Watch Part 1 as a video:
Lesson 2A, Part 1: Script best practices, arcpy and running simple tools

Accessing geoprocessing

arcpy

Running tools

Basic arcpy methods

Putting it together

Python Scripting for ArcGIS book (Zandbergen), Chapter 5

Python for ArcGIS Pro (Toms/Parker), Chapter 2

NR426
Programming for GIS I
Lesson 2 (a, b)

Instructor:

Elizabeth Tulanowski | ESS Department

What does Lesson 2 cover?

- Two class sessions: Lessons 2a, 2b
- Access ArcGIS functionality through Python
- Understand capabilities of arcpy
- Implement non-tool methods within arcpy
- Run any Geoprocessing tool through Python

Lesson 2a

Writing a script

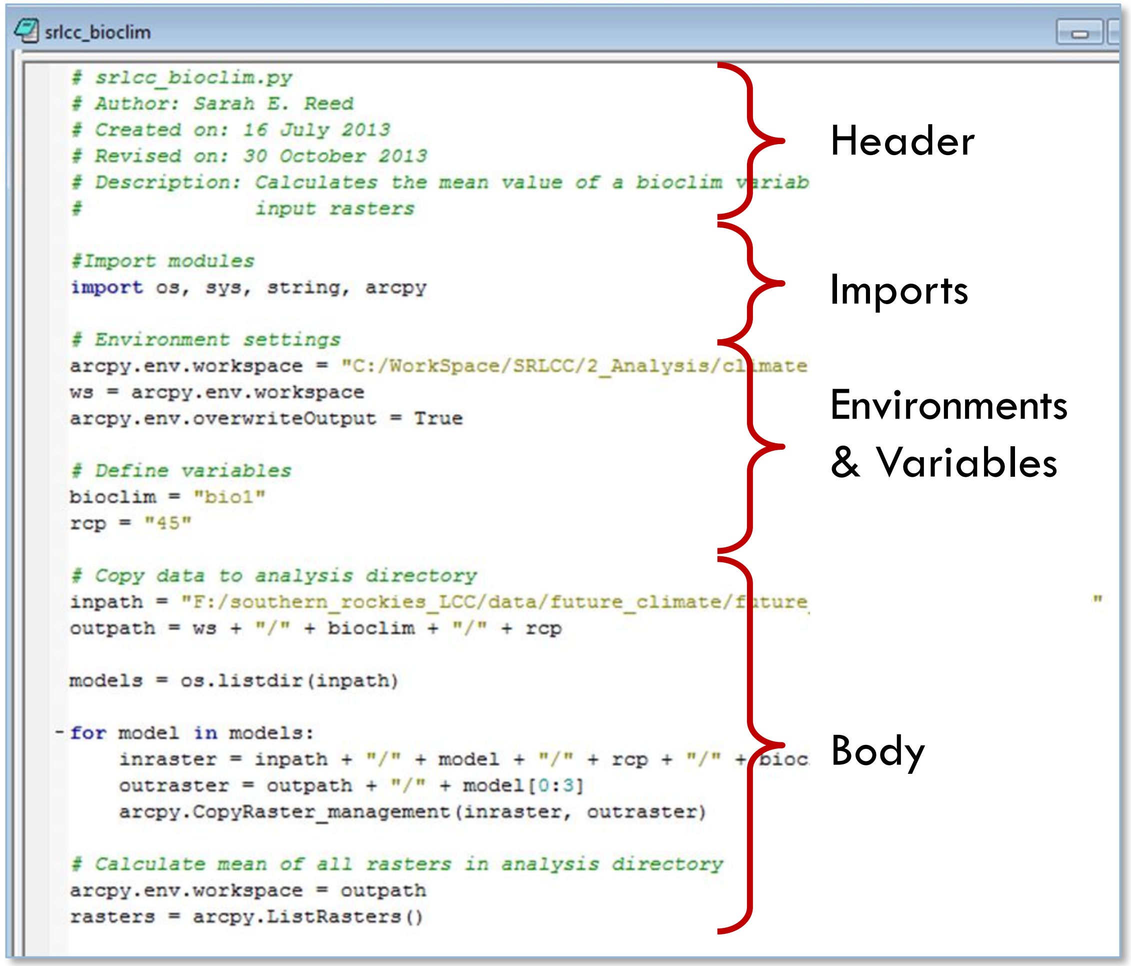
Introducing arcpy

Running tools

How do you start a script? What makes a good script?

[How to Write Beautiful Python Code With PEP 8 – Real Python]

- Pseudocode
  - Like an outline to help you organize script
  - Written as comments
  - Fill in the sections later with actual code
- Script components
  - Header
  - Variables
  - Body
  - Comments, Messaging



```
# srlcc_bioclim.py
# Author: Sarah E. Reed
# Created on: 16 July 2013
# Revised on: 30 October 2013
# Description: Calculates the mean value of a bioclim variab
#              input rasters

#Import modules
import os, sys, string, arcpy

# Environment settings
arcpy.env.workspace = "C:/WorkSpace/SRLCC/2_Analysis/climate
ws = arcpy.env.workspace
arcpy.env.overwriteOutput = True

# Define variables
bioclim = "bio1"
rcp = "45"

# Copy data to analysis directory
inpath = "F:/southern_rockies_LCC/data/future_climate/future_                    "
outpath = ws + "/" + bioclim + "/" + rcp

models = os.listdir(inpath)

for model in models:
    inraster = inpath + "/" + model + "/" + rcp + "/" + bioc
    outraster = outpath + "/" + model[0:3]
    arcpy.CopyRaster_management(inraster, outraster)

# Calculate mean of all rasters in analysis directory
arcpy.env.workspace = outpath
rasters = arcpy.ListRasters()
```

What is arcpy?

- A "site package" - a giant collection of GIS-related functions
  - Methods and properties to work with GIS data
  - Enables access to ArcGIS functionality: every tool in the toolbox plus additional non-tool methods, properties, and classes
- Must be imported into your script
  - Then you have access to ArcGIS functionality

```
import arcpy
arcpy.method()  #Tools
arcpy.env.property = value  #Env. settings
print arcpy.property
```

Ch. 5.1 – 5.3 (P. 154 in PSA book)

Step 1: Importing arcpy

- `import arcpy`
  - All code must be written as arcpy.method()

    Examples:

    ```
            arcpy.Buffer_analysis(in, out, dist)
            arcpy.analysis.Buffer (in, out, dist)
    ```
- `from arcpy import *`
  - Simplify code to call by method name without the “arcpy.”

    ```
    Buffer_analysis(in, out, dist)
    ```

Arcpy has modules within it for specific functions:

- `arcpy.mp, arcpy.sa, arcpy.na, arcpy.da`

  ```
  from arcpy.mp import *
  ```
  - Simplify code to call by method name without the “arcpy.”

Ch. 5.1 – 5.3 (P. 154 in PSA book)

Step 2: Setting a default workspace

```
arcpy.env.workspace = r“C:\Documents\NR426\Lesson2”
```

- Why?
  - Can reference data *without* the full path further down in script
  - Makes code shorter, easier to read
  - Not always required …but some methods (arcpy.List____ methods) do require it (More in Lesson 3)

Ch. 5.4 (P. 156 in PSA book)

With variables and a workspace vs. without

```python
# Import modules
import arcpy

# Set environments and variables
arcpy.env.workspace = r"C:\Student\ProgrammingPro\Databases\UnionCity.gdb"

census = "Census_Tract_2000"
watersheds = "Watersheds"
clip_out = "census_clipped"
print ("variables set")
print ("Your workspace is:" + arcpy.env.workspace)

# Clip(in_features, clip_features, out_feature_class, {cluster_tolerance})
print ("Starting clip.....")
arcpy.Clip_analysis(census, watersheds, clip_out)
```

```python
# Import modules
import arcpy

# Clip(in_features, clip_features, out_feature_class, {cluster_tolerance})
print ("Starting clip.....")
arcpy.Clip_analysis(r"C:\Student\ProgrammingPro\Databases\UnionCity.gdb\census”,
                    r"C:\Student\ProgrammingPro\Databases\UnionCity.gdb\watersheds”,
                    r"C:\Student\ProgrammingPro\Databases\UnionCity.gdb\clip_out”)
```

Ch. 5.5 (P. 160 in PSA book)

Step 3: Running tools

- Accessing geoprocessing functionality within a Python script (Help doc)
- Generic syntax to run any tool in the toolbox

  arcpy.Toolname_toolboxalias (arg, arg, arg)   **Required v. optional*

  -or- arcpy.toolboxalias.toolname (arg, arg, arg)
- Examples:

```
import arcpy
arcpy.Buffer_analysis (input, Bufoutput, 2)
arcpy.analysis.Clip(Bufoutput, “County”, clippedoutput)
```

- Every toolbox has an alias. Look at syntax in help.
- Tools, methods are case sensitive. Look at syntax in help.

Demo

- Importing arcpy
- Finding syntax for tools
- Running a tool

Watch this part as a video:
Lesson 2A, Part 2, Running tools that need Spatial Analyst

Running tools that need Spatial Analyst

- *Must* import the extension's module `import arcpy.sa`
- *Should* check that Spatial Analyst extension is available (More info)
- *Must* run the tool and save output with `.save()`

```
import arcpy, env
from arcpy.sa import *
if not arcpy.CheckExtension(“Spatial”) == “Available”:
     print (“License for SA tools not available”)
env.workspace = "C:/sapyexamples/data"
outSlope = Slope("elevation", "DEGREE", 0.3043)
outSlope.save("C:/sapyexamples/output/outslope01")
```

- *Maybe* check out the extension with `arcpy.CheckOutExtension(“Spatial”)`
  - Depends on licensing: only needed with concurrent or single-use licensing
- Help doc for the spatial analyst tools
- Other “sub-modules” of arcpy exist: Network Analyst, Data Access, Mapping

Ch. 5.13 and Ch. 10.3 -10.4 in PSA book

Demo

- Importing arcpy’s Spatial Analyst functionality
- Running a raster analysis tool

Watch this part as a video:
Lesson 2A, Part 3, Using the Help and a Brief Summary

ArcGIS Help system: Pro Python for Pro

Use the search

Tool help

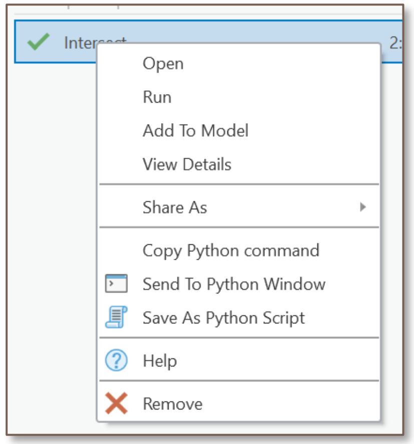
From the Toolbox:

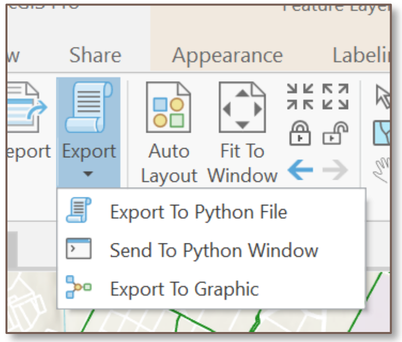
Right-click on tool, choose Help or Item Description

Ch. 5.14 (P. 185 in PSA book)

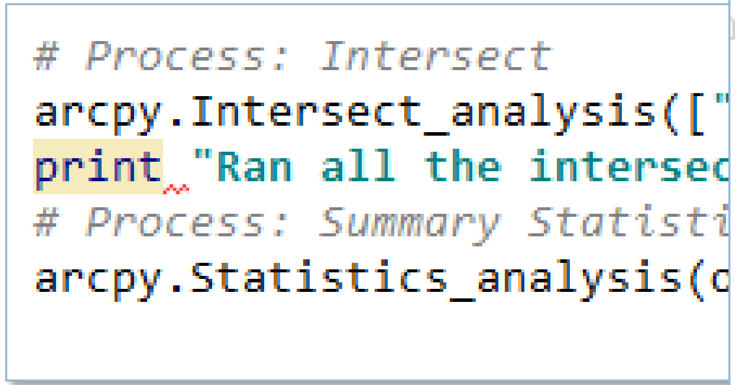
Help tips

- Copy – paste the syntax from the Help into the code so you have it in front of you
- Get the code snippet generated for you!
  - Run a tool
  - Go to the Analysis Ribbon > History
  - Find the tool you want to use in Python
  - Right-click and copy Python command
- Turn that ModelBuilder model into Python code
  - From the ModelBuilder ribbon > Export

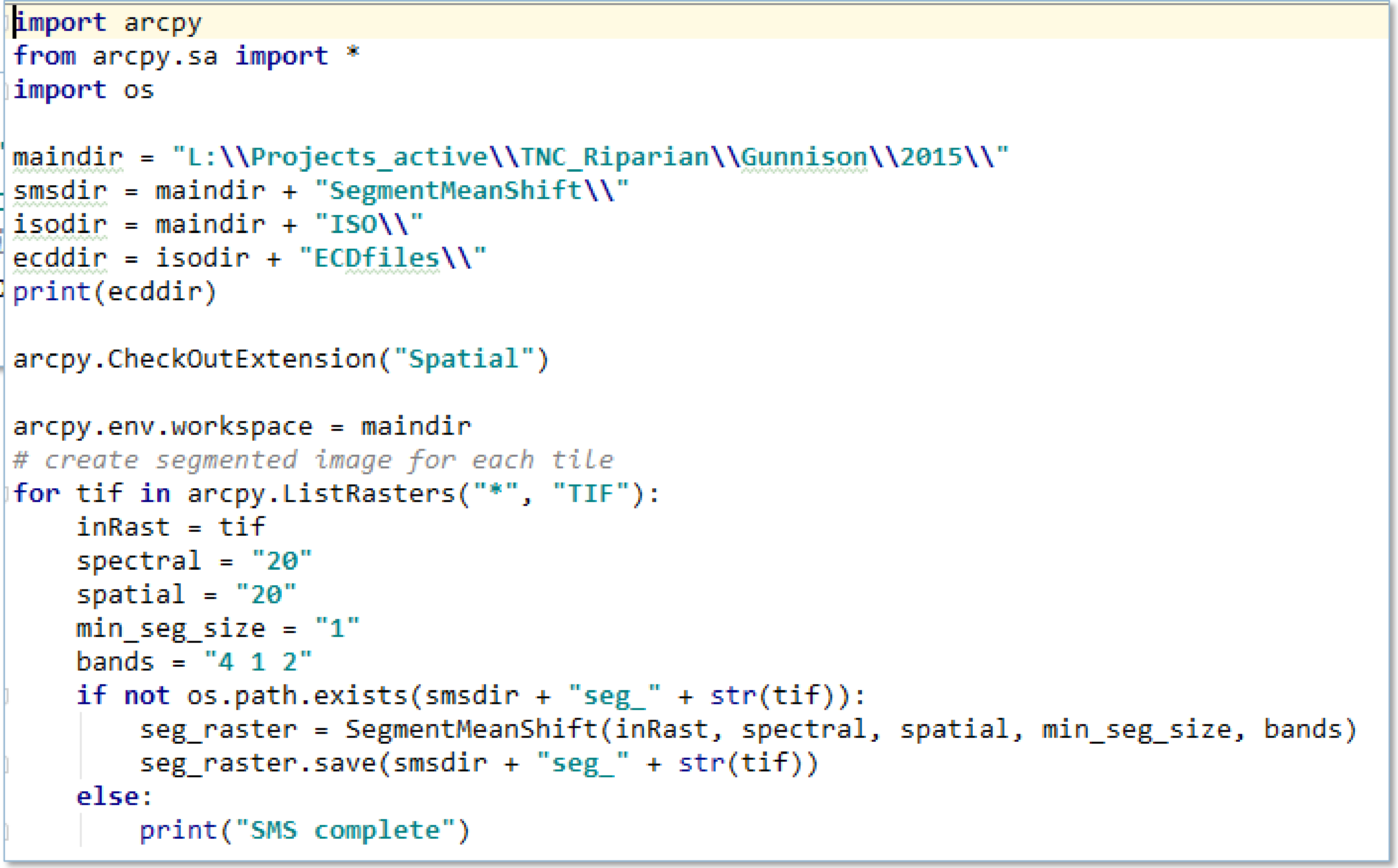




Running simple tools



```
# Process: Intersect
arcpy.Intersect_analysis(["
print "Ran all the intersec
# Process: Summary Statisti
arcpy.Statistics_analysis(c
```



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

maindir = "L:\\Projects_active\\TNC_Riparian\\Gunnison\\2015\\"
smsdir = maindir + "SegmentMeanShift\\"
isodir = maindir + "ISO\\"
ecddir = isodir + "ECDfiles\\"
print(ecddir)

arcpy.CheckOutExtension("Spatial")

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")
```

What did you just learn?

1. How do you find the syntax for a tool you're not familiar with?

2. How can you make your code more readable for yourself or others?

3. Where do your "print" messages go?

4. What is the generic syntax for any tool in the toolbox?

5. How can you tell if your script ran successfully?

Lab 2A: Accessing arcpy and running tools

- In-class hands-on activities
  - Finding syntax for tools
  - Setting up a script
  - Running tools in a script
- Self-directed activities
  - Write the code to perform some vector and raster analysis

Watch this part as a video:
Lesson 2B, Part 1, Other arcpy Methods

Lesson 2b

Other arcpy methods

- **Exists**
- **GetCount**
- **Messaging**
- Describe
  - …and more

arcpy Methods and Properties

- Methods on arcpy: `arcpy.method ()`
  - arcpy.Exists(data)
  - arcpy.Toolname(<arguments>)
- Properties: Environment settings, Describe
  - `arcpy.env.Environment = “value”`
  - `arcpy.env.workspace = “C:\NR426\LAB2”`
  - `arcpy.env.overwriteOutput = True`

- *Another overwrite option*
  - *Not arcpy…. From ArcGIS Pro Options*

Where's the Help for all the environment settings?
https://pro.arcgis.com/en/pro-app/arcpy/classes/env.htm
Search for *arcpy env class*

Useful arcpy methods (and 1 property)

Not from the Toolbox…

- overwriteOutput
- Exists
- Describe

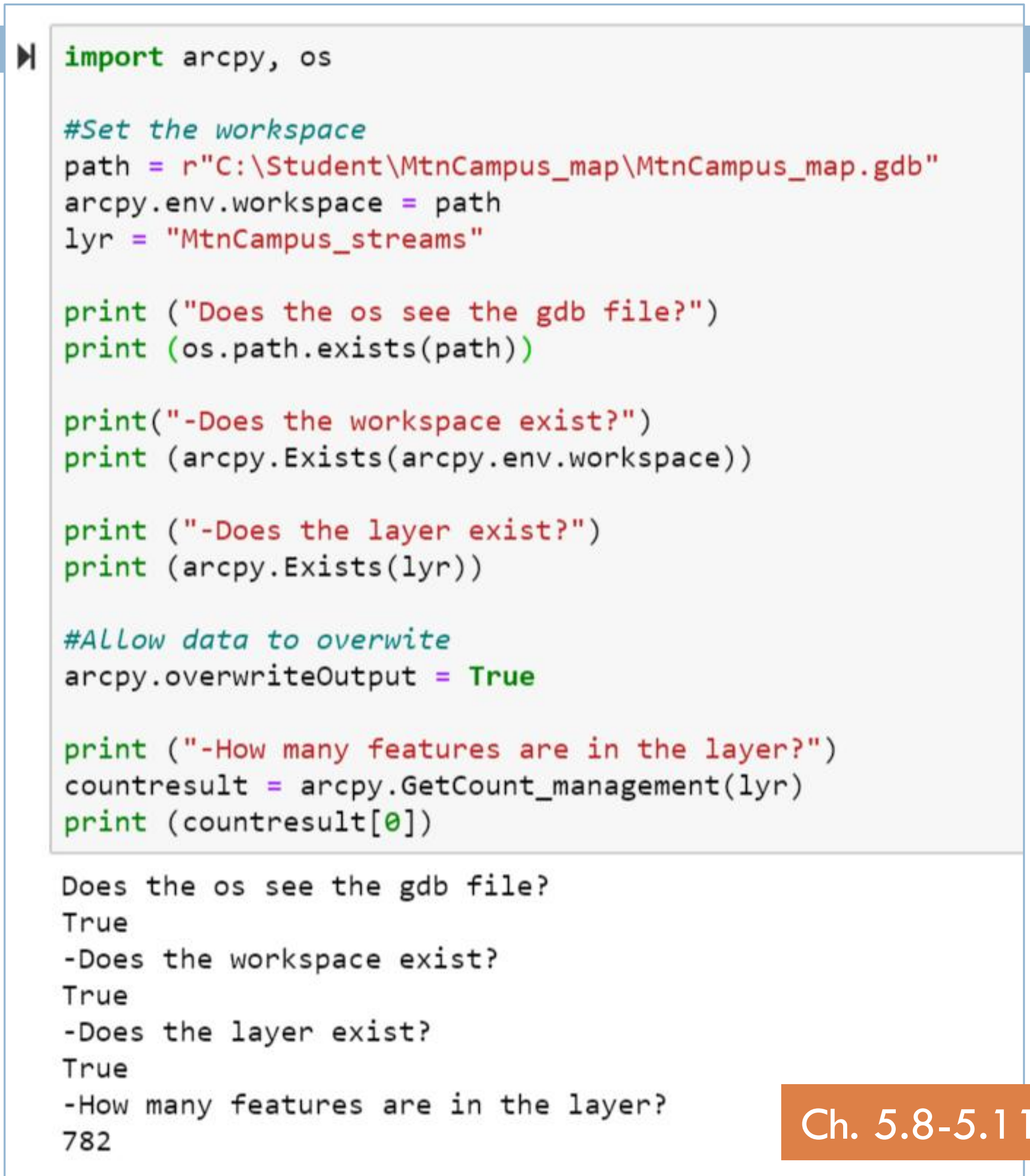
From the Toolbox…

- GetCount
- MakeFeatureLayer (Lesson 4)

There is also an Exists property from the os module.

- It works on <u>any file</u> os can see
- Won't recognize spatial data within a GDB

`os.path.exists`



```
import arcpy, os

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

print ("Does the os see the gdb file?")
print (os.path.exists(path))

print("-Does the workspace exist?")
print (arcpy.Exists(arcpy.env.workspace))

print ("-Does the layer exist?")
print (arcpy.Exists(lyr))

#Allow data to overwite
arcpy.overwriteOutput = True

print ("-How many features are in the layer?")
countresult = arcpy.GetCount_management(lyr)
print (countresult[0])
```

```
Does the os see the gdb file?
True
-Does the workspace exist?
True
-Does the layer exist?
True
-How many features are in the layer?
782
```

Ch. 5.8-5.11 in PSA book

Messaging

- Create your own messages for the geoprocessing window

```
arcpy.AddMessage(“your own text”)
```

  - `print` doesn’t work with script tools (Lesson 6), must use AddMessage
- Retrieve messaging from the geoprocessor

```
arcpy.GetMessages()
```

Ch. 5.12 in PSA book

Demo

- A quick look at:
  - Workspace
  - Overwrite
  - Exists
  - GetCount
  - AddMessage

Watch this part as a video:
Lesson 2B, Part 2, Describe and Lesson Conclusion

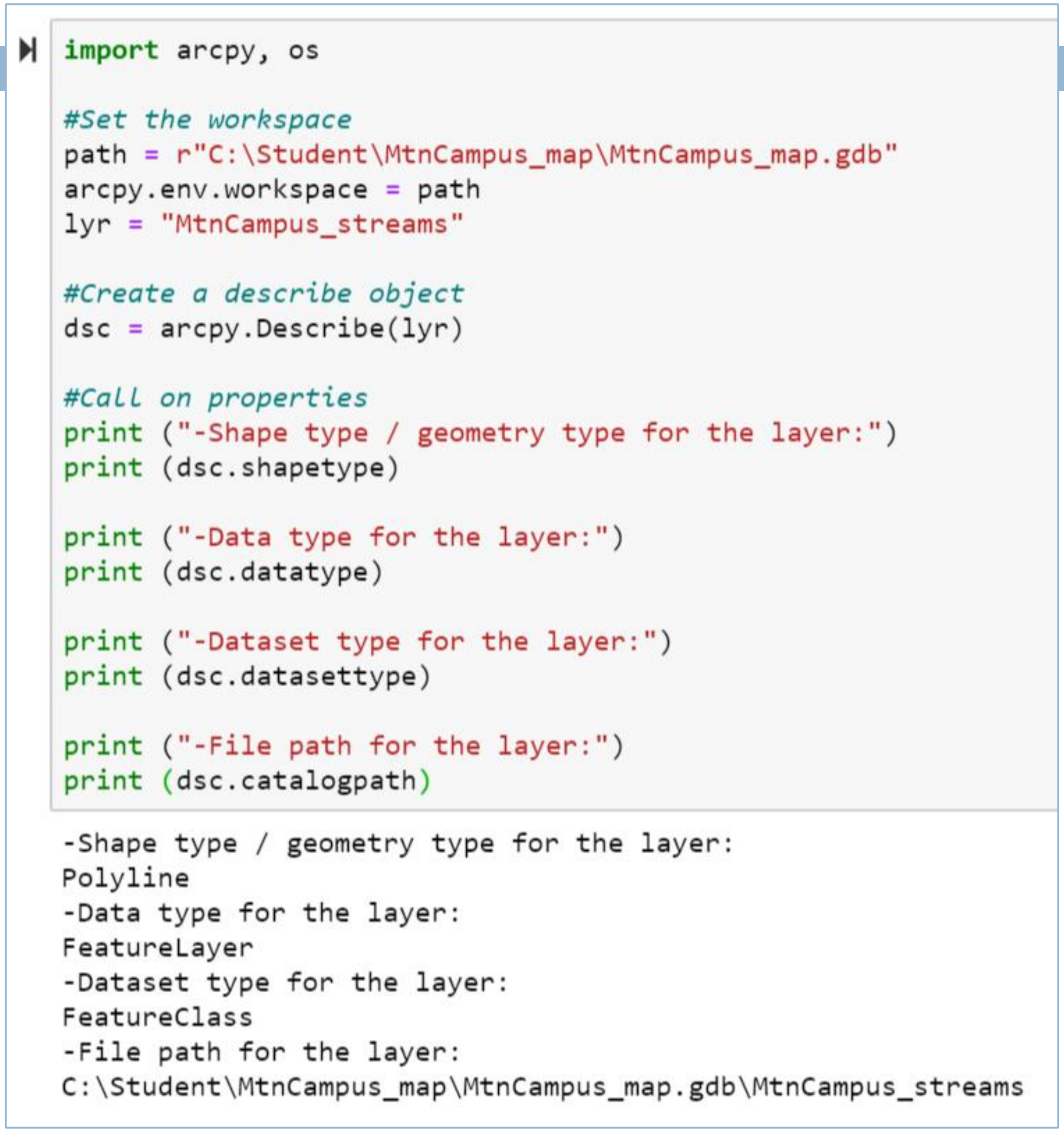
Lesson 2b

Other arcpy methods

- Exists
- GetCount
- Messaging
- **Describe**
  - …and more

Describing data with `arcpy.Describe()`

- Retrieve data properties
- Describe help doc
- All data have access to the Describe object properties
- More in Lesson 3



```
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)
```

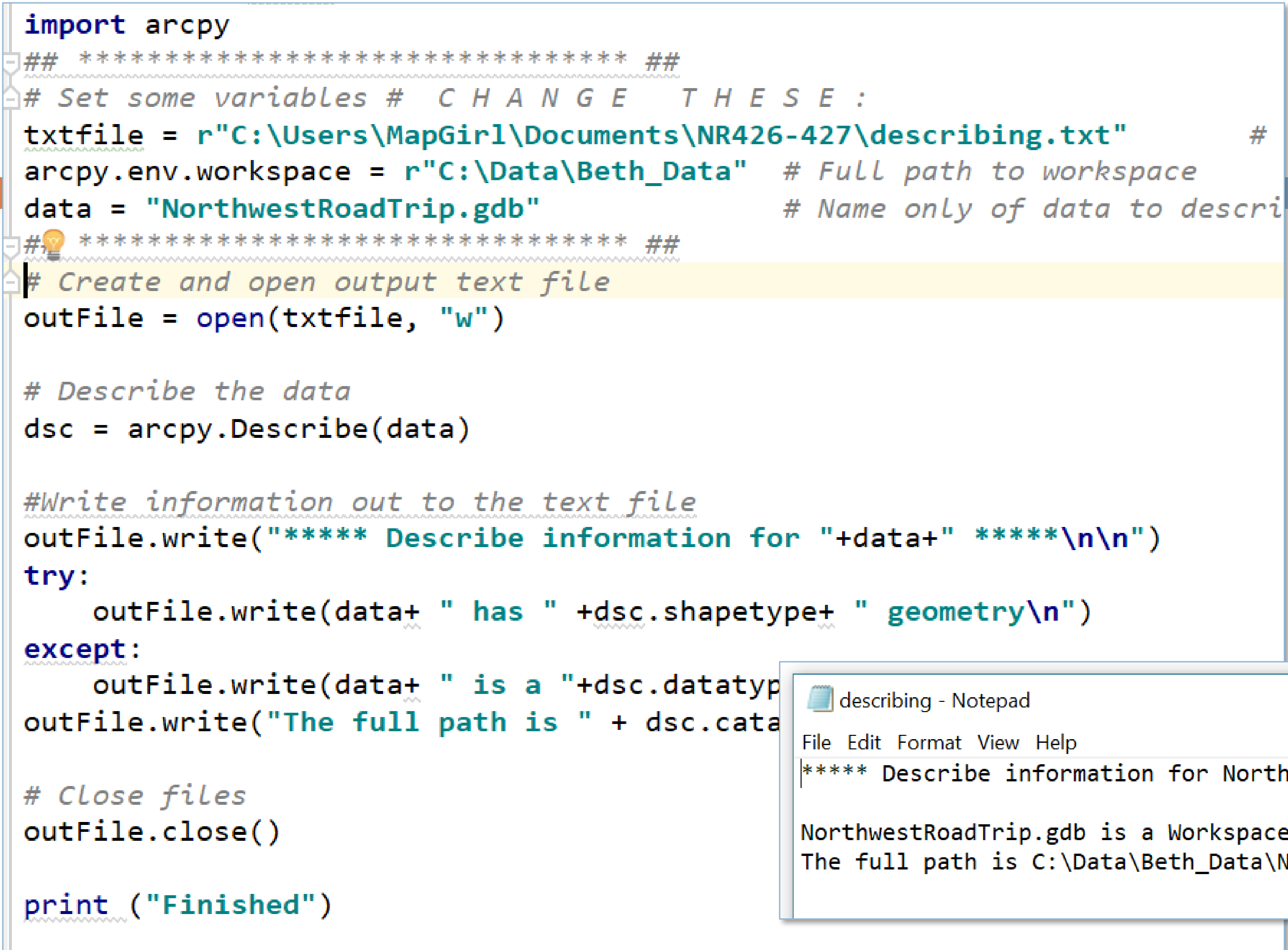
```
-Shape type / geometry type for the layer:
Polyline
-Data type for the layer:
FeatureLayer
-Dataset type for the layer:
FeatureClass
-File path for the layer:
C:\Student\MtnCampus_map\MtnCampus_map.gdb\MtnCampus_streams
```

Describing data with `arcpy.Describe()`

- Examples:

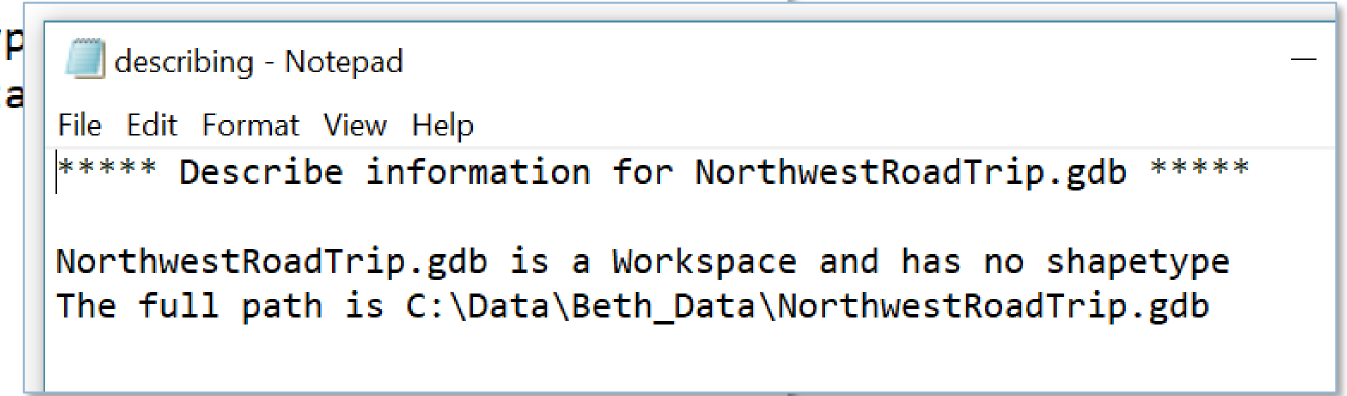
```python
import arcpy

# Create a Describe object from the feature class
#
desc = arcpy.Describe("C:/data/arch.dgn/Point")

# Print some feature class properties
#
print("Feature Type:  " + desc.featureType)
print("Shape Type :   " + desc.shapeType)
print("Spatial Index: " + str(desc.hasSpatialIndex))
```

```python
desc = arcpy.Describe("C:/Data/chesapeake.gdb")

# Print some Describe Object properties
#
if hasattr(desc, "name"):
    print("Name:        " + desc.name)
if hasattr(desc, "dataType"):
    print("DataType:    " + desc.dataType)
if hasattr(desc, "catalogPath"):
    print("CatalogPath: " + desc.catalogPath)
```



```
import arcpy
## ******************************** ##
# Set some variables # C H A N G E   T H E S E :
txtfile = r"C:\Users\MapGirl\Documents\NR426-427\describing.txt"       #
arcpy.env.workspace = r"C:\Data\Beth_Data"  # Full path to workspace
data = "NorthwestRoadTrip.gdb"              # Name only of data to descri
## ******************************** ##
# 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.datatyp
outFile.write("The full path is " + dsc.cata

# Close files
outFile.close()

print ("Finished")
```

DEMO



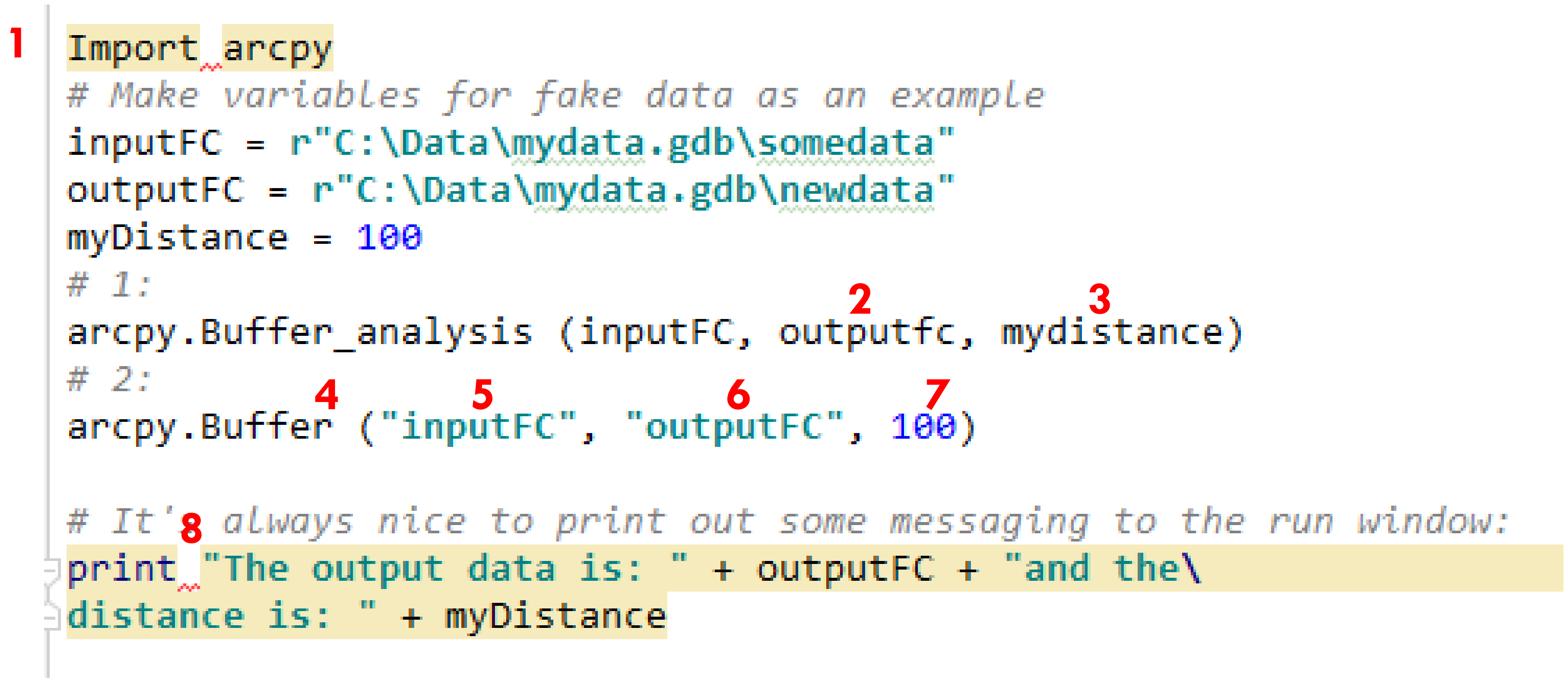
```
***** Describe information for NorthwestRoadTrip.gdb *****

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

Putting it together

```
Import arcpy
# Make variables for fake data as an example
inputFC = r"C:\Data\mydata.gdb\somedata"
outputFC = r"C:\Data\mydata.gdb\newdata"
myDistance = 100
# 1:
arcpy.Buffer_analysis (inputFC, outputfc, mydistance)
# 2:
arcpy.Buffer ("inputFC", "outputFC", 100)

# It's always nice to print out some messaging to the run window:
print "The output data is: " + outputFC + "and the\
distance is: " + myDistance
```

There are 8 mistakes/issues in this small snippet of code.
Can you find them?

Putting it together



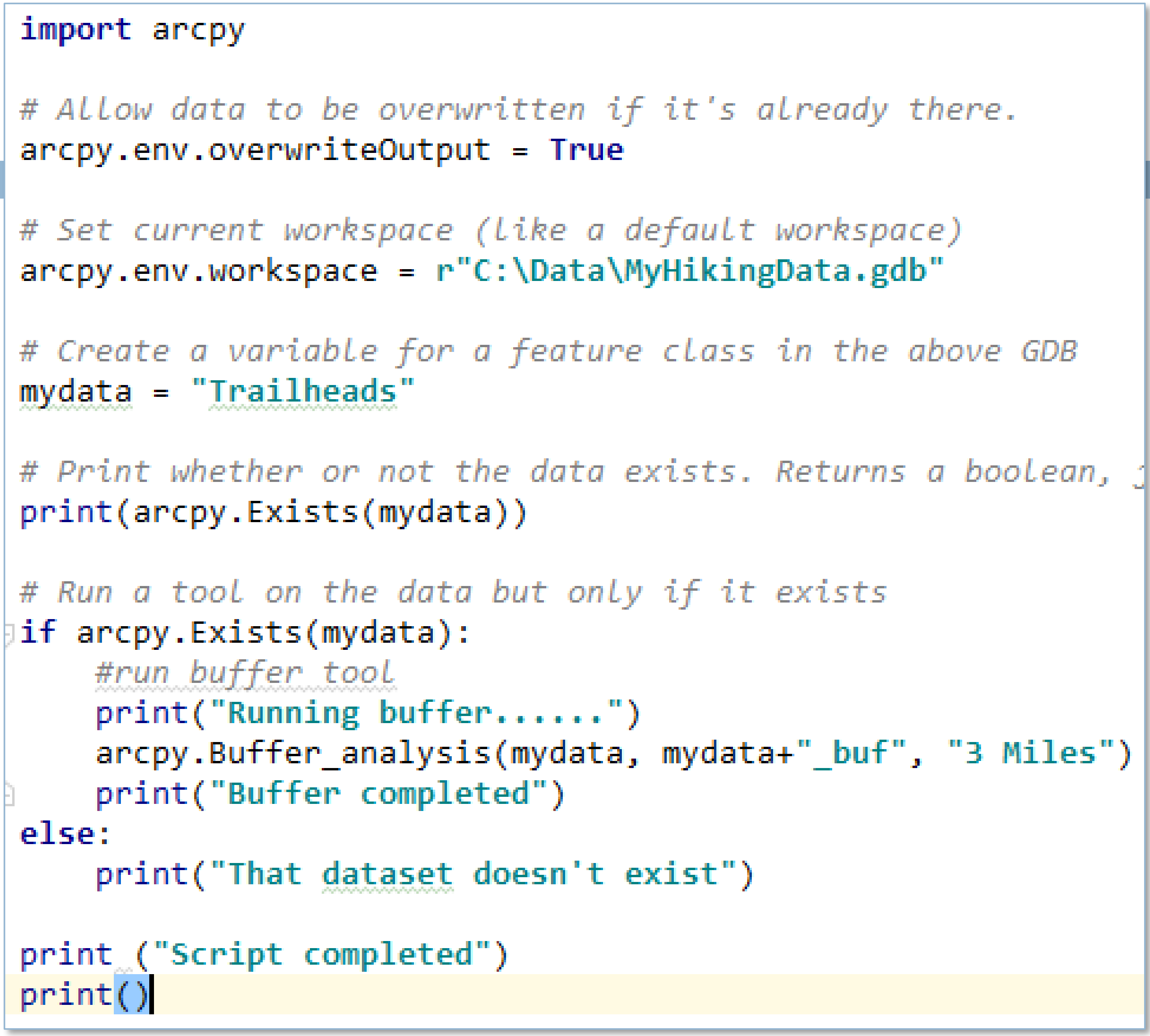
```
Import arcpy
# Make variables for fake data as an example
inputFC = r"C:\Data\mydata.gdb\somedata"
outputFC = r"C:\Data\mydata.gdb\newdata"
myDistance = 100
# 1:
arcpy.Buffer_analysis (inputFC, outputfc, mydistance)
# 2:
arcpy.Buffer ("inputFC", "outputFC", 100)

# It's always nice to print out some messaging to the run window:
print "The output data is: " + outputFC + "and the\
distance is: " + myDistance
```

Code example

What is each line doing?

Could you write something comparable on your own?

- Variables
- if
- arcpy



```
import arcpy

# Allow data to be overwritten if it's already there.
arcpy.env.overwriteOutput = True

# Set current workspace (like a default workspace)
arcpy.env.workspace = r"C:\Data\MyHikingData.gdb"

# Create a variable for a feature class in the above GDB
mydata = "Trailheads"

# Print whether or not the data exists. Returns a boolean,
print(arcpy.Exists(mydata))

# Run a tool on the data but only if it exists
if arcpy.Exists(mydata):
    #run buffer tool
    print("Running buffer......")
    arcpy.Buffer_analysis(mydata, mydata+"_buf", "3 Miles")
    print("Buffer completed")
else:
    print("That dataset doesn't exist")

print ("Script completed")
print()
```

Lab 2B: Working with arcpy methods

- In-class hands-on activities
  - Run a tool on data only if it exists
  - Allow data to overwrite
  - Add messaging
- Self-directed activities
  - Write code to run basic geoprocessing while exploring some additional arcpy properties and methods