be directly printed. We have to call out one of its properties, then print that. Refer to the Using Fields help or the Field object help for more info.
  - From the N:\drive: Lesson3Part2DemoStarterCode.py
- Describing properties of data
- Pick a dataset from the class data folder/GDBs and describe and print some of these properties:

dataType	datasetType
catalogPath	basename
file	extent
shapeType	spatialReference

Follow-along Demo in class

1. If you start with this line of code

```
dsc = arcpy.Describe("UnionCity.gdb")
```

a. Write the code to determine the *type* of geodatabase

b. Write the code to retrieve the *path* to this geodatabase?

2. What code would you type to describe a Watersheds feature class?

a. Write the code to determine the geometry type

b. Write the code to return some *spatial reference* properties (type, name, unit)

3. Working with hasattr and .spatialreference

What did you just learn?

- What do the arcpy List methods do?
- What does Describe do?
- What List methods require the workspace environment to be set?
- How do we find the syntax and parameters for a given List method?
- Do all Describe properties apply to all data objects?
- What happens if we call a Describe property that doesn't apply?
- Is there a way to List all the data in a folder, including its subfolders?

Lab 3B –Putting it all Together

Write the code to loop through a geodatabase and:

Report some information about the contents

Clip or project data if needed

Advanced options available, if you want to be challenged