Python for ArcGIS Advanced

Description

Building on Python ArcGIS Introduction, 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. Participants will learn to build multiple standalone geoprocessing scripts covering different GIS tasks and workflows. The workshop will also cover how to create custom script 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
- Explore the difference between Python scripts and ModelBuilder models
- Build and share custom ArcGIS script tools for automation
- Learn tips and tricks for validating script syntax and error handling

Outline

I. Introduction

Top

Instructor

James Whitacre

Chief, GIS Services Division | Pennsylvania Game Commission

jawhitacre@pa.gov

James Whitacre is Chief of the GIS Services Division of the Pennsylvania Game Commission where he leads, manages, and provides vision for the Agency's geospatial and mapping program. Formerly, he was the GIS Research Scientist for the Carnegie Museum of Natural History where he managed the GIS Lab at Powdermill Nature Reserve, the Museum's environmental research center, and supported museum staff and affiliated researchers with geospatial technologies and needs. Whitacre was also the GIS Manager from 2011 to 2014 at the Museum. Before returning to the Museum in 2018, Whitacre was the GIS Specialist for the Main Library at the University of Illinois at Urbana-Champaign where he provided GIS consultations for researchers and scholars, and taught GIS workshops to promote the use of GIS in research. Whitacre holds a Bachelor of Arts in Zoology from Ohio Wesleyan University and a Master of Science in Geography, concentrating on GIS and cartography, from Indiana University of Pennsylvania. Whitacre is past board member and Past President of Keystone GIS (formerly PaMAGIC).

Emily Clees

Geospatial Specialist II, Northcentral Region, GIS Services Division | Pennsylvania Game Commission

eclees@pa.gov

Emily Clees is a Geospatial Specialist II for the Northcentral Region of the Pennsylvania Game Commission where she completes quality control checks on collected data, runs analysis for regional and state-wide projects, and creates tools to help visualize data collected in the Pennsylvania Game Commission. Prior to her role as a Geospatial Specialist, she received a Bachelor of Science in Environmental Resource Management with a minor in GIS and a Master of Science in Forest Resources from Penn State. Emily also received a GIS certificate from Penn State World Campus.

Welcome (Back) to Python Bootcamp!

- That is what this will feel like!! And that is what is intended!
- Lots of information in a short amount of time...your brain will hurt!
- You may feel lost at first...practice and perseverance will help (I will try to go slow!)
- Have patience...it will take time for it all to settle into your head...and heart!

Philosophy Over Practice

- I am not a traditional Python developer...my journey comes only through GIS
- I will be presenting a philosophy of how to approach Python in ArcGIS
- The focus will be on:
  - Foundational principles
  - Methods for how to systematically approach a project that would benefit from Python
- There will be some practice today, but it will be minimal
- Practice will be on YOU after the workshop!!! You will only get better if you dive in and start implementing Python in your everyday work.

Why do I teach Python?
Because of Volvo and Seatbelts!

II. Data and Software

Downloads

- Go to repo at **https://github.com/whitacrej/Python-For-ArcGIS**
- Click **Code** then click **Download Zip**
- **Extract** zip file to your Desktop or other well-known folder

ArcGIS Pro

- Recommended to use latest version 3.4+ or 2.9+
- Minimum version of 2.7+ required

See **ArcGIS Pro Documentation**

ArcGIS Pro vs. ArcMap

- Stop using ArcMap! Seriously...it's time! ArcMap is currently in mature support and is being retired March 2026 (see ArcMap Life Cycle)
- Oh, you HAVE to use it? Stop...but...
    - Nearly everything in this workshop still applies!!!
    - Except:
        - ArcMap does not use ArcGIS Notebooks...Use a code editor or the Python Window instead
        - ArcMap uses Python 2, which is being deprecated...just like ArcMap!
        - I'll point out a few other things that are different along the way...
- No, sorry, I don't have time to address every ArcMap concern...stop using it...

ArcGIS Notebooks

- Built on top of **Jupyter Notebook**
- Included and integrated with ArcGIS Pro (starting at version 2.5)
- Optimized for ArcGIS Pro and Python 3
- Used to easily and quickly write and run code directly in ArcGIS Pro

Python in ArcGIS Notebooks

- Can perform analysis and immediately view results in a geographic context, interact with the emerging data, document and automate your workflow, and save it for later use or share it
- ALL Python functionality in ArcGIS Pro is available through ArcGIS Notebooks
- Provides access to content in your map allowing for interactive workflows

Markdown and HTML in ArcGIS Notebooks

- ArcGIS Notebooks utilize Markdown and HTML markup languages to format Markdown cells in a Notebook
- Markdown:
    - Lightweight markup language that you can use to add formatting elements to plaintext text documents
    - One of the world's most popular markup languages
    - Markdown syntax is added to text to indicate which words and phrases should look different
- HTML:
    - Stands for Hyper Text Markup Language
    - The standard markup language for creating Web pages

- Describes the structure of a Web page and tell the browser how to display the content
- This Notebook uses Markdown and HTML **EXTENSIVLEY** to make it look the way it does!
- We will not go over Markdown in detail, but will a little bit

See:

- **ArcGIS Notebooks Get Started**
- **The Markdown Guide**
- **W3 Schools HTML Tutorial**

Create a new ArcGIS Notebook

- Click the **Insert** tab, and click the **New Notebook** dropdown, and click 📝 **New Notebook**
- In the **Catalog Pane**, **right-click on the folder** where you want to create it, click **New** (at the top), the click 📝 **Notebook**

We will be using ArcGIS Notebooks for nearly all our coding int he workshop!!!

**I will go over tips and tricks as we go!**
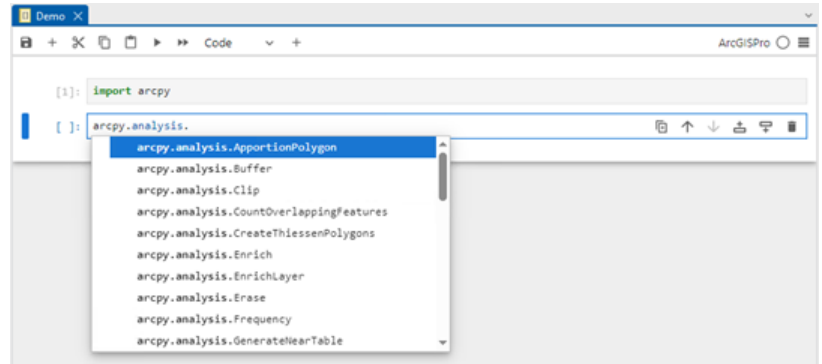
Code Editors

Visual Studio Code (or VS Code for short)

- Great all purpose code and text editor...an essential application for coding in numerous languages!
- Lightweight but powerful desktop source code editor
- Available for Windows, macOS and Linux
- Comes with built-in support for JavaScript, TypeScript and Node.js
- Rich ecosystem of extensions for other languages (such as C++, C#, Java, **Python**, PHP, Go) and runtimes (such as .NET and Unity).
- Requires some application savvy-ness to unleash full potential

*Requires special setup for use with ArcGIS Pro...but it is easy!
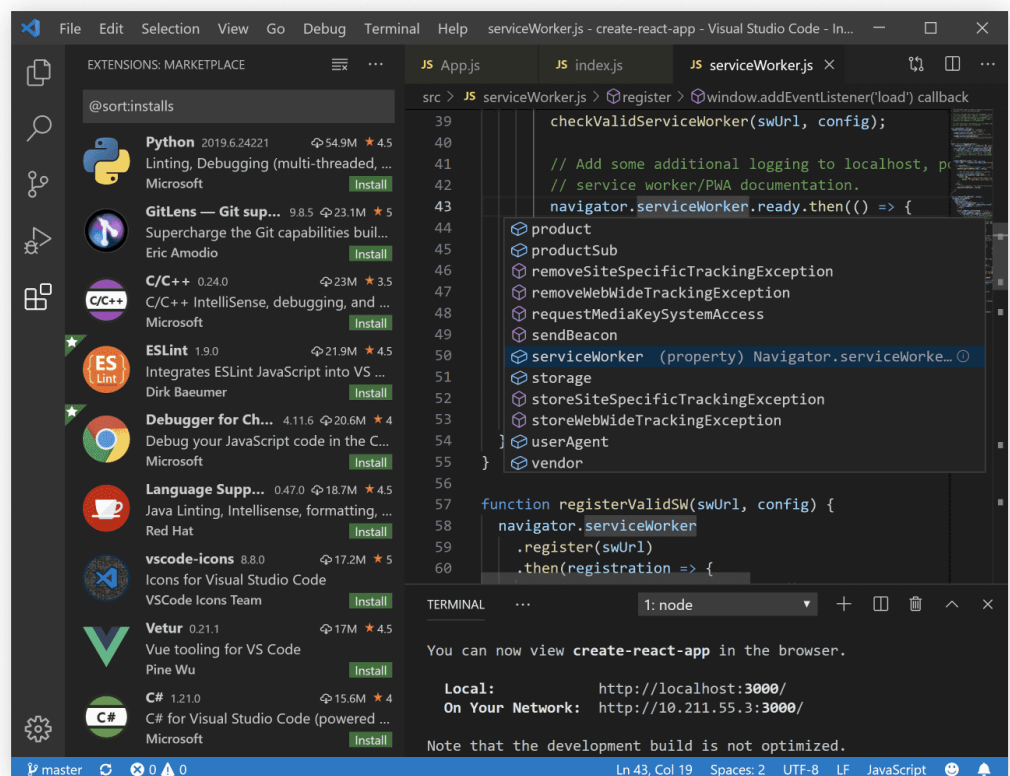
Download Visual Studio Code

See:

- **Getting Started with Python in VS Code**
- **How to configure Visual Studio Code with ArcGIS Pro's Python Environment**
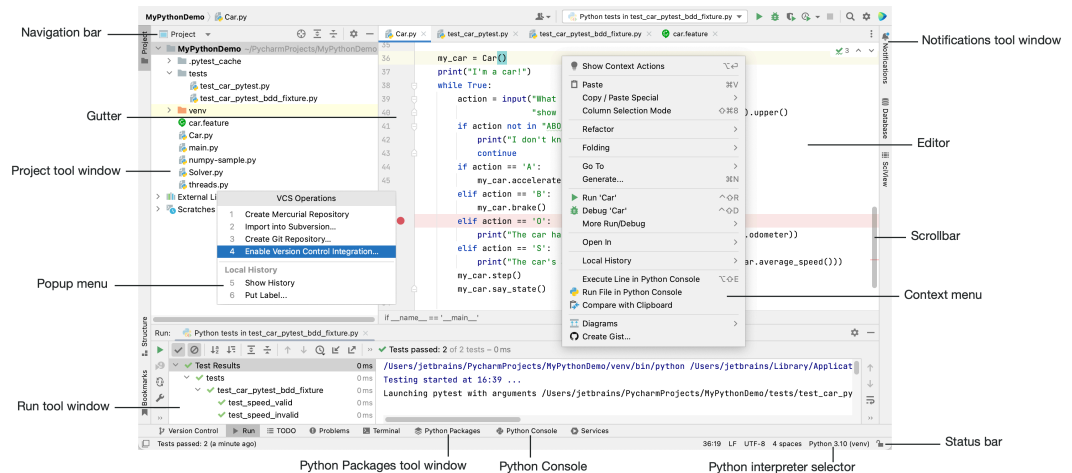
This is my favorite code editor!!!

PyCharm (Community Edition)

- Good for all around Python 3 development
- Can be installed without admin rights
- Easily run code in the application, but requires some setup

Download PyCharm

See:

- **PyCharm Setup for ArcGIS Desktop**
- **Configuring PyCharm for use with ArcGIS Pro (and ArcMap)**

Notepad++

- Great all purpose text editor for Windows
- Free (as in "free speech" and also as in "free beer") source code editor and Notepad
- Comes with built-in support for JavaScript, TypeScript and Node.js
- Rich ecosystem of extensions for other languages (such as C++, C#, Java, **Python**, PHP, Go) and runtimes (such as .NET and Unity).
- Highlights Python syntax so can be used to develop Python scripts
- Not very easy to run code in the application

Download Notepad++

Note sure how to set it up for use in ArcGIS Pro
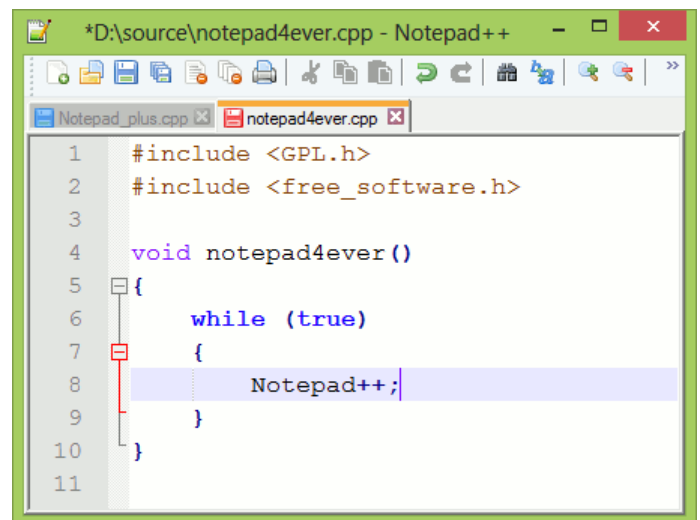
III. *Python for ArcGIS Introduction* Quick Review

Top

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

Python Basics

- What is Python?
- Print Function
- Variables
- Basic Data Types: Strings, Numbers, Booleans, Lists, Tuples, Dictionaries
- Data Type Conversions
- Simple Math with Python
- Python Syntax and Style
- Conditional Statements and Decision Making
- Loops
- Functions
- Classes and Methods

Calculate Fields Using Python

Introduction to ArcPy

- What is ArcPy?
- Modules vs. Site Packages
- `import` Statements
- ArcPy Help Documentation

Using ArcPy

- Describing Data
  - System Paths vs. Catalog paths
  - Formatting Strings
- Listing Data
  - Geoprocessing Environment Settings
  - List Comprehensions
- Geoprocessing Tools

IV. Working with Files in Python

Reading Files with Python

- When files are read with Python, files need to be stored as a variable
- There are two methods of opening a file:
  - **`open(file_name, access_mode)`** and **`.close()`**
    - This method requires you to close the file
  - **`with open(file_name, access_mode) as file:`** ***code indented below...***
    - This method closes the file automatically
- We will go over both, but will work with the `with` statement the most

For more commands and options see:

- **Python Documentation: Reading and Writing Files**
- **Tutorials Point: Python - File Handling**

In both cases, start with the file path as a string variable. Let's learn a trick to help construct our file paths

```
In [ ]:   # Get the ArcGIS Project Home (i.e. Default) folder path
          # See: ArcPy Help: ArcGISProject Class Documentation: https://pro.arcgis.com/en/pro-app/latest/arcpy/mapping/arcgisproject-class.htm

          import arcpy

          aprx = arcpy.mp.ArcGISProject('CURRENT')

          aprx_folder = aprx.homeFolder

          print(aprx_folder) # Check to ensure the folder is correct
```

```
In [ ]:   # Get the file name for the CSV file

          csv_file =  fr'{aprx_folder}\CSV\JeopardyContestants_LatLon.csv' # f = format string; r = raw string

          print(csv_file)
```

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

```
In [ ]:   # Open the CSV file

          f = open(csv_file, 'r') # 'r' is the access_mode

          print(f.closed)
```

Notice that it is indicating that the file (i.e. the object) is open.

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.

- **Tutorials Point: Python - File Handling**

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

```
In [ ]:  # Read the CSV file text data
         data = f.read()

         print(data)
```

```
In [ ]:  # Get the data type of the file data

         print(type(data))
```

```
In [ ]:  # Close the file

         f.close()

         print(f.closed)
```

Reading Lines in a File

- Many ways to read files, and well worth some investigation time...but not today!
- There is a better, very common method to read a file line by line: `.readlines()`

This time, we will use the `with` statement

```
In [ ]:  # Open the CSV file and read the text data line by line
         with open(csv_file, 'r') as f:
             data = f.readlines()

         # Check if the file open or closed
         print(f.closed)
```

```
In [ ]:  # Print the data

         print(data)
```

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

Is the output more or less readable than before?

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

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

Check out those extra blank lines. This is an artifact from exporting the data from Excel.

We can use `.strip()` string method 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()` methods in case those would be useful.

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

`csv` Module

What is the CSV module?

- `csv` module knows what a CSV is, how the text should behave and look, and how it should be sanitized, etc.
- Inclueds helper functions for working with CSV files to help the programmer save time
- When opening a CSV file, the `csv` module is imported using the **`import csv`** statement
- After **`import csv`**, we can now call the `csv` module functions by appending them to `csv.` .
- `csv` module is included in when ArcGIS Pro is installed
- We *can* code this all ourselves using the code we just ran...but we don't want to (trust me!). We will use 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

See Python Documentation: CSV File Reading and Writing

```
In [ ]:  import csv

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

             # Check the file type prior to CSV module
             print(type(file_in))

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

             # Check the file type after CSV module
             print(type(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 moves to the next line of the csv file

             # Create an empty list to store the data...this is a common workflow in Python!
             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)
```

```
In [ ]:  # Print the headers and data
         # What are the data types?
         print(f'Headers: {headers}')

         print(f'Data: {data}')
```

```
In [ ]:  # Print each line in the CSV file
         for line in data:
             print(line)
```

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

Each row is a list, with each table cell value as an element within the list.

What do you notice about the data type of each value in the list? Compare it to reading the raw file approach

Let's do some quick sanity checking...

```
In [ ]:   # Does each row contain the same number of elements (i.e. table columns)
          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 extract the data from a CSV file. What I've shown here is one of the simplest ways. There are more advanced methods for more complex data.

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

V. Creating Scripts with Python

What is a Script?

- A script is multiple lines of code intended to be run directly stored in a script file.
- Can be ran independently (e.g., command prompt) or in another application that can run code (e.g., ArcGIS Pro, Python Notebooks, or Python IDE).
- In Python, the script file is a '.py' file or ran in a Python Notebook (or other similar file for other applications)
- In ArcGIS, we generally distinguish between standalone scripts and custom script tools...we will cover both!

Why Create Scripts?

1. Scripts inherently document workflows and data transformations in a consistent and standard way...No more wondering what you did a year ago to create that ALL-IMPORTANT dataset!!
2. Scripts can be designed to be repeatable and replicable for when the workflow may work on other files or datasets or when there are updates to data or files
3. Scripts encourage you to think critically and efficiently about your workflows...this makes you a better GIS professional and data scientist/manager!
4. Scripts save time and ROI in the long run! Not in the short run though...we'll talk about that later!

Planning a Script

- Very important step in the process of coding!
- Requires you to think critically about the workflow and how the script will be organized
- Always recommended to write pseudo code as a starting point

What is Pseudo Code?

- A plain-language explanation or outline of the script workflow
- Helps to organize and plan the script workflow
- Allows others reading the code to understand more clearly what is being done
- Can be written directly in the code using comments (i.e., '#')

Note: I am teaching you just one of many methodologies...

- You can modify this as much as you want
- Find a structure that works for you

- Keep it simple

Scenario:

> Many of your GIS workflows require creating an Excel file of feature classes and standalone tables as a final standalone file to share with your colleagues. The ArcGIS **Table to Excel** tool has been working great! But, you recently learned that your colleagues have been deleting unnecessary fields and exporting the Excel files to CSV and they would like to eliminate these steps from their workflow. Therefore, they have asked you to deliver files as CSV files without the unnecessary fields.

> Existing ArcGIS tools do allow for selecting a subset of fields to be exported. But, this exercies will help you understand the nature of data and how those tools are designed. We will create a custom standalone script to replicate this task.

This is a great example for when a simple standalone script can solve a simple workflow that needs to be completed often

General Components of a Standalone Script for ArcGIS

Let's review the template script below...

Note: I have provided template files for ArcGIS Notebooks and an ArcGIS Custom Script Tools

```python
# -*- coding: UTF-8 -*-

''' Metadata, Copyright, License:
-------------------------------------------------------------------
Name:       <FileName>.py
Purpose:    <Purpose>
Author:     <Name, Your>
Created:    <YYYY/MM/DD>
Copyright:  Copyright <YYYY> <Your Name or Organization>
License:    <License text>
-------------------------------------------------------------------
'''

''' Import Modules '''
# arcpy and other modules needed to complete your code

''' Functions '''
# Any custom reusable functions

''' Parameters '''
# Inputs and outputs that may change each time the code is run

''' Script '''

# Environments
    # Gets the house in order for how data and outputs will be dealt with
    # These may also be dispersed contextually throughout the script

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

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

Create Pseudo Code

[Top](#)

1. Create a copy of the ArcGIS Notebook Template

- Copy it to the **Notebooks** folder
- Name it **Convert Table to CSV.ipynb**

Let's review the ArcGIS Notebook Template

2. Update the Metadata, Copyright, License section as below

```python
# -*- coding: UTF-8 -*-

''' Metadata, Copyright, License:
-------------------------------------------------------------------
```

```
Name:       Convert Table to CSV.ipynb
Purpose:    This script converts a table to a CSV table with selected
            fields.
Author:     Whitacre, James
Created:    2025/04/16
Version:    0.0.1
Copyright:  Copyright <YYYY> <Your Name or Organization>
License:    Licensed under the Apache License, Version 2.0 (the
            "License"); you may not use this file except in compliance
            with the License. You may obtain a copy of the License at
            http://www.apache.org/licenses/LICENSE-2.0
            Unless required by applicable law or agreed to in writing,
            software distributed under the License is distributed on an
            "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
            either express or implied. See the License for the specific
            language governing permissions and limitations under the
            License.
--------------------------------------------------------------------------
'''
```

3. Record the steps needed to create the script (i.e., the pseudo code in plain language)

- Think like a computer (remember, coding is a new way of thinking, like a foreign language)
- Not all steps may be known at first...that is ok! This is an iterative process
- The original order you expected may change or need modified...that is ok too!
- Think about how the `csv` module read CSV files

Human-readable Pseudo Code:

1. Input a feature class or standalone table
2. Select the list of field names to be exported
3. Read the feature class or standalone table data
4. Create an output CSV file
5. Open the output CSV file
6. Write the field names to the new CSV file
7. Write the feature class or standalone table data to the new CSV file

4. Identify which pseudo code elements should be parameters

- *Hint: Which steps require user input or may be changed each time the code is ran? We will cover this in more detail soon!*

Add the pseudo code below to the in the `''' Parameters '''` and `''' Script '''` sections of the Notebook

```
In [ ]:  ''' Parameters '''
         # Input feature class or standalone table

         # Select the list of field names to be exported

         # Output CSV file path and name
```

```
In [ ]:  ''' Script '''
         # Read the feature class or standalone table data

         # Create and open the output CSV file

         # Write the field names to the new CSV file

         # Write the feature class or standalone table data to the new CSV file
```

5. What Python modules are needed?

- Are there already some Python modules available to help write this code more efficiently?

- This is dependent on the processes and you may not know right away that there is (which is ok)

- Remember, there are many more modules out there! We cannot go over all of them...

- `arcpy` is required if we are working with ArcGIS

- Reading or writing a CSV? Yes!...Then also want to import the `csv` module

Add the following `import` statements and parameter variables to your script in the `''' Import Modules '''` section

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

Develop the Script

Make it Work, Make it Pretty, Make it Fast

- When developing code, prioritize development in this order
- When starting out, shoot for making scripts work first!
- When it works, go back to ensure the code is readable and well documented (i.e., pretty!)
- If needed, try to make the code more efficient to work faster. This is not always needed...

1. Create input and output parameters

- Think about this like an ArcGIS tool...
- What variables will change every time the script is run?
- Typically these are paths or other values that may be dynamic
- What is the path to the feature class we want to read?
- If the script requires this from the user, it should be a parameter

Add the following parameter variables to your script in the `''' Parameters '''` section

Don't Forget to update paths! Use 'Copy Path' (ArcGIS Pro 3.x)...

- Which parameters are inputs vs. outputs?
- What are the data types of each variable?

```
In [ ]:  ''' Parameters '''

         # Input feature class or standalone table
         input_table = arcpy.management.MakeFeatureLayer('https://services2.arcgis.com/eQgAMgHr2CRobt2r/arcgis/rest/services/UnconventionalWells
                                                         'Unconventional Wells')

         # Identify list of field names to be exported
         field_names = ['Shape', 'PERMIT_NO', 'FARM_NAME', 'COUNTY', 'PROD_GAS_QUANT']

         # Output CSV file path and name
         output_csv = r'{Copy Home folder Path here...}\CSV\UnconventionalWells.csv'
```

2. Include any ArcGIS Environments and Variables

- These items may or may not be needed; as you develop your code, this will become more apparent based on what functions are needed/used
- Environments and Variables may also be better situated within the context of the code
- As you gain experience, you will have a better idea if/when you need to create these components

Because this is a simple script, we will not need to set any `# Environments` or `# Variables`; we will revisit this later

3. Develop the code

3 a. Read the table

- Read the feature class table or standalone table
- Because we are using ArcPy, reading the table can be accomplished using a **Search Cursor**

Note the added comments to the code for workflow clarity...this is best practice!!

Cursors

What are Cursors?

- Data access object that can be used either to iterate through the set of rows in a table or to insert new rows into a table

- Have three forms: Search, Insert, or Update
- Commonly used to read and update attributes.
- Work similarly to reading CSV files by using a `with` statement:

  `with arcpy.da.SearchCursor(input_table, field_list) as cursor:`
- Search cursors are read-only
- Two other types of cursors, **Insert** and **Update**, that allow writing (we won't work with these in this workshop)

Note: *There are two types of cursors in ArcPy, one directly in ArcPy (e.g. `arcpy.SearchCursor()`) and the other in the Data Access module within ArcPy (e.g. `arcpy.da.SearchCursor()`). The ArcPy Data Access module* version 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.**

See: ArcPy Documentation: Data Access using cursors

```
In [ ]:  ''' Script '''

         # Read the feature class or standalone table data

         # Create an empty list to append data to
         data = []

         # Create a search cursor to access the data
         with arcpy.da.SearchCursor(input_table, field_names) as cursor:
             for row in cursor:
                 # Append each row to the data list
                 data.append(row)
```

Print the first 10 rows to see how it looks

- Note that the output is a list of tuples, not lists...Why do you think this is?
- Also, there is a coordinate pair from the **Shape** field

```
In [ ]:  # Print first 10 rows
         print(data[:10])
```

3 b. Create and write to the CSV file

- In the Parameters section above, we just established the name of the CSV file
- Now, we need to create and write to the CSV file...we will use the `csv` module
- Then need to create a CSV writer object to:
  - Write the field names as the header to the CSV file
  - Write the data as separate lines to the CSV file

Again, note the additional comments in the code

Creating and Writing to CSV Files

Creating Files

- Writing files can be done using a `with open()` function

  - For Python 3

    - Use the `'w'` mode when opening the file, therefore:
      - The file is automatically created if it doesn't exist
      - **Or...**
      - The file is ***Overwritten*** if it already exists
    - Need to identify what creates a `newline` when writing to the file
  - For Python 2, use the `'wb'` write mode

File Access - Write Modes Only...

Modes	Description
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.

See

Writing to CSV Files

- `csv.writer(file object)` class to create a writing object

- `writerow(row)` CSV writer method to write a single row

- `writerows(list of lists)` CSV writer method to write many rows

In [ ]:
```python
''' Script '''

# Create and open the output CSV file
with open(output_csv, 'w', newline='') as csv_file:
    # Creates CSV Writer object
    csv_writer = csv.writer(csv_file)

    # Write the field names to the new CSV file
    csv_writer.writerow(field_names)

    # Write the feature class or standalone table data to the new CSV file
    csv_writer.writerows(data)

# Message that the script is finished
print("CSV file complete; located at {}".format(output_csv))
```

In [ ]:
```python
# Python 2 with statement; For some reason, 'w' adds an extra line between rows, use 'wb' instead...
with open(output_csv, 'wb') as csv_file:
```

4. Finalize and Clean up the Notebook

- Update the Notebook information
- Delete any unnecessary cells
- Comment out any unnecessary cells

5. Run the Notebook

- Open the **PA Unconventional Wells** map
  - ArcGIS Notebooks will add any data created to the active or last active map
- Try **Cell > Run All**
- Refresh the **CSV** folder to see the new file

Final Standalone Script Code:

In [ ]:
```python
# -*- coding: UTF-8 -*-

''' Metadata, Copyright, License:
-------------------------------------------------------------------
Name:       Convert_Table_to_CSV.ipynb
Purpose:    This script converts a table to a CSV table with selected
            fields.
Author:     Whitacre, James
Created:    2024/04/10
Version:    0.0.1
Copyright:  Copyright <YYYY> <Your Name or Organization>
License:    Licensed under the Apache License, Version 2.0 (the
            "License"); you may not use this file except in compliance
            with the License. You may obtain a copy of the License at
            http://www.apache.org/licenses/LICENSE-2.0
            Unless required by applicable law or agreed to in writing,
            software distributed under the License is distributed on an
            "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
            either express or implied. See the License for the specific
            language governing permissions and limitations under the
            License.
-------------------------------------------------------------------
'''

''' Import Modules '''
import arcpy
import csv


''' Parameters '''
```

```python
# Input feature class or standalone table
input_table = arcpy.management.MakeFeatureLayer('https://services2.arcgis.com/eQgAMgHr2CRobt2r/arcgis/rest/services/UnconventionalWells
                                                'Unconventional Wells')

# Identify list of field names to be exported
field_names = ['Shape', 'PERMIT_NO', 'FARM_NAME', 'COUNTY', 'PROD_GAS_QUANT']

# Output CSV file path and name
output_csv = r'{Copy Home folder Path here...}\CSV\UnconventionalWells.csv'


''' Script '''

# Read the feature class or standalone table data

# Create an empty list to append data to
data = []

# Create a search cursor to access the data
with arcpy.da.SearchCursor(input_table, field_names) as cursor:
    for row in cursor:
        # Append each row to the data list
        data.append(row)

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


# Create and open the output CSV file
with open(output_csv, 'w', newline='') as csv_file:
    # Creates CSV Writer object
    csv_writer = csv.writer(csv_file)

    # Write the field names to the new CSV file
    csv_writer.writerow(field_names)

    # Write the feature class or standalone table data to the new CSV file
    csv_writer.writerows(data)

# Message that the script is finished
print("CSV file complete; located at {}".format(output_csv))
```

Challenge:

How might you change the code to use fewer lines?

Hint: *Is there anywhere you might be able to use list comprehension?*

Break

---

10 - 15 minutes

```python
In [ ]:  # Create list of data in input_data
         data = [row for row in arcpy.da.SearchCursor(input_table, field_names)]

         # Print first 10 rows
         print(data[:10])
```

VI. Adding Geoprocessing Tools to Scripts

- So far we haven't worked with many geoprocessing tools...So let's work on a few!
- The trick is to first run the tools in ArcGIS Pro, then Copy Python Command
- This method will GREATLY help document workflows much easier!

New Scenario:

> A client wants to estimate the acreage of unconventional well pads in each county for the entire Commonwealth of Pennsylvania. They need the data as a CSV table.

Pseudo Code:

- Select Drilled and Producing Wells
- Buffer each well 'farm' at 100 m to estimate the well pad area
- Summarize acreage of all well pads in each county
- Export the summary statistics as a CSV table

Run the ArcGIS Pro Tools First!

- **This is where the rubber hits the road with problem solving in GIS...and where the GIS professional thrives!!!**
- **This will be a rough as we will go through the GIS problem solving process together!**

ArcGIS Pro Geoprocessing Options

- Set options for running geoprocessing tools and scripts
    - ☑ Allow geoprocessing tools to overwrite existing datasets
    - ☐ Remove layers that reference data overwritten by geoprocessing tools
    - ☑ Add output datasets to an open map
    - ☑ Display disabled parameters
    - ☐ Enable Undo toggled on by default
    - ☑ Display data paths as shortened names
    - ☑ Analyze script and model tools for ArcGIS Pro compatibility
    - ☐ Open messages window automatically after running a tool

Possible Geoprocessing Tools to Run

1. Select *OR* Select Layer By Attribute

2. Buffer *OR* Pairwise Buffer

    - Method: Planar or Geodesic
    - Dissolve Type: Which option? What field?
3. Summary Statistics *OR* Summarize Within *OR* Calculate Geometry Attributes

4. Convert Table to CSV *OR* Export Table

Copy Code Below

```python
In [ ]:  # Select Drilled and Producing Wells


         # Buffer each well 'farm' at 100 m to estimate the well pad area


         # Summarize acreage of all well pads in each county


         # Export the summary statistics as a CSV table
```

Answer (Don't Look!!)

```python
In [ ]:  # Select Drilled and Producing Wells
         arcpy.management.SelectLayerByAttribute(
             in_layer_or_view="Unconventional Wells",
             selection_type="NEW_SELECTION",
             where_clause="WELL_STAGE IN ('Drilled', 'Producing')",
             invert_where_clause=None
         )

         # Buffer each well 'farm' at 100 m to estimate the well pad area
         arcpy.analysis.Buffer(
             in_features="Unconventional Wells",
             out_feature_class=r"{Copy Home folder Path here...}\PythonForArcGIS.gdb\UnconventionalWells_Buffer_100m",
             buffer_distance_or_field="100 Meters",
             line_side="FULL",
             line_end_type="ROUND",
```

```python
    dissolve_option="LIST",
    dissolve_field="COUNTY",
    method="GEODESIC"
)

# Calculate acreage of all well pads in each county
arcpy.management.CalculateGeometryAttributes(
    in_features="UnconventionalWells_Buffer_100m",
    geometry_property="Area_ac AREA_GEODESIC",
    length_unit="",
    area_unit="ACRES_US",
    coordinate_system='PROJCS["WGS_1984_Web_Mercator_Auxiliary_Sphere",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",637
    coordinate_format="SAME_AS_INPUT"
)

# Export the summary statistics as a CSV table
''' Import Modules '''
import csv


''' Parameters '''

# Input feature class or standalone table
input_table = "UnconventionalWells_Buffer_100m"

# Identify list of field names to be exported
field_names = ['COUNTY', 'Area_ac']

# Output CSV file path and name
output_csv = r'{Copy Home folder Path here...}\CSV\Unconventional_Wells_Pad_Area_County.csv'


''' Script '''

# Read the feature class or standalone table data

# Create an empty list to append data to
data = []

# Create a search cursor to access the data
with arcpy.da.SearchCursor(input_table, field_names) as cursor:
    for row in cursor:
        # Append each row to the data list
        data.append(row)

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


# Create and open the output CSV file
with open(output_csv, 'w', newline='') as csv_file:
    # Creates CSV Writer object
    csv_writer = csv.writer(csv_file)

    # Write the field names to the new CSV file
    csv_writer.writerow(field_names)

    # Write the feature class or standalone table data to the new CSV file
    csv_writer.writerows(data)

# Message that the script is finished
print("CSV file complete; located at {}".format(output_csv))
```

VII. Creating Custom ArcGIS Script Tools

Now, let's go back to the Convert Table to CSV script...

Scenario Update:

> Now you have a new intern starting and this standalone script is now getting used for many data requests, so it is getting annoying
> having to manually open the script in the ArcGIS Notebook, update the variables, and then run the code. You think, "Man it would be nice
> to have this as a tool in ArcGIS Pro...why hasn't Esri created this tool?"

What is an ArcGIS Script Tool?

An ArcGIS Script Tool enables turning Python scripts and functionality into custom geoprocessing tools that look and act like the tools included with ArcGIS Pro. A script tool provides many advantages:

- Custom script tools are an integral part of geoprocessing
  - Act like any other ArcGIS tool that you can open from the Catalog pane
  - Can be used in ModelBuilder, the Python Window, an ArcGIS Notebook, and standalone scripts
- Enables writing messages and errors to the Geoprocessing History and tool dialog box
- Can add parameter validation to ensure inputs are correct
- Uses built-in documentation tools that can be customized
- When run as a script tool in ArcGIS Pro, `arcpy` is fully aware
  - Settings made in the application, such as `arcpy.env.overwriteOutput` and `arcpy.env.scratchWorkspace`, are available from `arcpy` in your script tool.
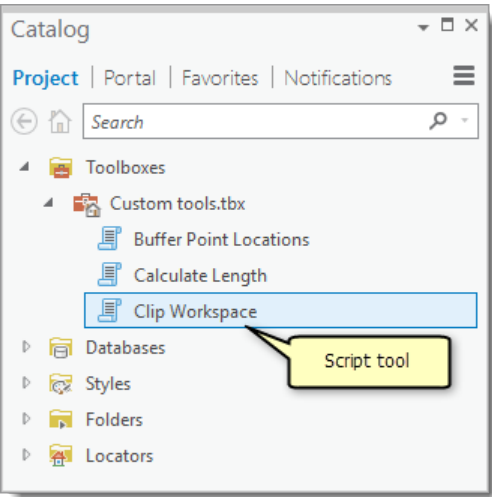
Comparing Python Script Tools, Standalone Scripts, and ModelBuilder

Python Script Tool	Python Standalone Script	ModelBuilder
Textual programming language	Textual programming language	Visual Programming language
Easy to learn with very flexible structure	Easy to learn with very flexible structure	Relatively easier to learn for GIS beginners, but restrictive structure
Lower-level geoprocessing tasks (e.g. cursors, loops)	Lower-level geoprocessing tasks (e.g. cursors, loops)	Lower-level geoprocessing tasks may not be possible
More advanced error handling	More advanced error handling	Errors handled by the tools in the model
Can use other Python modules and wrap other software (e.g. R)	Can use other Python modules and wrap other software (e.g. R)	Restricted by ArcToolbox tools
Uses the ArcGIS Tool dialog	Must use an ArcGIS Notebook, Python Window, IDE, or Command Line	Uses the ArcGIS Tool dialog

Create a Script Tool

Scenario Update

> Ok, so by now hopefully the light bulb should have gone off, and you should be thinking, "Oh cool, I can make my own ArcGIS Script Tool!" But, where should you start when creating a script tool? I am so glad you asked! First, we need to modify the standalone script to work in a script tool.

To create a script tool in a custom toolbox, we need a few things (luckily we already have a few!):

- ☑ A project toolbox (created when you create a new ArcGIS Pro Project!)
- ☐ A script tool
- ☑ A script
- ☐ A precise definition of the parameters of your script

See **ArcPy Documentation: A quick tour of creating tools with Python**

1. Create a new Script Tool in the project Toolbox

- Follow the demonstration to create a new Script Tool

  - See **ArcPy Documentation: Add a Script Tool** for detailed instructions
- **General Properties**

- Name: **ConvertTableToCSV**
- Label: **Convert Table to CSV**
- Description: **This script tool converts a table to a CSV table with selected fields.**
- Summary: **This script tool converts a table to a CSV table with selected fields.**

ArcGIS Toolboxes

There are three toolbox formats in ArcGIS Pro:

- **ArcGIS toolbox** — This toolbox format is stored as an .atbx file on disk. This toolbox format stores tools, scripts, and models. It provides better cross-release compatibility and persistence, improved performance and scalability, and less possibility of file corruption due to binary storage, as this format is JSON based with an open specification compared to the legacy toolbox format (.tbx) described below. You can create toolsets, add new model and script tools, change properties, and copy and paste tools from legacy toolboxes into the new toolbox format using the Catalog pane.
- **Legacy toolbox** — This toolbox format is stored as a .tbx file on disk or in a database.
- **Python toolbox** — This toolbox format is stored as a .pyt file on disk. All properties of a Python toolbox and its tools are defined using Python code.

Starting with ArcGIS Pro 3.0, you can't create legacy toolboxes (.tbx) in a project. However, you can still add existing legacy toolboxes to a project and edit them.

2. Setup parameters for the Script Tool

- **Parameter Options**
    - **Label**: This how the parameter will be labeled in the tool dialog
    - **Name**: Note that it is the same as the script parameter variable...more on this in a bit!
    - **Data Type**: Note the many different types!
    - **Type**: Required vs. Optional vs. Derived
    - **Direction**: Is the parameter an input or an output?
    - **Category**: These are user-defined categories to help organize parameters on the tool dialog...we won't use it here
    - **Filter**: Certain Data Types can be filtered (e.g., polygon feature layer only)
    - **Dependency**: Certain Data Types can have dependencies (e.g., Fields from a table can be loaded into the parameter list)
    - **Default**: Can be used if there is a common value usually used...we won't use it here
    - **Environment**: Can set the default value for a parameter to the value of an environment setting
    - **Symbology**: If output is a feature set, raster, TIN, or layer, specify the location of a layer file (.lyrx); When tool is run and output is added to display, it will draw using the symbology in the layer file
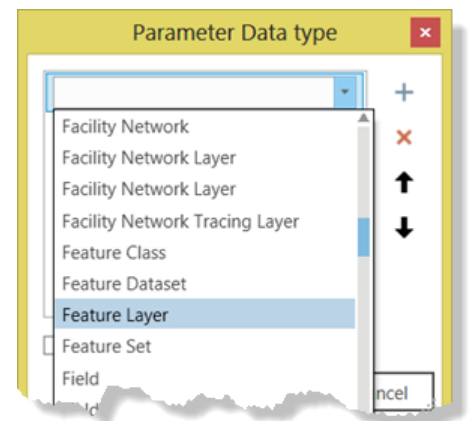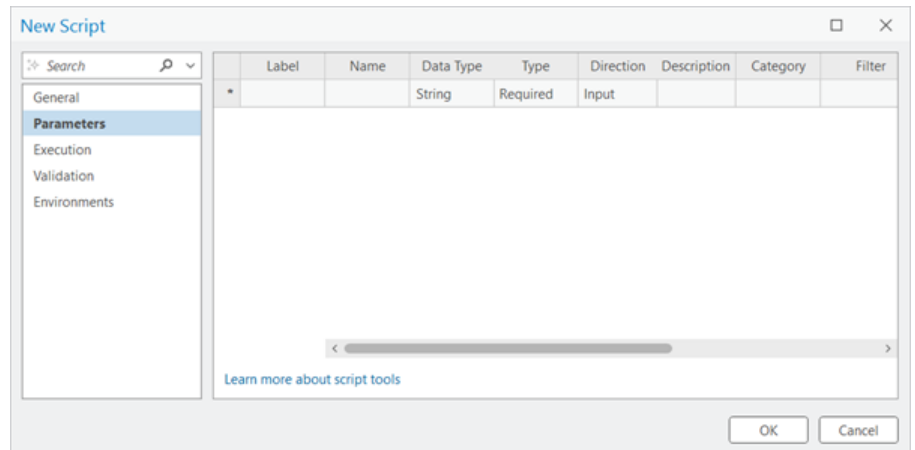
See **ArcPy Documentation: Setting script tool parameters**

- **Parameters**
    - Set the parameters as described in the table below
    - **Bold** indicates those that need edited or that are not default

Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default	Environment	Symbology
Input Table	**input_table**	Table View	Required	Input						
Field Names	**field_names**	**[Field]** **[✓] Multiple values**	Required	Input		**Field** **[✓] All except Blob, Raster, XML**	**input_table**			
Output CSV Table	**output_csv**	File	Required	**Output**		**File** **csv;txt**				

- **Execution**
    - We'll come back to this!

- **Validation**
  - We'll come back to this!
- **Environments**
  - We might come back to this...

Prepare Script for ArcGIS Script Tool

1. Create a copy of the ArcGIS Script Tool Template Python script file

- Copy **ArcGIS_Script_Tool_Template.py** to the **Scripts** folder
- Name it **Convert_Table_to_CSV.py**
- Open the script file in a code editor

Challenge:

Can you figure out how to do this with Python?

Review the ArcGIS Script Tool Template

- Note that the ArcGIS Notebook Template is pretty much the same
- Note that the script created in the ArcGIS Notebook is already implementing most of the template elements
- Also note that the template conforms to the **PEP 8 -- Style Guide for Python Code**

```python
In [ ]: # -*- coding: UTF-8 -*-

# Follow the Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/
# Replace sections below enclosed by brackets "<>" with appropriate identifying information, but do not include the brackets.
# Delete lines 3-6

''' Metadata, Copyright, License:
----------------------------------------------------------------------
Name:       <FileName>.py
Purpose:    <Purpose>
Usage:      <Usage>
Author:     <Name, Your>
Source:     <Web link, Author, Acknowledgments - Optional>
Created:    <YYYY/MM/DD>
Modified:   <YYYY/MM/DD>
Version:    #.#.#
Copyright:  Copyright <YYYY> <Your Name or Organization>
License:    Licensed under the Apache License, Version 2.0 (the
            "License"); you may not use this file except in compliance
            with the License. You may obtain a copy of the License at
            http://www.apache.org/licenses/LICENSE-2.0
            Unless required by applicable law or agreed to in writing,
            software distributed under the License is distributed on an
            "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
            either express or implied. See the License for the specific
            language governing permissions and limitations under the
            License.
----------------------------------------------------------------------
'''

''' Import Modules '''
import arcpy


''' Functions '''
def function_name(var_1, var_2):
    """Docstring...Style Guide: https://www.python.org/dev/peps/pep-0257/

    Parameters:
    var_1: data type
        Description...

    var_2: data type
        Description...
    """
    # Function Code

    return
```

```
''' Parameters '''
# Param Name (Param Type; Param Notes)
param_0 = arcpy.GetParameterAsText(0)
# Param Name (Param Type; Param Notes)
param_1 = arcpy.GetParameterAsText(1)


''' Script '''

# Environments

# Variables

# Processes
```

2. Copy script from ArcGIS Notebook and modify script parameters

- In order for a script tool to input parameters into the tool, we need to use the `arcpy.GetParameterAsText()` or `arcpy.GetParameter()` functions
    - Note the number in the parentheses; this indicates the index location of parameter in the Script Tool
- It is also best practice to make comments on how the parameters should be set up in the Script Tool so someone can recreate the parameters from the script itself
- This will make a bit more sense when we create the Script Tool in an ArcGIS Toolbox

```
''' Parameters '''

''' Update Code...'''

# Input Table (Table View)
input_table = arcpy.GetParameterAsText(0)

# Output Fields (Field, Multiple values: Yes; Filter: Field [NOT Shape, Blob, Raster, XML]; Dependency: input_table)
field_names = arcpy.GetParameterAsText(1)

# Output CSV Table (File; Output; Filter: File [csv;txt])
output_csv = arcpy.GetParameterAsText(2)

''' End Update Code...'''
```

3. Add ArcPy elements to expand the script tool functionality

- **Progressors**

    - Show messages in the Geoprocessing pane progress bar while a tool is running
    - Good for passing messages about upcoming tasks or information in the script
    - Can help see how far along a specific task is to completion
- **Messages, Warnings, and Errors**

    - Essentially acts like the Python `print()` function, but in the Details of script
    - Good for passing messages about optional tasks that completed, are anomalies/uncommon occurrences, or any other important message useful for being stored in the Geoprocessing History
    - Warnings can relay warning messages and you will see the warning icon: ⚠️
    - Errors can be created to stop the script from running any further and relay an error messages and you will see the error icon: ❌

Add some Progressors and a Message to the Script

1. Add a step progressor to track writing the rows

2. Reset the progressor and add a default progressor to notify that the CSV file is being created

3. Add a message to let users know that the script completed and how many rows and field were written

```
''' Script '''

''' New Code...'''

# Set the progressor, first count the number of records
rows = int(arcpy.managemene.GetCount(input_table)[0])
arcpy.SetProgressor('step', '{0} rows in dataset...'.format(rows), 0, rows, 1)

''' End New Code'''
```

```python
# Read the feature class or standalone table data
# Create an empty list to append data to
data = []

# Create a search cursor to access the data
with arcpy.da.SearchCursor(input_table, field_names) as cursor:
    for row in cursor:
        # Append each row to the data list
        data.append(row)
        # Update the progressor position
        arcpy.SetProgressorPosition()


''' New Code...'''

# Reset and create new progressor to show that CSV file is being created
arcpy.ResetProgressor()
arcpy.SetProgressor('default', 'Creating CSV file...')

''' End New Code'''


# Open the new output CSV file
with open(output_csv, 'w', newline='') as csv_file:
    # Creates CSV Writer object
    csv_writer = csv.writer(csv_file)

    # Write the field names to the new CSV file
    csv_writer.writerow(field_names)

    # Write the feature class or standalone table data to the new CSV file
    csv_writer.writerows(data)


# Message that the script is finished
arcpy.AddMessage('CSV file complete: {0} rows and {1} fields exported.'.format(rows, len(field_names)))
```

4. Add the script to the Script Tool

- Follow the demonstration to add the Python script file
    - **See ArcPy Documentation: Add a Script Tool for detailed instructions**

5. Test the Script Tool

- Did it work?
- If you got an error, what kind of error is it? Can you figure out where the error is in the code?

Hint: Click View Details

Handling Errors and Debugging

Three types of errors in Python

- **Syntax Errors** - prevents code from running

- **Exceptions** - code will stop running mid-process

- **Logic Errors** - code will run, but produces undesired results

What is debugging?

- Methodological process for finding errors in the script

- Debugging methods don't usually tell you why, but rather where the issue is occurring

- Examples include:

    - Carefully reviewing error messages (we just did this!)
    - Adding `arcpy.AddMessage()` to track the process and pinpoint where the error is (`print()` statements will work in standalone scripts)
    - Selectively commenting out code while testing

- Using a Python IDE debugger (we won't cover this...sorry)

Add some `arcpy.AddMessage()` statements and Comment out some code

- Recall the line number from the error...what might we want to look into?

- Comment out the entire Script section

- In the script, add a descriptive `arcpy.AddMessage()` statement after the Parameters section, but before the Script section

  - Cycle through each parameter variable
  - Save the script in the IDE and re-run the script tool after any changes
- Where do you notice any anomalies?

  - ***HINT***: Recall the review of cursors...

Note: For future reference, if a `arcpy.AddMessage()` statement does not execute, then that is the process to check for issues/errors

```python
In [ ]:  ''' Parameters '''
         # Input Table (Table View)
         input_table = arcpy.GetParameterAsText(0)

         # Output Fields (Field; MultiValue: Yes; Filter: field [NOT Shape, Blob, Raster, XML])
         field_names = arcpy.GetParameterAsText(1)

         # Output CSV Table (File; Direction: Output; Filter: File [csv, txt])
         output_csv = arcpy.GetParameterAsText(2)


         ''' Add Error Handling Code...'''

         # Add a message to test the parameter input
         arcpy.AddMessage(table) # field_names output_csv

         ''' End Error Handling Code...'''


         ''' Script '''
         # Comment out everything below here...
```

Fix the Error

- For the problematic parameter add:

`arcpy.GetParameterAsText(#).split(';')`

- Re-run the script tool to see how the parameter changes

- Un-comment the Script section, Save the script, and re-run to see if there are other errors

  - **Any other errors? Use your problem solving skills to debug!!**

Handling 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

Handling 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

```python
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.

Tool Parameter Validation

Scenario Update:

> Some of your geodatabase feature classes and tables have field aliases that are much easier for the users of the CSV files to understand. So, you would like to add an option to output the aliases as either the header or an additional row of the CSV table.

> However, sometimes tables have field aliases that are the same as the field names. So, you would like to only have the new alias parameter show up if the fields aliases are different than the field names.

This requires Script Tool Validation

Adding this code is a bit more complicated...I will do my best to explain!

Validation

- Provides custom behavior for the script tool dialog box, such as enabling and disabling parameters, providing default values, or modifying filter lists.
- Helps validate data inputs and outputs and check for errors BEFORE running the script
- Some validation is managed by ArcGIS, while some needs to be coded

See ArcPy Documentation: Customizing script tool behavior

1. Add a new parameter to the Script Tool

- Add the parameter code below after the last parameter in the script

```
In [ ]:
''' Parameters '''

'''Add New Parameter...'''
# Add Field Aliases to CSV Table (String; Type: Optional; Filter:  Value List [NONE, AS_HEADER, AS_ROW]; Default: NONE)
alias_incl = arcpy.GetParameterAsText(3)
```

2. Modify the Script Tool code as below

- Add the code snippets below into the script as shown

```
In [ ]:
''' Script '''

# Set the progressor, first count the number of records
rows = int(arcpy.managemene.GetCount(input_table)[0])
arcpy.SetProgressor('step', '{0} rows in dataset...'.format(rows), 0, rows, 1)


# Read the feature class or standalone table data
# Create an empty list to append data to
data = []

# Create a search cursor to access the data
with arcpy.da.SearchCursor(input_table, field_names) as cursor:
    for row in cursor:
        # Append each row to the data list
        data.append(row)
        # Update the progressor position
        arcpy.SetProgressorPosition()

''' New Code...'''

# Create a describe object of the input table
desc = arcpy.Describe(input_table)

# If aliases are included: Create a list of all of the field aliases in the table
if alias_incl != 'NONE':
    # Create a list of all of the fields in the table
    aliases = [field.aliasName for field in desc.fields if field.name in field_names]

''' End New Code'''


# Reset and create new progressor to show that CSV file is being created
arcpy.ResetProgressor()
arcpy.SetProgressor('default', 'Creating CSV file...')
```

```
    # Open the new output CSV file
    with open(output_csv, 'w', newline='') as csv_file:
        # Creates CSV Writer object
        csv_writer = csv.writer(csv_file)

        ''' Modify Code '''

        # Write aliases or fields names as the header to the output CSV file
        if alias_incl == 'AS_HEADER':
            csv_writer.writerow(aliases)
            arcpy.AddMessage('Aliases written as the header...')
        else:
            csv_writer.writerow(field_names)

        # Write aliases as a row to output CSV file if 'AS_ROW' is selected
        if alias_incl == 'AS_ROW':
            csv_writer.writerow(aliases)
            arcpy.AddMessage('Aliases written as a row...')

        ''' End Modify Code '''

        # Write the feature class or standalone table data to the new CSV file
        csv_writer.writerows(data)


    # Message that the script is finished
    arcpy.AddMessage('CSV file complete: {0} rows and {1} fields exported.'.format(rows, len(field_names)))
```

3. Add validation to the Script Tool

- Review the **Validation** code below
- Open the **Convert Table to CSV** script tool **Properties**
- Click **Validation**
- Copy the **Validation** code below

In [ ]:
```
class ToolValidator:
    # Class to add custom behavior and properties to the tool and tool parameters.

    def __init__(self):
        # set self.params for use in other function
        self.params = arcpy.GetParameterInfo()

    def initializeParameters(self):
        # Customize parameter properties.
        # This gets called when the tool is opened.

        # Disable the optional alias parameter
        self.params[3].enabled = False

        return

    def updateParameters(self):
        # Modify parameter values and properties.
        # This gets called each time a parameter is modified, before
        # standard validation.

        # Check if the parameters have a value and have been validated
        if self.params[0].value and not self.params[0].hasBeenValidated:
            try:
                # Create a describe object for 'Input Table'
                desc = arcpy.Describe(self.params[0].value)

                # Check if 'Input Table' contains aliases, and if so enable Add Field Aliases to CSV Table (optional)' parameter
                aliases = [field.aliasName for field in desc.fields if field.aliasName != field.name]
                if aliases:
                    self.params[3].enabled = True
                else:
                    self.params[3].enabled = False

            except:
                pass

        return

    def updateMessages(self):
        # Customize messages for the parameters.
        # This gets called after standard validation.
        return
```

```
# def isLicensed(self):
#     # set tool isLicensed.
# return True

# def postExecute(self):
#     # This method takes place after outputs are processed and
#     # added to the display.
# return
```

4. Test the Script Tool

- Test the Script Tool on the **Illinois Counties** layer in the **Illinois** map
  - Did it work?
  - If you got an error, what kind of error is it?
  - Is the error in the code? If not, can you figure out where the error is?

Hint: Check Field Names parameter 'Field list settings' on the Tool Dialog in the Geoprocessing Pane

---

Documenting Script Tools

Top

- Right click on the Script Tool

- Click Edit Metadata

  - Borrow and steal from Esri's script tools
    - Good example: Table To Excel
  - Be very descriptive, but concise
- Save metadata edits

- Open the script tool and view the help

VIII. Conclusion and Moving Forward

Top

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!')
```