Advanced Python for ArcGIS

Description

Building on Python Programming Basics for GIS Professionals but open to anyone with some programming experience, this workshop will expand on those skills to further use Python in ArcGIS. The workshop will focus on the ArcPy Python site package to expand geoprocessing capabilities with Python scripts. Participants will learn to build multiple standalone geoprocessing scripts covering different GIS tasks and workflows. The workshop will also cover how to create scripting tools in ArcGIS toolboxes for reuse and sharing. Participants will finish with the skills to explore more resources and options for utilizing Python in ArcGIS.

Specific Topics Include:

- Work with the ArcPy Python site package for ArcGIS
- Convert ModelBuilder models to Python scripts
- Build and share stand-alone Python script tools for automation
- Learn tips and tricks for valid script syntax and error handling

Computing and Software Needs

- Your own laptop; computers will not be provided.
- ArcGIS Desktop 10.x (Standard or Advanced preferred)
- Python IDE
  - PyScripter (http://sourceforge.net/projects/pyscripter/files/) (v. 2.6.0 **32-bit** version)
    - *NOTE:* The zip files contain portable versions of PyScripter. No installation is needed. Just unzip the archive and start using PyScripter.
  - IDLE (https://docs.python.org/2/library/idle.html) is also shipped with ArcGIS for Desktop, so it can also be used

Instructor

James Whitacre, GIS Specialist, University Library, University of Illinois at Urbana-Champaign

As the GIS Specialist in the Main Library at the University of Illinois at Urbana-Champaign, Mr. Whitacre's primary role is to provide GIS consultation and research assistance for faculty, staff, and students. Additionally, he teaches a myriad of GIS workshops for beginner to advanced users and helps manage the Library's GIS data and software assets. He is also a central resource for the GIS community on campus to promote the use of GIS in research. Mr. Whitacre holds a Master of Science in Geography and was previously the GIS Manager for the Carnegie Museum of Natural History.

I. Introductions and Outline

Outline

Data and Software Setup

Review of *Introduction to Python for ArcGIS*

Working with Files in Python

- Reading Files with Python
- `csv` Module
- Reading CSV files

Standalone Scripts with Python

- Planning a Standalone Script (i.e. Pseudo Code)
- Search Cursors
- Writing CSV files

Break

Python vs. ModelBuilder

- Convert ModelBuilder Tools to Python Scripts

Automating Geoprocesses with Standalone Scripts

- Handling Errors and Debugging

Adding Python Script Tools to Toolboxes

Conclusion and Moving Forward

II. Data and Software Setup

Download Exercise Data

https://uofi.box.com/ILGISA-Python (https://uofi.box.com/ILGISA-Python)

- Extract the data to a folder on the Desktop or other well-known folder
- For folks who can't download the files, we have it on a flash drive

Python IDE: PyScripter

- Download PyScripter (http://sourceforge.net/projects/pyscripter/files/) (v. 2.6.0 **32-bit** version)
- Open PyScripter: Two Ways...
  - Install it like a normal program
    - Download **PyScripter-v2.6.0-Setup.exe**, run the install file, open program
    - Admin privileges needed
  - Run the portable version
    - Download **PyScripter-v2.6.0.zip**, extract, and run **PyScripter.exe**
    - Admin privileges *NOT* needed
- Already using a Python IDE? If you feel comfortable in your own envirionment, please use it, but there is no guarantee that issues can be supported
- If all esle fails, you may need to use IDLE, which ships with ArcGIS Desktop and is installed automatically

Type the code below and run it.

I will give explanation of concepts, then have code examples that can be run in the IDE in the grey boxes.

Please *type* the code as we go and try to avoid copy and pasting unless instructed otherwise. Typing the code will help you learn it better!

In [ ]:
```
print("hello world")

# Note: Using just `print` without `()` is proper syntax in Python 2.x, but ge
tting in the habit of usng `print()`
#     will prepare you for Python 3.x and ArcGIS Pro, which is the only way `p
rint()` will work
```

III. Review of *Introduction to Python for ArcGIS*

Review of minimal skills needed for those who didn't attend first session

IV. Working with Files in Python

Reading Files with Python

For more commands and options see:

- https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files (https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files)
- https://www.tutorialspoint.com/python/python_files_io.htm (https://www.tutorialspoint.com/python/python_files_io.htm)

In both cases, start with the file path as a string variable.

Modes	Description

In [ ]:
```
# replace the '...' with the location where you placed your downloaded data

filename =  r'C:\...\Data\CSV\JeopardyContestants_LatLon.csv'

# filename = r"C:\Users\jvwhit\Documents\ILGISA\Data\Answers_Adv\CSV\JeopardyContestants_LatLon.csv"

print(filename)
```

Now we need to make our file object/variable. We will print the result to see what it looks like

In [ ]:
```
f = open(filename, 'r')

print(f)
```

Notice that it indicating that it is an open file (the object)

File Access Modes

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

*See: https://www.tutorialspoint.com/python/python_files_io.htm (https://www.tutorialspoint.com/python/python_files_io.htm)

Now we can read in the file. We can see the raw text values using a `print()` statment

In [ ]:
```
data = f.read()

print(data)
```

In [ ]:
```
# Don't forget to close the file!

f.close()
print(f)
```

Reading Lines in a File

This time, we will use the with statement

In [ ]:
```
with open(filename, 'r') as f:
    data = f.readlines()

# Is the file open or closed now
print(f)
# OR
print(f.closed)

# Print the data
print(data)
```

What type of data does the `.readline()` method produce?

Is the output very readable?

Let's make it more readable using a `for` statement

In [ ]:
```
for e in data:
    print(e)
```

Check out those extra blank lines. This can be an artifact from coming in from Excel.

When you see this, use `strip()` to clean things up. This function cleans off any white space characters from both the left and the right of a line of text.

There are also `rstrip()` and `lstrip()` in case those would be useful.

In [ ]:
```
for e in data:
    print(e.strip())
```

`csv` Module

What is the CSV module?

Two core functions:

- **`csv.reader(file object)`** to create a reading object
- **`csv.writer(file object)`** to create a writing object.

To write files, use these two functions:

- **`writerow(row)`** to write a single row
- **`writerows(list of lists)`** to write many rows

Below are some basic formulas for reading in data. Many of these formulas came from chapter 6 of the Python Cookbook (third edition) (http://chimera.labs.oreilly.com/books/1230000000393). Not all of this syntax will make sense, but just treat it as a formula for now. Things will make more sense as you learn more about Python.

You can read the chapter for free here: http://chimera.labs.oreilly.com/books/1230000000393/ch06.html (http://chimera.labs.oreilly.com/books/1230000000393/ch06.html)

In [ ]:
```
import csv

# Open the File
with open(filename, 'r') as file_in:

    # Read the file using the csv module reader
    file_in = csv.reader(file_in)

    # Read the first line to store the headers as a list
    headers = next(file_in) # The next() method reads the line as a list and m
oves to the next line of the csv file

    # Create an empty list to store the data
    data = []

    # Loop through the remaining lines and append each line as a list element
 to the data list
    for line in file_in:
        data.append(line)

print('Headers: {}'.format(headers))
print('Data: {}'.format(data))
```

In [ ]:
```
for l in data:
    print l
```

Now this looks like something we can start dealing with...

Each row is coming in as a list, with each cell value as an element within that Nth corresponding position.

Let's do some quick sanity checking...

In [ ]:
```
for row in data:
    print(len(row))
```

Is each row the same length?

If it looks good, we can now loop through and get all the contestant names...

In [ ]:
```
for row in data:
    print( row[1])
```

There are many ways to get at the data from a CSV file. What I've shown here is one of the simplest possible mechanisms. There are more advanced methods for more complex data.

We will cover writing CSV files in a little bit...promise!

V. Standalone Scripts with Python

Scenario:

For many of your GIS workflows, you need to create an Excel file of feature classes and standalone tables as a final standalone file to share with your colleagues. The ArcGIS **Table to Excel (http://desktop.arcgis.com/en/arcmap/latest/tools/conversion-toolbox/table-to-excel.htm)** tool has been working great! But, you recently learned that your colleagues have been exporting the Excel files to CSV and they would like to elminate this step from their workflow. Therefore, they have asked you deliver files as CSV files instead. You scour ArcMap for a ***Table to CSV*** tool, but can't find it (Believe me, it doesn't exist! There are tools that get close, but not quite!). Therefore you decide to make a script that will export tables to CSV files.

This is a great example for when a simple standalone script would solve a common problem

Planning a Standalone Script (i.e. Pseudo Code)

What is Pseudo Code?

General Components of a Script

Indentation is shown for formatting purposes only

In [ ]:
```
# Script Info and Metadata

    #-------------------------------------------------------------------------
------
    # Name:        <ScriptName>.py
    # Purpose:     <Some important purpose.>
    # Author:      <Name, Your>
    # Created:     DD/MM/YYYY
    # Copyright:   (c) <Your Name> YYYY
    # Licence:     <licence>
    #-------------------------------------------------------------------------
------

# Import Modules
    # arcpy and other modules that may help you complete your code

# Parameters
    # What inputs and outputs will the user have contorl over?

# Environments
    # Gets the house in order for how data and outputs will be dealt with

# Variables
    # Anything that is not a parameter thay may be used multiple times
    # These may also be dispersed throughout the script

# Processes
    # The code that dictates the workflow
```

Create Pseudo Code for the Stand-alone Script

1. Create a new script in your IDE

2. Add the pseudo code below as comments in the script

3. Save it to the 'Scripts' folder named as *TableToCSV.py*

In [ ]:
```
#-------------------------------------------------------------------------
--
# Name:         TableToCSV.py
# Purpose:      <Some important purpose.>
# Author:       <Name, Your>
# Created:      17/10/2016
# Copyright:    (c) <Your Name> 2016
# Licence:      <licence>
#-------------------------------------------------------------------------
--

# Import Modules

# Parameters

# Environments

# Variables

# Processes
```

4. What are the steps needed to create the script (i.e. the pseudo code in plain language)?

Pseudo Code

- Read in a feature class or standalone table
- Read the feature class or standalone table field names
- Read the data
- Write the field names to a new CSV file
- Write the data to the new CSV file

5. Add the pseudo code to the `# Processes` seciton of the script

In [ ]:
```
# Processes

# Read in a feature class or standalone table
# Read the feature class or standalone table field names
# Read the data
# Write the field names to a new CSV file
# Write the data to the new CSV file
```

6. What modules and user parameter inputs and outputs are needed?

Add the following `import` statements and parameter variables to your script *above the previous pseudo code*

In [ ]:
```
# Import Modules
import arcpy
import csv

# Parameters

# Path to the input feature class or standalone table
# Replace the '...' with the location where you placed your downloaded data
# input_table = r"C:\...\Data\Illinois.gdb\Illinois_Counties"
input_table = r"C:\Users\jvwhit\Documents\ILGISA\Data\Answers_Adv\Illinois.gdb
\Illinois_Counties"

# Path to the output CSV file (this CSV file will not exist)
# Replace the '...' with the location where you placed your downloaded data
# output_csv = r"C:\...\Data\CSV\Illinois_Counties.csv"
output_csv = r"C:\Users\jvwhit\Documents\ILGISA\Data\Answers_Adv\CSV\Illinois_
Counties.csv"
```

7. Write the code

First, we will read the table and grab the field names

In [ ]:
```
# Processes

# Read in a feature class or standalone table
# Read the feature class or standalone table field names
field_list = [field.name for field in arcpy.ListFields(input_table)]

print(field_list) # Do this to make sure it works
```

Now we can read the data...but first we need to learn about a new method, the *Search Cursor*

Search Cursors

`with arcpy.da.SearchCursor(input_table, field_list) as cursor:`

*Note: There are two types of cursors in ArcPy, one directly in ArcPy (e.g. `arcpy.SearchCursor(dataset)`) and the other in the Data Access module within ArcPy (e.g. `arcpy.da.SearchCursor(dataset)`). The Data Access module vesion of cursors is newer and faster and is recommended over the normal cursors. Also, the Data Access search cursor requires a list of fields, whereas for the normal cursor a field list is optional.

In [ ]:
```
# Processes

# Read the data
data = []
with arcpy.da.SearchCursor(input_table, field_list) as cursor:
    for row in cursor:
        data.append(row)

print(data[:10]) # Print the first 10 rows to check the data


# Write the field names to a new CSV file
# Write the data to the new CSV file
```

What do you notice about the data? Is it what you expected?

**Below is what your code should look like to this point:

In [ ]:
```
# Import Modules
import arcpy
import csv


# Parameters

# Path to the input feature class or standalone table
# Replace the '...' with the location where you placed your downloaded data
# input_table = r"C:\...\Data\Illinois.gdb\Illinois_Counties"
input_table = r"C:\Users\jvwhit\Documents\ILGISA\Data\Answers_Adv\Illinois.gdb
\Illinois_Counties"

# Path to the output CSV file (this CSV file will not exist)
# Replace the '...' with the location where you placed your downloaded data
# output_csv = r"C:\...\Data\CSV\Illinois_Counties.csv"
output_csv = r"C:\Users\jvwhit\Documents\ILGISA\Data\Answers_Adv\CSV\Illinois_
Counties.csv"


# Processes

# Read in a feature class or standalone table
# Read the feature class or standalone table field names
field_list = [field.name for field in arcpy.ListFields(input_table) if field.n
ame != "Shape"] # Added an if statement

print(field_list) # Do this to make sure it works

# Read the data
data = []
with arcpy.da.SearchCursor(input_table, field_list) as cursor:
    for row in cursor:
        data.append(list(row)) # Added a list function to conver the tuple

print(data[:10]) # Print the first 10 rows to check the data


# Write the field names to a new CSV file
# Write the data to the new CSV file
```

Now it's time to write the data to a CSV file...but we need to remeber how to write the data (I promised you I would do this!!).

Review: Writing CSV Files

- **`csv.writer(file object)`** to create a writing object
- **`writerow(row)`** to write a single row
- **`writerows(list of lists)`** to write many rows

In [ ]:
```
# Processes

# Write the field names to a new CSV file
# Write the data to the new CSV file
with open(output_csv, 'wb') as csv_file: # For some reason, 'w' add an extra l
ine between rows...

    csv_file = csv.writer(csv_file) # Creates a CSV writer object
    csv_file.writerow(field_list) # Writes out one line: the fields
    csv_file.writerows(data) # Writes out all data from the list of lists

print("Finished! Open the file located at {}".format(output_csv)) # Lets you k
now that the script is finished
```

Challenge:

How might you cahgne the code to write out the field aliases instead of field names?

How might you give the user an option to choose between aliases and names?

Break

Python	ModelBuilder

IV. Python vs. ModelBuilder

Scenario:

You developed a model in **ModelBuilder** for a common geoprocessing workflow that many people use in your organization. However, your fellow GIS users are asking for some more sofisticated functionality that ModelBuilder cannot perform. Therefore, you think Python will do the trick! How can you assess this whether the ROI is worth the effort?

Comparing Python and ModelBuilder

Python	ModelBuilder
Textual programming language	Visual Programming language
Easy to learn with very flexible structure	Relativiely easier to learn for GIS beginners, but restrictive structure
Python scripts can be used in ModelBuilder	ModelBuilder can export to Python (but not vice versa)
Lower-level geoprocessing tasks (e.g. cursors)	Lower-level geoprocessing tasks may not be possible
More advanced error handling	Errors handled by the tools in the model
Better for iterations and loops	Can do iterations and loops, but can be difficult to setup properly
Can use other Python modules and wrap other software (e.g. R)	Restricted by ArcToolbox tools
Can run Python scripts outside of ArcGIS	ModelBuilder works when ArcGIS is running

Converting ModelBuilder Tools to Python Scripts

If you decide you want to create a script, you can start by export a ModelBulider model tool to a Python script. This will give you a great head start!

Open the Model Named 'Illinois in Jeopardy' in edit mode (right-click > Edit...)

Convert the Model to a Python Script

Open the new script in the IDE

V. Automating Geoprocesses with Standalone Scripts

Senario:

Your supervisor is a huge **Jeopardy!** fan (he watches it every night!). To make matters worse, he has compiled a list of all of the **Jeopardy!** contestants on his nerdy website (It's probably a Blogspot site from 2003!). He's learnign GIS so he was able to find latitude and longitude values for contestants' home towns. He shared his CSV file with you containing all the contestants from the state of Illinois and has asked you to map them out and determine how wealthy their home county is and how many contestants came from each county.

Your supervisor has recently discovered the beauty of geodatabases, as opposed to Shapefiles, so he definitely wants a file geodatabase feature class. But, you know he is prone to changing his mind, and you suspect he will ask you for a Shapefile the next time (and he will probalby want all of his contestants mapped the next time!).

Since you took this GIS Python workshop, you decide it might best for you write a script...and one that allows him to choose which format, shapefile or a FGDB feature class. You also need to impressive your boss so your job isn't in Jeopardy! (Is that going too far...?).

1. Create a new script in your IDE

2. Add the pseudo code below as comments in the script

3. Save it to the 'Scripts' folder named as *IllinoiInJeopardy.py*

In [ ]:
```
#-------------------------------------------------------------------------------
--
# Name:        IllinoiInJeopardy.py.py
# Purpose:     <Some important purpose.>
# Author:      <Name, Your>
# Created:     17/10/2016
# Copyright:   (c) <Your Name> 2016
# Licence:     <licence>
#-------------------------------------------------------------------------------
--

# Import Modules

# Parameters

# Environments

# Variables

# Processes
```

4. What are the steps needed to create the script (i.e. the pseudo code in plain language)?

Pseudo Code

- Copy CSV table to geodatabase stand-alone table
- Convert table to XY Layer
- Make the XY Layer a permanent point feature class or shapefile based on user input
- Intersect the points with counties layer and include all fields from both layers
- Sum the number of contestants from each county and total winnings and output as a table

5. Add the pseudo code to the `# Processes` seciton of the script

In [ ]:
```
# Processes

# Copy CSV table to geodatabase stand-alone table

# Convert table to XY Layer

# Make the XY Layer a permanent point feature class or shapefile based on user
 input

# Intersect the points with counties layer and include all fields from both la
yers

# Sum the number of contestants from each county and total winnings and output
 as a table
```

6. What modules and user parameter inputs and outputs are needed?

Add the following `import` statements and parameter variables to your script *above the previous pseudo code*

In [ ]:
```
# Import Modules
import arcpy


# Parameters

# Inputs:
# CSV File with XY coordinates
# Replace the '...' with the location where you placed your downloaded data
csv_file = r'C:\...\Data\CSV\JeopardyContestants_LatLon.csv'

# Fields that holds the lat and long values
lon = 'lon' # this is the x value
lat = 'lat' # this is the y value

# Spatial reference CSV XY coordinates
csv_spatialref = arcpy.SpatialReference('WGS 1984')

# Counties feature class
counties = 'Illinois_Counties'

# Outputs:
# Contestants Intersect Feature Class or Shapefile
fc_output = 'JeopardyContestants_Income'

# Sum of Contestants by County Table
tbl_output = 'JeopardyContestants_ByCounty'
```

7. Set the Environment Settings

Now we're ready to write our script! But first, we need to do some house keeping!

Review: Environmental Settings

See http://desktop.arcgis.com/en/desktop/latest/tools/environments/what-is-a-geoprocessing-environment.htm (http://desktop.arcgis.com/en/desktop/latest/tools/environments/what-is-a-geoprocessing-environment.htm)

See http://desktop.arcgis.com/en/desktop/latest/tools/environments/environment-levels-and-hierarchy.htm (http://desktop.arcgis.com/en/desktop/latest/tools/environments/environment-levels-and-hierarchy.htm)

In [ ]:
```
# Execute this code to see the result...

print(arcpy.ListEnvironments())

# Comment out the code or delete it when done
```

Add a workspace as an environment

Add the following code below the **`# Environments`** comment

In [ ]:
```
# Environments

# Replace the '...' with the location where you placed your downloaded data
arcpy.env.workspace = r'C:\...\Data\Illinois.gdb'
arcpy.env.overwriteOutput = True
```

8. Add Geoprocessing Tools

Type the following code in the IDE script after the matching pseudo code:

In [ ]:
```
# Copy CSV table to geodatabase stand-alone table

xy_table = 'JeopardyContestants_Table'

arcpy.CopyRows_management(csv_file, xy_table)
```

Why type when you can copy from an exisitng script!

In [ ]:
```
# Convert table to XY Layer

xy_layer = 'Jeopardy Contestants'

arcpy.MakeXYEventLayer_management(xy_table, lon, lat, xy_layer,
csv_spatialref)
```

Test the tool in ArcMap, then 'Copy as Python Snippet'

On your Own, add the code for last two processes using any method

Handling Errors and Debugging

Three types of errors in Python

What is debugging?

Add some `print()` statements to the code

In [ ]:
```
# Copy CSV table to geodatabase stand-alone table

xy_table = 'JeopardyContestants_Table'

arcpy.CopyRows_management(csv_file, xy_table)

print('{} converted to table.'.format(arcpy.Describe(csv_file).name))
```

Handeling Errors Using conditional statements

- Replace the following code for the identified process
- Note the indents after the **`if:`** statement
- Try to test the error by changing a field name

In [ ]:
```
# Processes
# ...

# Convert table to XY Layer

field_list = [field.name for field in arcpy.ListFields(xy_table)]

if lat in field_list and lon in field_list:
    print('{} and {} fields exist'.format(lat, lon))

else:
    print('{} or {} fields DO NOT exist. Check field names.'.format(lat, lon))

xy_layer = 'Jeopardy Contestants'

arcpy.MakeXYEventLayer_management(xy_table, lon, lat, xy_layer)

print('XY Layer created')
```

Handeling Errors Using `try:` and `except:`

- **`try:`** and **`except:`** statements are used for when you don't want your script to stop if there is an error

```
try:
      # Some code that might have an error...
      print("two" + 2)

  except:
      print("Error!")
```

- Note the indents after the **`try:`** and **`except:`** statements

Add the following `try:` and `except:` statement before executing the processes to test the error.

In [ ]:
```
# Processes

try:

    # Indent all processes...

    # Copy CSV table to geodatabase stand-alone table

    xy_table = 'JeopardyContestants_Table'

    arcpy.CopyRows_management(csv_file, xy_table)

    print(arcpy.Describe(csv_file).name + ' converted to table.')

    # Convert table to XY Layer

    field_list = [field.name for field in arcpy.ListFields(xy_table)]

    if lat in field_list and lon in field_list:
        print('{} and {} fields exist'.format(lat, lon))
    else:
        print('{} or {} fields DO NOT exist. Check field names.'.format(lat, l
on))
```

Add the following geoprocessing tool and the except statements at the end of the script.

In [ ]:
```
# Sum the number of contestants from each county and total winnings and output
 as a table

    arcpy.Frequency_analysis(fc_output, tbl_output, 'NAME_1', 'totalwinnings')

    print('Contestants by county summary complete. Script complete.')

# Print any errors from any geoprocessing tools if they fail
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
    print('Could not complete script')

# Add a general error print statement
except:
    print('Could not complete script')
```

VI. Adding Python Script Tools to Toolboxes

Senario:

Your supervisor so impressed by your script that he has asked you map his ***ALL*** the **Jeopardy!** contestants! You expect this will be aregular occurance, so now you want to figure out how to get that script into an ArcGIS toolbox for him and others to use.

Creating and sharing scripts with custom ArcGIS toolboxes

Two ways to run a script:

Advantages to using script tools over standalone scripts:

Steps to creating a tool

Note: You will also notice options for Python Toolboxes. These are more advanced and allow additional capabilities. However, they are beyond the scope of this workshop.

Help Documentation: http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/a-quick-tour-of-creating-tools-in-python.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/a-quick-tour-of-creating-tools-in-python.htm)

Modify Script Inputs and Outputs to Be Used in the Script Tool

- Inputs and outputs use the **`arcpy.GetParameterAsText(<index>)`** funtion to pass variable between the tool and the script.
- All inputs and outputs must be modified to utilize tools

1. In the IDE, save a copy (i.e. save as...) of your stand-alone script and add '_ScriptTool.py' to the name

2. Modify the scripts code to align with the code below

- Part 1: Inputs and outputs converted to user input parameters
- Part 2: Environments are managed by the tool
- Part 3: Replace all Print Statements with **`arcpy.AddMessage`**; this prints messages to the Results window in ArcMap

In [ ]:

```
# Parameters

# Inputs:
# CSV File with XY coordinates
csv_file = arcpy.GetParameterAsText(0) #r'C:\...\Data\CSV\JeopardyContestants_
LatLon.csv' # Replace the '...' with the location where you placed your downlo
aded data

# Fields that holds the lat and long values
lon = arcpy.GetParameterAsText(1) # 'lon' # this is the x value
lat = arcpy.GetParameterAsText(2) #'lat' # this is the y value

# Spatial reference CSV XY coordinates
csv_spatialref = arcpy.GetParameterAsText(3) # arcpy.SpatialReference('WGS 198
4')

# Counties feature class
counties = arcpy.GetParameterAsText(4) #'Illinois_Counties'

# Outputs:
# Contestants Intersect Feature Class or Shapefile
fc_output = arcpy.GetParameterAsText(5) # 'JeopardyContestants_Income'

# Sum of Contestants by County Table
tbl_output = arcpy.GetParameterAsText(6) # 'JeopardyContestants_ByCounty'


# Environments
# Managed by the tool, so we can comment out
#arcpy.env.workspace = r'C:\...\Data\Illinois.gdb' # Replace the '...' with th
e location where you placed your downloaded data
#arcpy.env.overwriteOutput = True


# ..............................

# Below is an example; be sure to change all print statements

# Convert table to XY Layer

    field_list = [field.name for field in arcpy.ListFields(xy_table)]

    if lat in field_list and lon in field_list:
        arcpy.AddMessage('{} and {} fields exist'.format(lat, lon))

    else:
        arcpy.AddMessage('{} or {} fields DO NOT exist. Check field names.'.fo
rmat(lat, lon))
```

Add a Script to a Toolbox

- Inputs and outputs are actually passed as text strings
- The script tool allows for those text strings to be pre-validated before using them in the script
- This is essentially a form of error handeling

1. In the Catalog window in ArcMap, right-click on the custom toolbox and hover over 'Add' and click 'Script...'

This will activate the Add Script wizard

2. Add/Edit script tool properties

- Name: IllinoisInJeopardyScript
- Label and Description: Illinois in Jeopardy Script (Final)
- Check 'Store relative path names...'
- Check 'Always run in foreground' (for now...)

3. Add the script file (.py)

- Add the script ending in '_ScriptTool.py'.
- Check 'Run Python script in process'

4. Add inputs and outputs

- Reference the script for all inputs and outputs
- Add descriptitive names - spaces are ok
- Choose the correct and expected data types for the inputs and outputs
- Set up the appropriate parrameter properties

Hints:

- For the CSV file, note that there are many data types that could be used.
  - What is the best one?
  - You may need to experiment
  - It is helpful to check the help documentation of the tools you are using to help determine data types
- For the lat and lon fields:
  - Filter the input to number fields only
  - You can read in the fields from the CSV file to create a drop-down list of usable fields
- Yes, there is a spatial reference data type!
- For the county layer with income, there is more than one data type choice. Which is the best?
- Don't forget to change the Direction property for outputs

5. Run and test the script tool

- What messages were returned?
- Learn how to add warning and error messages to the tool output: http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/writing-messages-in-script-tools.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/writing-messages-in-script-tools.htm)
- Try outputting a shapefile and DBF table in a folder
- Did it work? If it didn't, how can you fix this?

6. Edit and debug the tool as needed

- Shapefiles???
- Are there any outputs you don't want to keep?
- Try adding code to delete those unwanted feature classes
- Or learn about the `in_memory` workspace: http://desktop.arcgis.com/en/desktop/latest/analyze/modelbuilder/the-in-memory-workspace.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/modelbuilder/the-in-memory-workspace.htm)

- If there are no errors now, just wait until people start using it a lot...they'll find them if they exist! This is an iterative process

7. Share the tool

- Make sure the script and/or script folder is easily accessible by the toolbox and tool (i.e. let relative paths work for you)...Zip and ship!
- Try embedding the script

Help Documentation: http://desktop.arcgis.com/en/desktop/latest/analyze/sharing-workflows/a-quick-tour-of-sharing-tools.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/sharing-workflows/a-quick-tour-of-sharing-tools.htm) Oldie, but a goodie: http://help.arcgis.com/en/arcgisdesktop/10.0/help/#/A_structure_for_sharing_tools/005700000004000000/ (http://help.arcgis.com/en/arcgisdesktop/10.0/help/#/A_structure_for_sharing_tools/005700000004000000/)

Conclusion and Moving Forward

Exapnd on today's scripts

- Learn about modifying the script tool documentation and metadata: http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/a-quick-tour-of-documenting-tools-and-toolboxes.htm (http://desktop.arcgis.com/en/desktop/latest/analyze/creating-tools/a-quick-tour-of-documenting-tools-and-toolboxes.htm)
- Try to combine the morning and afternoon final scripts to create a 'mega' script

Start using python in your everyday workflows!

- Resist the temptation to fall back on ModelBuilder
- Read Python and ArcPy documentation...and read it again!
- Collaborate with and try to teach colleagues
- Get involved with a local development group...Esri has some

In [ ]:
```
print("Thank you! Goodbye!")
```