Python for ArcGIS

Description

Programming tools are now a standard feature within GIS software packages and allow GIS users to automate, speed up, and become more robust in their data management and analytic work. This workshop is designed for GIS users who have some experience with Python programming, but who want to take their Python skills to the next level in ArcGIS. The workshop will focus on the ArcPy Python site package in ArcGIS with hands-on exercises designed to provide the essential skills to use Python in ArcGIS for effectively. Participants will learn to use ArcPy tools, build standalone geoprocessing scripts, and cover how to create custom script tools in ArcGIS toolboxes for reuse and sharing. Participants will also develop skills to explore more resources and options for utilizing Python in ArcGIS.

NOTE: Basic Python programming will not be covered in this workshop, so participants are expected to know some basic Python concepts such as syntax, variables, data types (strings, numbers, Booleans, lists, tuples, dictionaries, etc.) conditional statements, functions, and loops.

Specific Topics Include:

- Work with the ArcPy Python site package for ArcGIS
- Explore the difference between Python scripts, Notebooks, and ModelBuilder models
- Build and share custom ArcGIS script tools for automation
- Learn tips and tricks for validating script syntax and error handling

Instructor

James Whitacre

GIS Research Scientist, Carnegie Museum of Natural History, Powdermill Nature Reserve

whitacrej@canregiemnh.org

James Whitacre is the GIS Research Scientist for the Carnegie Museum of Natural History where he manages the GIS Lab at Powdermill Nature Reserve, the Museum’s environmental research center, and supports museum staff and affiliated researchers with geospatial technologies and needs. This is Whitacre’s second appointment at the Museum as he was formerly the GIS Manager from 2011 to 2014. 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.

Outline

I. Data and Software Setup

Computing and Software Needs

1. ArcGIS Pro 2.5.x+ or ArcGIS Desktop 10.4.x+ (Standard or Advanced preferred)

- **ArcGIS Notebooks (https://pro.arcgis.com/en/pro-app/arcpy/get-started/pro-notebooks.htm)** will be used for practicing writing code
  - Included with ArcGIS Pro 2.5+
  - Optimal for ArcGIS and Python 3
  - Easily run code directly in ArcGIS Pro

2. Python Code Editor or IDE

The workshop is designed to use any script editor or integrated development environment (IDE) and can be completed using any combination of the software included in the list below. The items in bold are the recommended applications for the workshop in order.

- **Jupyter Notebook (https://jupyter.org/)**
  - Installed with ArcGIS Pro 2.2+
  - Optimal for Python 3
  - Easily run code in the application

- **Visual Studio Code (https://code.visualstudio.com/)**
  - Great all purpose code and text editor...an essential application for coding in numerous languages!
  - See Getting Started with Python in VS Code (https://code.visualstudio.com/docs/python/python-tutorial) for Python setup
  - Requires some application savvy-ness to unleash full potential
  - Not very easy to run code in the application

- **Notepad++ (https://notepad-plus-plus.org/)**
  - Great all purpose text editor
  - Highlights Python syntax so can be used to develop Python scripts
  - Not very easy to run code in the application

- IDLE (https://docs.python.org/2/library/idle.html)
  - Installed with ArcGIS Desktop
  - Can be used to run and edit scripts
  - A little clunky for Python development...but is a standard software worth knowing

- PyScripter (http://sourceforge.net/projects/pyscripter/files/)
  - Good for Python 2 script development for ArcMap
  - Can be used to run and edit scripts
  - Download **v. 3.6.1 32-bit** version for ArcMap without 64-bit Background Geoprocessing
  - Download **v. 3.6.1 64-bit** version for ArcMap with 64-bit Background Geoprocessing
  - *NOTE: The zip files contain portable versions of PyScripter. No installation is needed. Just unzip the archive and start using PyScripter.*

- Spyder (https://www.spyder-ide.org/)
  - Better for scientific Python 3 and ArcGIS Pro development
  - Requires special installaiton through ArcGIS Pro and/or Anaconda that may be tricky
  - Easily run code in the application, but may require some setup

- Pycharm (https://www.jetbrains.com/pycharm/)
  - Good for all around Python 3 development
  - Requires a normal installation from a free download
  - Easily run code in the application, but may require some setup

- There are many other script editors and IDEs! Always good to experiment with others

Download Exercise Data

From GitHub

- Go to repo at **https://github.com/whitacrej/Python-for-ArcGIS-ILGISA-2020 (https://github.com/whitacrej/Python-for-ArcGIS-ILGISA-2020)**
- Click on the **Code** dropdown
- Click **Download Zip**
- **Extract** zip file to desktop or well-known folder

From Pathable

- Go to **https://ilgisa.pathable.co/meetings/virtual/GgicDzbmPNjyrJKxz (https://ilgisa.pathable.co/meetings/virtual/GgicDzbmPNjyrJKxz)**
- Click on **Files**
- Click on **Python for ArcGIS ILGISA 2020 GitHub Repo**
- **Extract** zip file to desktop or well-known folder

Opening and Accessing Exercise Data and Scripts

- All of the GIS data and ArcGIS Notebooks can be opened in ArcGIS Pro 2.5+
- Python Scripts will need to be opened in a script editor or IDE

Note about ArcMap

- ArcMap should be usable for all exercises, however it has not been tested
- Jupyter Notebooks cannot be used with ArcMap; the ArcMap Python Window and script editors will need to be used to write and test code

ArcGIS vs. ArcMap vs. ArcGIS Pro

- ArcGIS, ArcGIS Desktop, or Desktop = Both ArcMap AND ArcGIS Pro
- ArcMap = ArcMap
- ArcGIS Pro or Pro = ArcGIS Pro

IDE: Jupyter Notebook

Jupyter

- **Actually, it is Project Jupyter (https://jupyter.org/)**

> Project Jupyter is a non-profit, open-source project, born out of the IPython Project (https://ipython.org/) in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license (https://opensource.org/licenses/BSD-3-Clause).

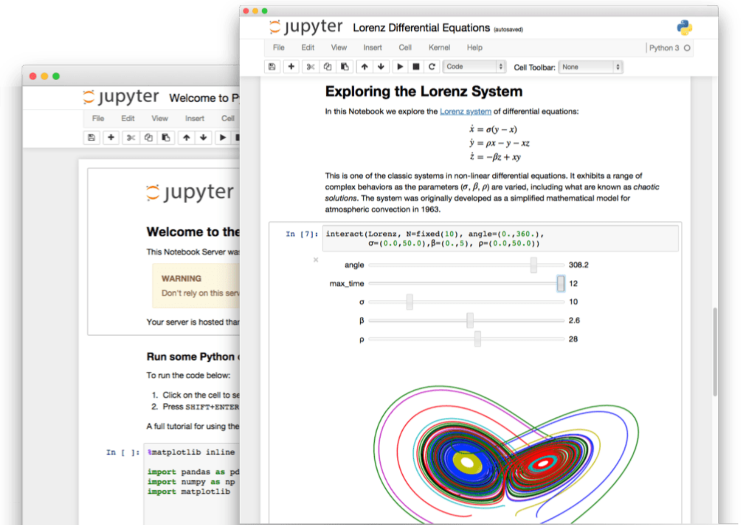
- **IPython (https://ipython.org/) is...**

IPython Interactive Computing

> IPython provides a rich architecture for interactive computing with:
>
> - A powerful interactive shell.
> - A kernel for Jupyter.
> - Support for interactive data visualization and use of GUI toolkits.
> - Flexible, embeddable interpreters to load into your own projects.
> - Easy to use, high performance tools for parallel computing.

- **So, it has it roots in Python**
- **But can be used with other programming languages**

Jupyter Notebook


jupyter Welcome to P
File Edit View Insert Cell
jupyter
jupyter Lorenz Differential Equations (autosaved)
File Edit View Insert Cell Kernel Help
Python 3
Code
Cell Toolbar: None
angle 308.2
max_time 12
σ 10
β 2.6
ρ 28

Welcome to the

This Notebook Server was

WARNING

Don't rely on this serv

Your server is hosted thar

Run some Python

To run the code below:

1. Click on the cell to se
2. Press SHIFT+ENTER

A full tutorial for using the

```
In [ ]: %matplotlib inline

        import pandas as pd
        import numpy as np
        import matplotlib
```

Exploring the Lorenz System

In this Notebook we explore the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = -\beta z + xy$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters (σ, β, ρ) are varied, including what are known as *chaotic solutions*. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: interact(Lorenz, N=fixed(10), angle=(0.,360.),
                 σ=(0.0,50.0),β=(0.,5), ρ=(0.0,50.0))
```

- **Web-based application (i.e. browser-based) to help capture entire computational workflows and processes**
  - Development
  - Documentation
  - Code execution
  - Visualizing and communicatong results
- **Main Features**
  - Acts as an interactive IDE, including syntax highlighting, indentation, code completion
  - Executes code directly in the browser environment and maintains results until cleared or when code is run again
  - Utilizes Markdown markup language to provide additional code commentary or for presentations

For further resources:

Project Jupyter: https://jupyter.org/index.html (https://jupyter.org/index.html)

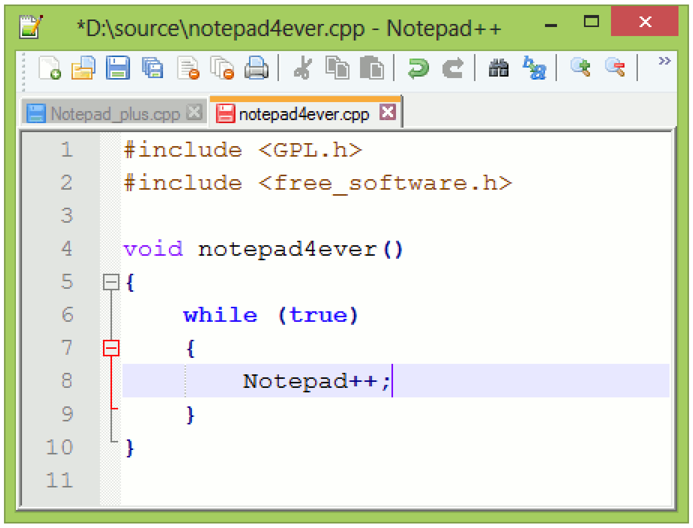
Jupyter Notebook Documentation: https://jupyter-notebook.readthedocs.io/en/stable/index.html (https://jupyter-notebook.readthedocs.io/en/stable/index.html)

Starting Jupyter Notebook

- Go to **Start Menu > All Programs > ArcGIS > Jupyter Notebook**
  - Wait for the Jupyter Notebook window to run and the dashboard to open in your default browser
- Navigate to the folder where you extracted the **GitHub download**
- Open the **Introduction to Python for ArcGIS.ipynb** Jupyter Notebook

Code Editor: Notepad++

- Free (as in "free speech" and also as in "free beer") source code editor and Notepad
- Available for Windows
- 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).



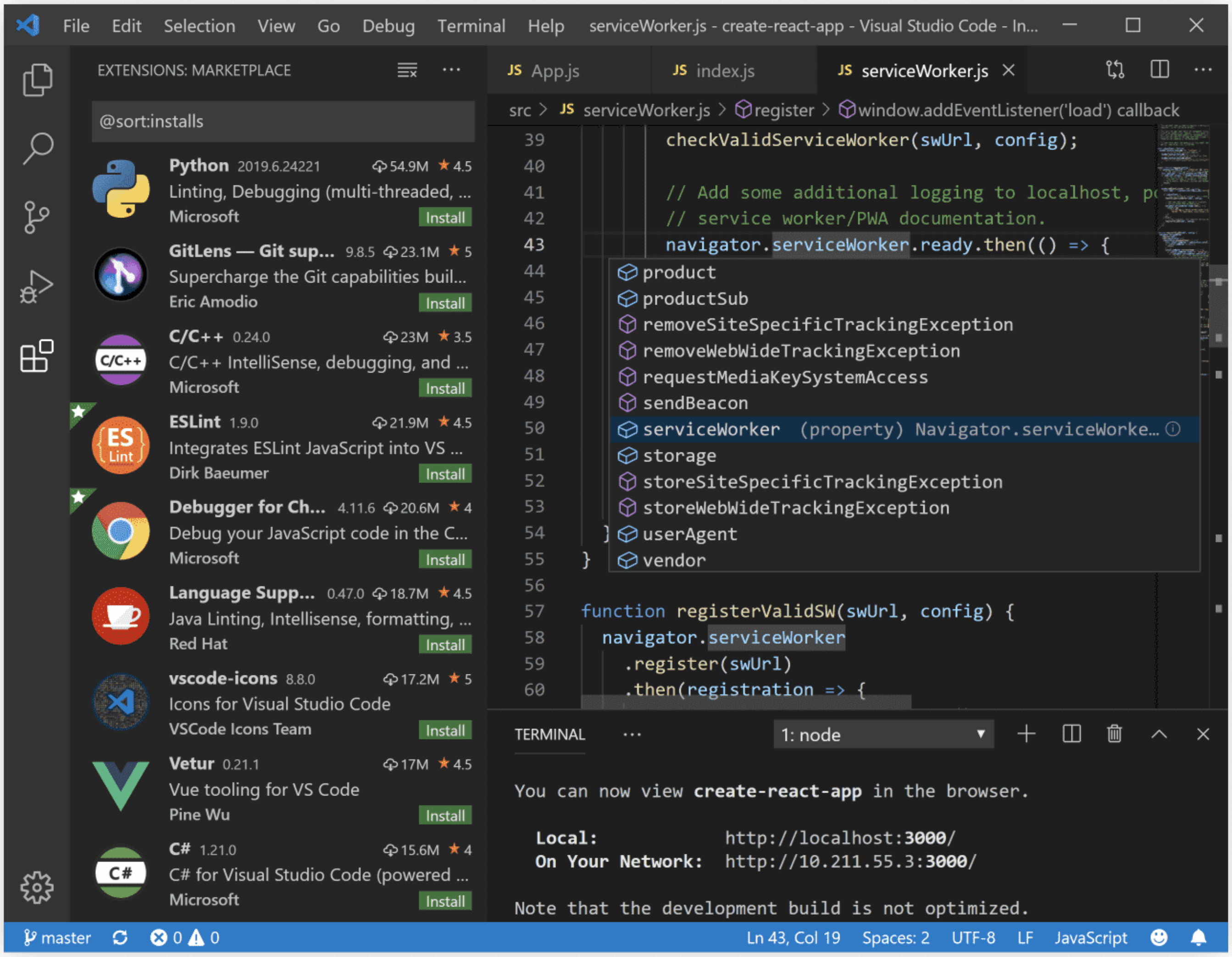
```
#include <GPL.h>
#include <free_software.h>

void notepad4ever()
{
    while (true)
    {
        Notepad++;
    }
}

```

Code Editor: Visual Studio Code (or VS Code for short)

- 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).


File Edit Selection View Go Debug Terminal Help serviceWorker.js - create-react-app - Visual Studio Code - In...
EXTENSIONS: MARKETPLACE
@sort:installs
JS App.js
JS index.js
JS serviceWorker.js
src > JS serviceWorker.js > register > window.addEventListener('load') callback
TERMINAL
1: node
master
0 0
Ln 43, Col 19 Spaces: 2 UTF-8 LF JavaScript

Python 2019.6.24221 54.9M ★4.5
Linting, Debugging (multi-threaded, ...
Microsoft Install

GitLens — Git sup... 9.8.5 23.1M ★5
Supercharge the Git capabilities buil...
Eric Amodio Install

C/C++ 0.24.0 23M ★3.5
C/C++ IntelliSense, debugging, and ...
Microsoft Install

ESLint 1.9.0 21.9M ★4.5
Integrates ESLint JavaScript into VS ...
Dirk Baeumer Install

Debugger for Ch... 4.11.6 20.6M ★4
Debug your JavaScript code in the C...
Microsoft Install

Language Supp... 0.47.0 18.7M ★4.5
Java Linting, Intellisense, formatting, ...
Red Hat Install

vscode-icons 8.8.0 17.2M ★5
Icons for Visual Studio Code
VSCode Icons Team Install

Vetur 0.21.1 17M ★4.5
Vue tooling for VS Code
Pine Wu Install

C# 1.21.0 15.6M ★4
C# for Visual Studio Code (powered ...
Microsoft Install

```
            checkValidServiceWorker(swUrl, config);

            // Add some additional logging to localhost, p
            // service worker/PWA documentation.
            navigator.serviceWorker.ready.then(() => {
              product
              productSub
              removeSiteSpecificTrackingException
              removeWebWideTrackingException
              requestMediaKeySystemAccess
              sendBeacon
              serviceWorker  (property) Navigator.serviceWorke...
              storage
              storeSiteSpecificTrackingException
              storeWebWideTrackingException
          }   userAgent
      }       vendor

      function registerValidSW(swUrl, config) {
        navigator.serviceWorker
          .register(swUrl)
          .then(registration => {
```

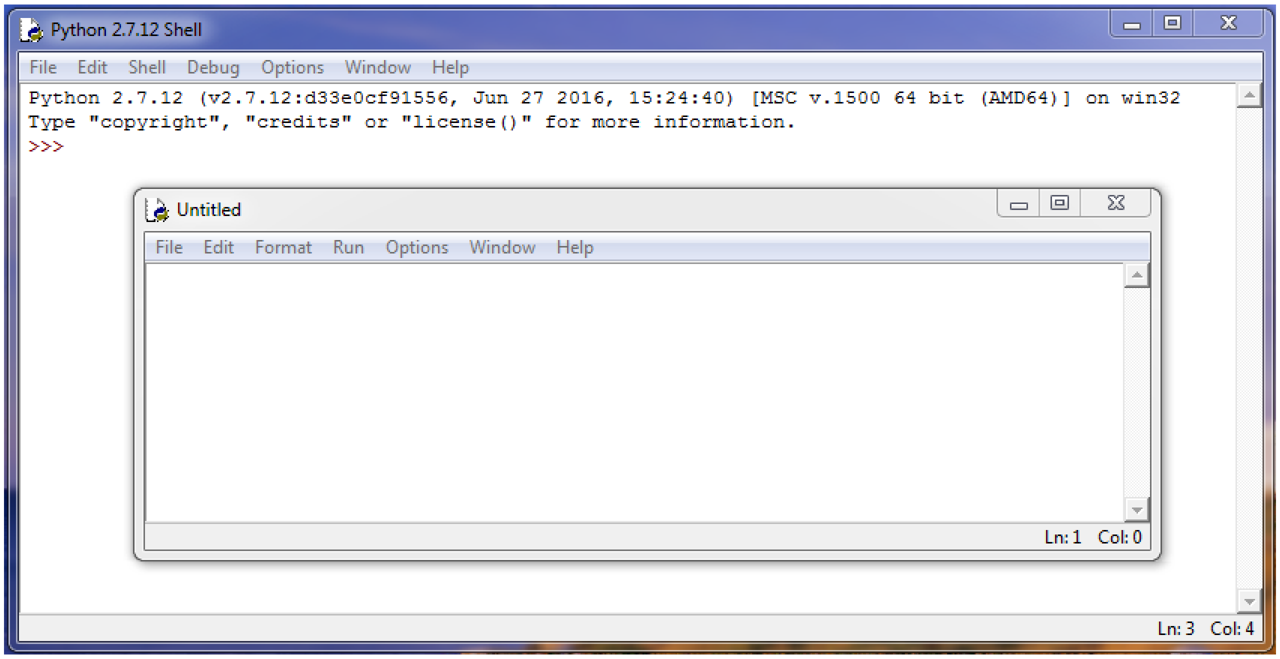
```
You can now view create-react-app in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://10.211.55.3:3000/

Note that the development build is not optimized.
```

IDE: IDLE

- Already installed as a part of ArcGIS Destkop Install...the fall back!
- Let's take a quick tour!
  - Two windows:
    - Python shell window (interactive interpreter)
    - Text Editor window (for scripts)


Python 2.7.12 Shell
File Edit Shell Debug Options Window Help
Untitled
File Edit Format Run Options Window Help
Ln: 1 Col: 0
Ln: 3 Col: 4

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:24:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

II. Review of *Python Basics*

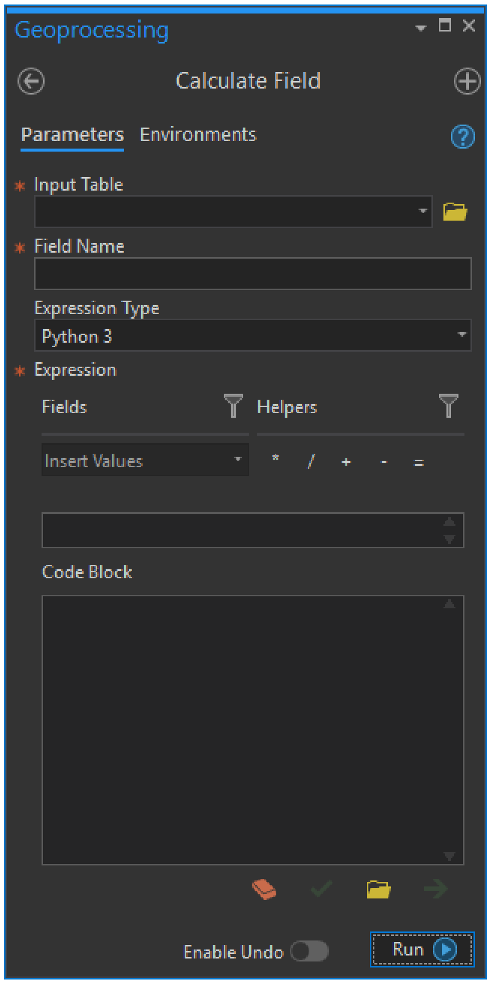
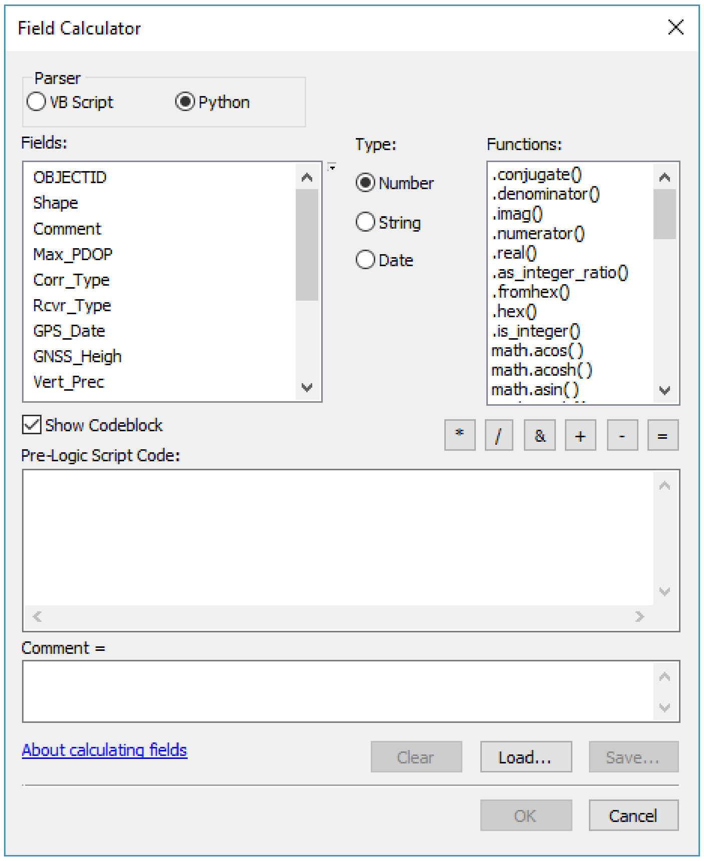
Review of minimal skills needed for those who didn't attend first session

What is Python?

- Print Statement
- Variables
- Basic Data types: Strings, Numbers, Booleans, Lists, Tuples, Dictionaries
- Data Type Conversions
- Simple Math with Python
- Python Basic Syntax
- Conditional Statements
- Functions
- Loops

III. Calculate Fields Using Python

- **A great place to practice Python in ArcGIS is by calculating fields**
- ArcGIS Pro:
  - Python 3, Arcade, and SQL can be used to calculate fields; Python 3 is the default
  - Must use the Calculate Field geoprocessing tool; Field Calculator Window is removed
- ArcMap:
  - Python and VB Script can be used to calculate fields; VB Script is the default, so Python needs to be set as the parser
  - Can use the Calculate Field geoprocessing tool or Field Calculator Window
- Simple expressions use the Expression parameter
- More complex calculations can use the Code Block parameter (tool) or the Pre-Logic Script Code block (window)
- **Note: Python can be difficult to debug and check for errors when calculating fields**

ArcGIS Pro Calculate Field Tool	....	ArcMap Field Calculator Window
	....	

Geoprocessing
Calculate Field
Parameters Environments
* Input Table
* Field Name
Expression Type
Python 3
* Expression
Fields Helpers
Insert Values * / + - =
Code Block
Enable Undo Run

Field Calculator
Parser
VB Script [✓ Python]
Fields:
OBJECTID
Shape
Comment
Max_PDOP
Corr_Type
Rcvr_Type
GPS_Date
GNSS_Heigh
Vert_Prec
Type:
[✓ Number]
String
Date
Functions:
.conjugate()
.denominator()
.imag()
.numerator()
.real()
.as_integer_ratio()
.fromhex()
.hex()
.is_integer()
math.acos( )
math.acosh( )
math.asin( )
[✓] Show Codeblock
* / & + - =
Pre-Logic Script Code:
Comment =
About calculating fields
Clear Load... Save...
OK Cancel

Practice Calculating Fields Using Python

1. Open the *Illinois* Map in the *PythonForArcGIS.aprx* ArcGIS Project File

- In ArcMap, add the Illinois feature class from PythonForArcGIS.gdb into a new ArcMap Document (.mxd)

2. Open the Attribute Table for the 'Illinois Counties' Layer

- We see that there is a field named 'AREA', but we don't know the units.
- Note the other fields

3. Add a field with the following parameters:

- Name: AreaTest
- Type: Double

4. Right-click on the new field header and select Calculate Field

- ArcMp: Note if the VB Script radio button is selected; change it to Python if so

5. See the Calculate Field Python examples (https://pro.arcgis.com/en/pro-app/tool-reference/data-management/calculate-field-examples.htm) ArcGIS Pro help documentation

- This page is a great resource
- Notice the different examples
- Find the 'Code samples-geometry' section
- These expressions can be used in lieu of the Calculate Geometry Attributes tool (ArcGIS Pro) or window (ArcMap)

6. Enter the following code in the expression box

In [ ]:
```
""" Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# First try square meters

!Shape.area@squaremeters!

# How close are the values compared to the AREA field?


# Second, try square miles

!Shape.area@squaremiles!

# Are these values closer to the AREA field? Do you think we can get better?


# Now try

!Shape.geodesicArea@squaremiles!

# Which method is more acurate? Why do you think that is?


# One last trick, type:

None

# This will make all the values NULL in the field...it is good to know about!
```

Practice Calculating Fields Using Python (continued)

- The following examples will be breifly covered, but will not be demostrated

Now we know the units, but we want to calculate the population density. We also want to know the difference between the median and average household income.

1. Delete the 'AreaTest' field

2. Add two fields with the following parameters:

- Name: DensitySqmi
  - Type: Double
  - Alias: Density Per SqMi
- Name: DiffIncome
  - Type: Double
  - Alias: Avergae vs. Median Income

3. Calculate the fields using Python

- Clear the expression parameter

In [ ]:
```
""" Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# DensityPerSqmi calculation

!ACSTOTPOP! / !AREA!

########################################################

# DiffIncome calculation

!ACSAVGHINC! - !ACSMEDHINC!
```

Practice Calculating Fields Using Python (continued)

- The following examples will be breifly covered, but will not be demostrated

Now we would like to have three more fields for a short name, a long name, and a name with the area.

1. Add three fields with the following parameters:

1. Name: NameLong
   - Type: Text
   - Alias: Long Name
   - Length: 100

1. Name: NameShort
   - Type: Text
   - Alias: Short Name
   - Length: 50

1. Name: NameArea
   - Type: Text
   - Alias: Name (with Area)
   - Length: 100

2. Calculate the fields using Python

- Clear the expression parameter

In [ ]:
```
""" Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# NameLong =
!NAME! + ', ' + !STATE_NAME!

########################################################

# NameShort =
!NAME!.replace(' County', '')

# Note the first parameter...what do you notice?

########################################################

# NameArea =
!NAME! + ' (Area: ' + !AREA! + ' SqMi)'

# Did it work? Why or why not? Hint: Check the ArcMap Results window if you can't figure it out
# Change the expression to be correct...


# NameArea can also be calculated this way...

# NameArea =
' '.join([!NAME!, '(Area:', str(!AREA!), 'SqMi)'])

# What do you notice that is different with the elements being joined? I'm looking for 2 specific things...
# Hint: Don't space out on me...and...0, 1, 2, 3


# Or this way...

# NameArea =
'{0} (Area: {1} SqMi)'.format(!NAME!, !AREA!)

# Note this is the best practice...and that you do not need to convert the AREA to a string! More on this late
r...
```

Practice Calculating Fields Using Python (continued)

- The following examples will be breifly covered, but will not be demostrated

Now we would like to classify the median income levels.

1. Add a field with the following parameters:

- Name: IncomeLevel
  - Type: Text
  - Alias: Income Level
  - Length: 50

2. Calculate the fields using Python

- Clear the expression parameter
- ArcMap: Check the 'Show Codeblock' box

3. When you have completed the calculation, open the result from the Geoprocessing History (ArcGIS Pro) or Results (ArcMap)

- How does this look different?

In [ ]:
```
""" Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool. """

# When calulating a field, a function usually is used in the
# Code Block (tool) Pre-Logic Script Code (window):

def income_level(median_income, low_income, mid_income):
    """Classifies median income levels given low income and middle income values"""
    if median_income < low_income:
        return 'Low Income'
    elif low_income <= median_income < mid_income:
        return 'Middle Income'
    else:
        return 'High Income'


# Expression:
# IncomeLevel =
income_level(!ACSMEDHINC!, 35000, 60000)
```

Practice Calculating Fields Using Python (continued)

- The following examples will be breifly covered, but will not be demostrated

Now we would like to know how many vertices are in each feature...just for fun!

We will use an example from the ArcGIS Help Documentation: Calculate Field Python examples (https://pro.arcgis.com/en/pro-app/tool-reference/data-management/calculate-field-examples.htm)

1. Add a field with the following parameters:

- Name: VertexCount
  - Type: Long Integer
  - Alias: Number of Vertices

2. Calculate the fields using Python

- Clear the expression parameter
- ArcMap: Check the 'Show Codeblock' box

Feel free to **Copy and Paste** the code below; it is for demonstrating that while loops can be used in the Field Calculator

In [ ]:
```python
""" Note: The following code is intended to be used to calculate an attribute table field in
ArcGIS using the Field Calculator window or Calculate Field tool."""

# Code Block / Pre-Logic Script Code:
def VertexCount(feat):
    partnum = 0

    # Count the number of points in the current multipart feature
    partcount = feat.partCount
    pntcount = 0

    # Enter while loop for each part in the feature (if a singlepart
    # feature this will occur only once)
    #
    while partnum < partcount:
        part = feat.getPart(partnum)
        pnt = part.next()

        # Enter while loop for each vertex
        #
        while pnt:
            pntcount += 1
            pnt = part.next()

            # If pnt is null, either the part is finished or there
            # is an interior ring
            #
            if not pnt:
                pnt = part.next()
        partnum += 1
    return pntcount


# Expression:
# VertexCount =
VertexCount(!Shape!)
```

Future Challenge: Try using Python with map labels in ArcGIS

IV. Introduction to ArcPy

Python in ArcGIS (Pro)

- First introduced at ArcGIS Desktop 9.0 with Python 2 and `arcgisscripting` site package
- ArcGIS Desktop 10 introduced `arcpy` site package
- ArcGIS Pro uses Python 3 `arcpy` site package
- ArcGIS Pro integrates Conda to manage Python packages and modules via the **Python Package Manager (https://pro.arcgis.com/en/pro-app/arcpy/get-started/what-is-conda.htm)**
- ArcPy is the primary desktop Python site package designed exclusively for ArcGIS Desktop
- **ArcGIS API for Python (https://pro.arcgis.com/en/pro-app/arcpy/get-started/arcgis-api-for-python.htm)** is designed for ArcGIS Online and Portal for ArcGIS
  - We will not go over this API in this workshop

What is ArcPy?

> ArcPy is a Python site package that provides a useful and productive way to perform geographic data analysis, data conversion, data management, and map automation with Python.
>
> This package provides a rich and native Python experience offering code completion (type a keyword and a dot to get a pop-up list of properties and methods supported by that keyword; select one to insert it) and reference documentation for each function, module, and class.
>
> The additional power of using ArcPy is that Python is a general-purpose programming language. It is interpreted and dynamically typed and is suited for interactive work and quick prototyping of one-off programs known as scripts while being powerful enough to write large applications in. ArcGIS applications written with ArcPy benefit from the development of additional modules in numerous niches of Python by GIS professionals and programmers from many different disciplines.

From ArcGIS Pro Help Documentation - What is ArcPy? (https://pro.arcgis.com/en/pro-app/arcpy/get-started/what-is-arcpy-.htm)

What is a Python Module?

- Extensions that can be imported and used in Python scripts to expand the built-in capabilities
- Contain pre-written code to help out with specialized tasks and defines functions, classes and variables for those tasks
- Also allows you to logically organize your Python code
- Single file consisting of Python code and also includes executable code
- Modules are imported into a script using the **`import`** statement
- Many times you need to download and install new modules; but ArcGIS includes many of the most popular ones

What is a Python Site Package?

- Like a module, but contains a collections of modules, functions, and classes
- Site packages and modules both add functionality to Python

`import` Statements

- We aren't able to use everything Python has to offer without importing modules
- Only need to import a module once in a script
- Customary to import everything at the beginning of your script

```
import arcpy
  import os
  import sys

  # Best practice is to stack each import statement as above, but this is not wrong:
  import arcpy, os, sys

  import arcpy as a # This can sometimes make things simpler; you can use a.whatever instead of arcpy.whatever
  from arcpy import da # Do this if you only want a specific part of the site package/module
  from arcpy.sa import * # This also works, but should be used sparingly
```

Importing ArcPy

- To work with ArcPy, you must `import` it
- Once imported, you can now use all the modules, functions, tools, and classes

See Importing ArcPy (https://pro.arcgis.com/en/pro-app/arcpy/get-started/importing-arcpy.htm) for more information

In [ ]:
```
# It is good practice to always import the ArcPy site package, even in ArcGIS Notebooks, and other modules fir
st in your script
# This may take a while if you are working outside of ArcGIS

import arcpy
```

Utilizing the Python and ArcPy ArcGIS Desktop Help Documentation

Python and ArcPy Help

- ArcGIS Python Help (https://pro.arcgis.com/en/pro-app/arcpy/main/arcgis-pro-arcpy-reference.htm)
- Contains help on functions and classes not listed in the tool documentation
- Note the additional modules

Python and ArcPy Geoprocessing Tool Help

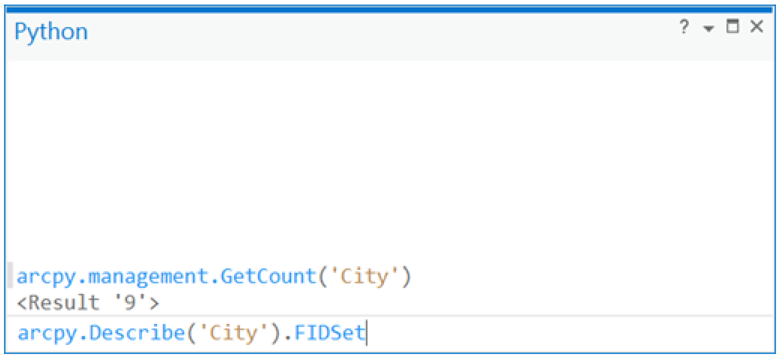
- Each geoproscceing tool includes syntax and sample code for the Python window and stand-alone scripts
- Includes detailed explanations of each parameter
- See **Calculate Field (https://pro.arcgis.com/en/pro-app/tool-reference/data-management/calculate-field.htm)** tool help as an example

V. Running Python and ArcPy Scripts in ArcGIS

- **ArcGIS Pro**
  - Python Window
  - ArcGIS Notebooks
- **ArcMap**
  - Python Window ONLY

Python Window

- Bottom section is the interactive Python prompt
  - This is where code is entered
- Top section is the transcript of the code ran and the output
- Code is generally executed one line at a time and displayed immediately. Exceptions include:
  - Multi-line constucts (such as `if` statements)
  - Hitting ENTER at the end of multi-line code will run it (you may need to hit ENTER a few times when you are ready to run the code)
- Other Advanatages of the Python window
  - Autocompletion
  - Conditional and Iteration executuion
  - Scripts can be saved and reused or opened with another Python IDE

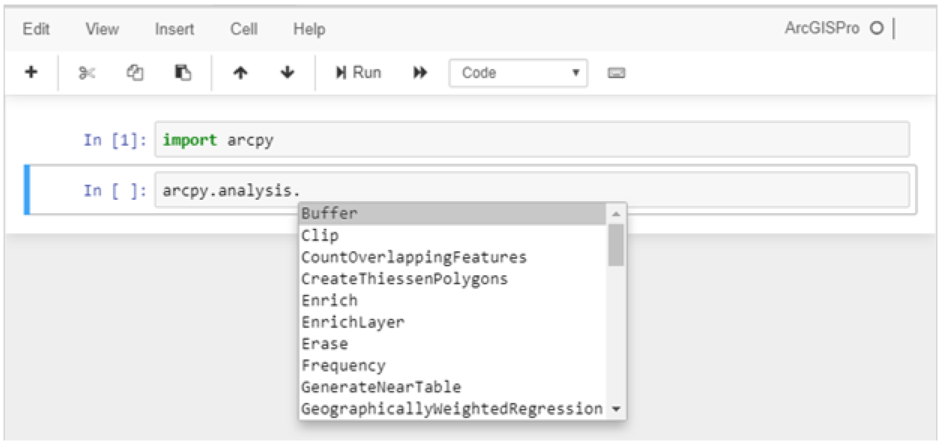


```
arcpy.management.GetCount('City')
<Result '9'>
arcpy.Describe('City').FIDSet
```

See **Python Window (https://pro.arcgis.com/en/pro-app/arcpy/get-started/python-window.htm)** for more info

ArcGIS Notebooks

- Built on top of Jupyter Notebook (https://jupyter.readthedocs.io/en/latest/)
- 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
- Provides access to content in your map allowing for interactive workflows
- All Python functionality in ArcGIS Pro is available through ArcGIS Notebooks
- To create a notebook, click the **Analysis** tab, and click the **Python** button



```
In [1]: import arcpy
In [ ]: arcpy.analysis.
```

```
Buffer
Clip
CountOverlappingFeatures
CreateThiessenPolygons
Enrich
EnrichLayer
Erase
Frequency
GenerateNearTable
GeographicallyWeightedRegression
```

Describing Data with ArcPy

- Describing data allows to learn more about the data we are working with
- Describe functions are useful when scripts may be dependent on the type of data being used
- Good for controlling script flow and valdating parameters
- Many property groups and some properties exist for only some types

See ArcGIS Documentation: https://pro.arcgis.com/en/pro-app/arcpy/functions/describe.htm (https://pro.arcgis.com/en/pro-app/arcpy/functions/describe.htm)

System Paths vs. Catalog paths

- System paths are recognized by the Windows operating system
  - Files in folders
  - Ex. Shapefiles and rasters
- Catalog paths are only recognized by ArcGIS
  - Used for feature classes and other data in geodatabases
  - Geodatabases could be file (.gdb) or enterprise (.sde)
  - Contain 2 parts:
    - Workspace - could be the geodatabase root or a feature dataset
    - Base name - the feature class, raster, or other file types that can be saved in geodatabases
  - Requires the programmer to be aware of the context in which the path is being used

Type, do not copy, the following code into a blank Notebook cell or the Python window

Note: The following code is intended to only be run in an ArcGIS Notebook or the Python window of ArcGIS

In [ ]:
```
# Step 1:
import arcpy

file_name = r'C:\Python-for-ArcGIS-ILGISA-2020\CSV\JeopardyContestants_LatLon.csv'
print(arcpy.Exists(file_name))

# What kind of path is this? System or Catalog?
# What is the result?


# Try again, but this time type the up arrow to reload the last code
# Replace the '...' with the location where you placed your downloaded data

file_name = r'C:\...\Python-for-ArcGIS-ILGISA-2020\CSV\JeopardyContestants_LatLon.csv'
print(arcpy.Exists(file_name))

# Does it exist? Keep trying until you get the path correct and the result returns True


print(arcpy.Describe(file_name).dataType)

# What type of data is this?
```

In [ ]:
```
# Step 2:

# Replace the '...' with the location where you placed your downloaded data

feature_class = r'C:\...\Python-for-ArcGIS-ILGISA-2020\PythonForArcGIS.gdb\Illinois_Counties'
# What kind of path is this?

print(arcpy.Exists(feature_class))

print(arcpy.Describe(feature_class).dataType)
# What type of data is this?
```

In [ ]:
```
# Step 3:

layer_name = 'Illinois Counties'

# What kind of path is this? System or Catalog?

print(arcpy.Exists(layer_name))

lyr_desc = arcpy.Describe(layer_name)
# Note how this Describe function and variable look different than before...
# We can have describe objects become variables to be used later

print(lyr_desc.dataType)
# What type of data is this?

# For layers, there is a Describe property called 'dataElement' that accesses
# the 'dataType' of what the layer is referencing

print(lyr_desc.dataElement.dataType)
# What type of data is this?
```

What is the difference between the last two `print()` results?

Lets look at the help:

- https://pro.arcgis.com/en/pro-app/arcpy/functions/describe-object-properties.htm (https://pro.arcgis.com/en/pro-app/arcpy/functions/describe-object-properties.htm)
- https://pro.arcgis.com/en/pro-app/arcpy/functions/layer-properties.htm (https://pro.arcgis.com/en/pro-app/arcpy/functions/layer-properties.htm)

In [ ]:
```
# Step 4:

lyr_datatype = lyr_desc.dataElement.dataType

lyr_path = lyr_desc.path

lyr_basename = lyr_desc.baseName

lyr_spatialref = lyr_desc.spatialReference.name

lyr_count = arcpy.GetCount_management(layer_name)

# Here is a new way to format strings with inline variable substitution

print('{0} is a {1} stored at {2} with a file name of {3} with the {4} coordinate system and contains {5} feat
ures.'\
    .format(layer_name, lyr_datatype, lyr_path, lyr_basename, lyr_spatialref, lyr_count))

    # The slash does not need to be typed, it is there for formatting purposes
```

Formatting Strings

- Formatting strings pretty slick
- Can save space and time

*See https://pyformat.info/ (https://pyformat.info/) and https://docs.python.org/3/library/string.html#custom-string-formatting (https://docs.python.org/3/library/string.html#custom-string-formatting) and https://realpython.com/python-f-strings/ (https://realpython.com/python-f-strings/)

In [ ]:
```
# Here is another way to quickly format strings
print(f'{layer_name} is a {lyr_datatype} stored at {lyr_path} with a file name of {lyr_basename}\
with the {lyr_spatialref} coordinate system and contains {lyr_count} features.')

    # The slash does not need to be typed, it is there for formatting purposes
```

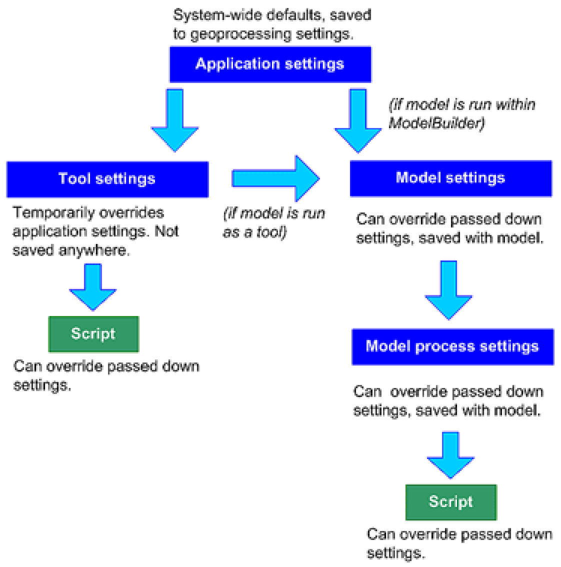
Listing Data with ArcPy

- Helps with batch processing (primary reason for developing scripts)
- Lists are good for interating processes using loops (another primary reason for developing scripts)
- Easy inventory of data (e.g. list of fields in a table; feature classes in a geodatabase)
- In order to list data, mant time need to utilize the environmental settings class

Environmental Settings

https://pro.arcgis.com/en/pro-app/tool-reference/environment-settings/what-is-a-geoprocessing-environment.htm (https://pro.arcgis.com/en/pro-app/tool-reference/environment-settings/what-is-a-geoprocessing-environment.htm)

- Hidden parameters that influence how tools run
- Fundamental in controlling the geoprocessing workflow
- Exposed as properties and set using the `arcpy.env` class
- Corresponds to the 'Environments' found in geoprocession tool dialog
- There is a hierarchy with how ArcGIS deals with environmental settings


System-wide defaults, saved to geoprocessing settings.
Application settings
(if model is run within ModelBuilder)
Tool settings
Temporarily overrides application settings. Not saved anywhere.
(if model is run as a tool)
Model settings
Can override passed down settings, saved with model.
Script
Can override passed down settings.
Model process settings
Can override passed down settings, saved with model.
Script
Can override passed down settings.

See also:

- https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/using-environment-settings.htm (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/using-environment-settings.htm)
- https://pro.arcgis.com/en/pro-app/arcpy/classes/env.htm (https://pro.arcgis.com/en/pro-app/arcpy/classes/env.htm)

Type, do not copy, the following code into a blank Notebook cell or the Python window

Clear the Python transcript if needed

Note: The following code is intended to only be run in an ArcGIS Notebook or the Python window of ArcGIS

In [ ]:
```
# Step 1:

# First, we must set the environmental settings
# Replace the '...' with the location where you placed your downloaded data
# Don't forget...drag, drop, and roll!
arcpy.env.workspace = r'C:\...\Python-for-ArcGIS-ILGISA-2020\CSV'

print(arcpy.ListFiles())

# How many files are listed?


arcpy.env.workspace = r'C:\...\Python-for-ArcGIS-ILGISA-2020\PythonForArcGIS.gdb'

print(arcpy.ListFiles())

print(arcpy.ListFeatureClasses())

# What is the difference between the two list functions?
```

In [ ]:
```
# Step 2:

print(arcpy.ListFields('Illinois Counties'))

# What is the result? Can you understand it?
# What is actually being output in this case? (Hint: What type/class of object is being displayed?)

fields = arcpy.ListFields('Illinois Counties')
print(type(fields[0]))

# We can print each field name...
for field in fields:
    print(field.name) # '.name' is property of the Field class

# Or the field aliases...
for field in fields:
    print(field.aliasName) # '.aliasName' is another property of the Field class
```

In [ ]:
```
# Step 3:
# We can also create a list of the field names; this is called a 'list comprehension' and they are very powerf
ul

field_list = [field.name for field in arcpy.ListFields('Illinois Counties')]

print(field_list)

# How does this result look differently than above in Step 2?

test_field = 'OBJECTID'

if test_field in field_list:
    print('{} exists'.format(test_field))
else:
    print('{} does not exist'.format(test_field))

# Repeat the above code using a test field that you know doesn't exist to test the 'else:' statement code
# Does it work?
```

List Comprehensions

- Used for creating list concisely and quickly
- Consists of brackets containing an expression followed by a `for` clause
- Optionally can have one or more `if` or `for` clauses
- Result will be a new list resulting from evaluating the expression in the context of the `for` and `if` clauses
- Always returns a list as a result

See:

- https://pro.arcgis.com/en/pro-app/arcpy/functions/listfields.htm (https://pro.arcgis.com/en/pro-app/arcpy/functions/listfields.htm)
- http://www.learnpython.org/en/List_Comprehensions (http://www.learnpython.org/en/List_Comprehensions)
- http://www.pythonforbeginners.com/basics/list-comprehensions-in-python (http://www.pythonforbeginners.com/basics/list-comprehensions-in-python)

Once you understand these, you are on your way to becoming a Python ninja!

Geoprocessing Tools with ArcPy

- ArcPy gives access to all geoprocession tools in ArcGIS in addition to many non-tool functions
- Working in the Python window allows for testing ideas and seeing how things work
- Tested scripts can also be saved for reuse in the Python window, Jupyter Notebook, or in an IDE or script editor

Type, do not copy, the following code into a blank Notebook cell or the Python window

Clear the Python transcript if needed

Note: The following code is intended to only be run in an ArcGIS Notebook or the Python window of ArcGIS

In [ ]:
```
# Step 1:
import arcpy

layer_name = 'Illinois Counties'

expression = 'ACSMEDHINC <= 40000'

arcpy.SelectLayerByAttribute_management(layer_name, 'NEW_SELECTION', expression)
```

In [ ]:
```
# Now clear the selection
arcpy.management.SelectLayerByAttribute(layer_name, 'CLEAR_SELECTION')

# Note the difference in how the code is written...See note below
```

Tool organization in ArcPy

> Geoprocessing tools are organized in two different ways. All tools are available as functions on ArcPy but are also available in modules matching the toolbox alias name. Although most of the examples in the help show tools organized as functions available from ArcPy, both approaches are equally valid. Which approach you use will come down to a matter of personal preference and coding habits. In the following example, the Get Count tool is shown using both approaches.

From **Using tools in Python - Tool organization in ArcPy (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/using-tools-in-python.htm#ESRI_SECTION1_9C29CCC8373B4623BF5FD1F31EFEDBF2)**

In [ ]:
```
# Step 2:

# Ensure that a default geodatabase is set
arcpy.env.workspace = r'C:\...\Python-for-ArcGIS-ILGISA-2020\PythonForArcGIS.gdb'

# Create a variable for dissolve fields
dissolve_fields = ['STATE_NAME', 'ST_ABBREV']

arcpy.Dissolve_management(layer_name, 'Illinois', dissolve_fields, '', 'SINGLE_PART', 'DISSOLVE_LINES')

arcpy.PolygonToLine_management('Illinois', 'Illinois_Boundary')

# You may want to rearrage your layers in the TOC
```

In [ ]:
```
# Step 3:

# Replace the '...' with the location where you placed your downloaded data

csv_file = r'C:\...\Python-for-ArcGIS-ILGISA-2020\CSV\JeopardyContestants_LatLon.csv'

arcpy.CopyRows_management(csv_file, 'JeopardyContestants_Table')

arcpy.MakeXYEventLayer_management('JeopardyContestants_Table', 'lon', 'lat', 'Jeopardy Contestants')

arcpy.Select_analysis('Jeopardy Contestants', 'JeopardyContestants', '"lat" IS NOT NULL OR "lon" IS NOT NULL')
# Note the quotes!!

arcpy.Buffer_analysis('JeopardyContestants', 'JeopardyContestants_Buffer', '5 Miles', 'FULL', 'ROUND', 'ALL',
'', 'GEODESIC')
```

Break

VI. Standalone Scripts with Python

Scenario:

> Many of your GIS workflows require creating an Excel file of feature classes and standalone tables to share with your colleagues. The ArcGIS **Table to Excel (https://pro.arcgis.com/en/pro-app/tool-reference/conversion/table-to-excel.htm)** tool has been working, but you recently learned that your colleagues have been deleteing unnecessary fields and exporting the Excel files to CSV and they would like to elminate these steps from their workflow. Therefore, they have asked you to deliver files as CSV files wihtout the unnecessary fields.
>
> Exisitng ArcGIS tools don't *easily* allow for selecting a subset of fields to be exported. Therefore, you would like to create a custom script to make this task easier. So, you decide to create a standalone script...

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

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

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 (e.g. '#')

Note: This is just one of many methodologies...

- You can modify this as much as you want
- Find a stucture that works for you
- Keep it simple

General Components of a Standalone Script for ArcGIS

Below is a template you can start from:

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

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

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

''' Functions '''
# Any 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

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

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

Create Pseudo Code for the Standalone Script

0. Open the PA Unconventional Wells Map

- Close all other maps
- In ArcMap, create a new Map Document (.mxd)

1. Create a new script file in the IDE or script editor

2. Save it to the 'Scripts' folder named as *TableToCSV_Standalone.py*

3. What are 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 reads CSV files

'Human-readable' Pseudo Code:

```
1. Input a feature class or standalone table
2. Create user-defined list of the fields to be exported
3. Read the feature class or standalone table data using a cursor
4. Create and open a new output CSV file
5. Write the field names to the new CSV file
6. Write the feature class or standalone table cursor data to the new CSV file
```

4. Add the pseudo code to the script

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

''' Metadata, Copyright, License:
------------------------------------------------------------------------
Name:       TableToCSV_Standalone.py
Purpose:    This script converts a table to a CSV table with selected
            fields.
Author:     Whitacre, James
Created:    2020/10/06
Copyright:  Copyright <Your Organization/Name> <YYYY>
Licence:    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 '''


''' Functions '''


''' Parameters '''
# Input feature class or standalone table

# Create user-defined list of the feature class or standalone table field names to be exported

# Create a new output CSV file


''' Script '''
# Read the feature class or standalone table data

# Open the new 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?

- Is there already some Python code out there to help me wirte my code more effeciently?
- This is dependent on the processes and you may learn later what needs to be added (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 you need 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
```

6. What user input and output parameters are needed?

- Think about this like an ArcGIS tool...
- What variables may change every time the script is run?
- Typically these are paths or other values that may be dynamic
- Key is what variables in the script are dependent on input from the user

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

- 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/ser
vices/UnconventionalWellsPA/FeatureServer/0', 'Unconventional Wells')

# Create user-defined list of the feature class or standalone table field names to be exported
field_names = ['Shape', 'PERMIT_NO', 'FARM_NAME', 'COUNTY', 'PROD_GAS_QUANT']

# Create a new output CSV file
# Replace the '...' with the location where you placed your downloaded data
output_csv = r'C:\...\Python-for-ArcGIS-ILGISA-2020\CSV\UnconventionalWells.csv'
```

7. ArcGIS Environments and Variables?

- These items may or may not be needed; as you write your code, this will become more aparent
- 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
- Because this is a simple script, we will not need to set any `# Environments` or `# Variables` ; we will revisit this later

8. Develop the code

- First, we need to read 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

- 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(dataset)`) and the other in the Data Access module within ArcPy (e.g. `arcpy.da.SearchCursor(dataset)`). 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.*

For more information on Cursors, see: https://pro.arcgis.com/en/pro-app/arcpy/get-started/data-access-using-cursors.htm (https://pro.arcgis.com/en/pro-app/arcpy/get-started/data-access-using-cursors.htm)

- Now, we need to actually create the CSV file...in the Parameters section above, we just established the name of the file
- This can be done using a `with open()` function
  - For Python 3, we will 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
  - For Python 2, we use the `'wb'` write mode
  - For Python 3, we also need to identify what creates a `newline` when writing to the file

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

- Next, create a CSV writer object with the newly create CSV file
  - Again, note the additional comments in the code

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

- Finally write the field names as the header and the data as seperate lines in the CSV file

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

# Open the new output CSV file

# Python 2.x...For some reason, 'w' adds an extra line between rows, use 'wb' instead...
# with open(output_csv, 'wb') as csv_file:

# Python 3.x
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))
```

Final Standalone Script Code:

In [1]:
```
# -*- coding: UTF-8 -*-

''' Metadata, Copyright, License:
------------------------------------------------------------------------
Name:       TableToCSV_Standalone.py
Purpose:    This script converts a table to a CSV table with selected
            fields.
Author:     Whitacre, James
Created:    2020/10/06
Copyright:  Copyright <Your Organization/Name> <YYYY>
Licence:    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 '''

# Input feature class or standalone table
input_table = arcpy.management.MakeFeatureLayer('https://services2.arcgis.com/eQgAMgHr2CRobt2r/arcgis/rest/ser
vices/UnconventionalWellsPA/FeatureServer/0', 'Unconventional Wells')

# Create user-defined list of the feature class or standalone table field names to be exported
field_names = ['Shape', 'PERMIT_NO', 'FARM_NAME', 'COUNTY', 'PROD_GAS_QUANT']

# Create a new output CSV file
# Replace the '...' with the location where you placed your downloaded data
output_csv = r'C:\Users\whitacrej\Documents\GitHub\Python-for-ArcGIS-ILGISA-2020\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)


# Open the new output CSV file

# Python 2.x...For some reason, 'w' adds an extra line between rows, use 'wb' instead...
# with open(output_csv, 'wb') as csv_file:

# Python 3.x
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))
```

```
CSV file complete; located at C:\Users\whitacrej\Documents\GitHub\Python-for-ArcGIS-ILGISA-2020\CSV\Unconvent
ionalWells.csv
```

Challenge:

How might you change the code to use fewer lines?

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

VII. Creating ArcGIS Script Tools

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 an IDE or the ArcGIS Python Window, update the variables, and then run the code. You think, "Man it would be nice to have this as a tool in ArcGIS...why hasn't Esri created this tool?"

What is an ArcGIS Script Tool?

Creating a script tool allows you to turn your own Python scripts and functionality into your own geoprocessing tools—tools that look and act like system geoprocessing tools. Once created, a script tool provides many advantages:

- A script tool that you create is an integral part of geoprocessing, just like a system tool—you can open it from the Catalog pane, use it in ModelBuilder and the Python window, and call it from another script.
- You can write messages to the Geoprocessing history and tool dialog box.
- Using built-in documentation tools, you can provide documentation.
- When the script is run as a script tool, arcpy is fully aware of the application it was called from. Settings made in the application, such as `arcpy.env.overwriteOutput` and `arcpy.env.scratchWorkspace` , are available from ArcPy in your script tool.

To create a script tool in a custom toolbox, you need three things:

- [X] A script
- [X] A custom toolbox
- [ ] A precise definition of the parameters of your script

http://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/a-quick-tour-of-creating-script-tools.htm (http://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/a-quick-tour-of-creating-script-tools.htm)

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	Relativiely 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 IDE, Python window, or Command Line	Uses the ArcGIS Tool dialog

Creating Scripts for Script Tools

Scenario Update:

> Ok, so by now 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.

- Use the template below as a guide
- Modify the template: (**Bold** indicates required)
  - **Determine module imports** - as script development progresses, you may need to add more later
  - Determine if any functions will be used
  - **Determine the input, output, and derived parameters**
  - **Add and modify the script**
- Save the modified script as the name you give it (it is ok to overwrite for this exercise, but for your own scripts, I would suggest saving as to preserve the original script)
- Note that the original stand alone script is already implementing some of the template elements
- Also note that the templplate conforms to the **PEP 8 -- Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)**

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 includ
e the brackes.
# Delete lines between    # -*- coding: UTF-8 -*-    and    ''' Metadata, Copyright, License:

''' 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>
Licence:     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
```

Modifying the Standalone Script Parameters for a Script Tool

- Copy the **TableToCSV_Standalone.py** standalone script file and name it **TableToCSV.py**
- 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 scritp itself
- This will make a bit more sense when we create the Script Tool in an ArcGIS Toolbox

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)
```

Adding a Script Tool to an ArcGIS Toolbox

Now that the parameters are modified to be used in a Script Tool, we can create on in an ArcGIS Toolbox

- See Adding a Script Tool (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/adding-a-script-tool.htm) ArcGIS Pro Help Documentation for the detailed instructions; otherwise, follow the demonstration in ArcGIS Desktop
- **Set Properties**
  - Name: **ConvertTableToCSV**
  - Label: **Convert Table to CSV File**
  - Script: **The path to the script you just modified/created**
- **Set Parameters**
  - Input Table
    - Data Type: **Table View**
    - Direction: **Input**
  - Output Fields
    - DataType: **Field**
    - Direction: **Input**
    - MultiValue: **Yes**
    - Filter: field: **All EXCEPT: Blob, Raster, XML**
    - Dependency: **Input Table**
  - Ouput CSV Table
    - Data Type: **File**
    - Direction: **Output**
    - Filter: File: **csv; txt** (delimit by semi-colon)
- **Validation**
  - We'll come back to this!

Adding ArcPy elements to expand the script tool functionality

- Progressors (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/understanding-the-progress-dialog-in-script-tools.htm)
  - Shows 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 (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/understanding-messaging-in-script-tools.htm)
  - Essentially acts like the Python `print()` function, but in the Details of script
  - Good for passing messages about optional tasks that completed, are anomolies/uncommon occurences, 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: ⊗

Let's Add some Progressors and a Message to the Script

- Add a step progressor to track writing the rows
- Reset the progressor and add a default progressor to notify that the CSV file is being created
- Add a message to let users know that the script completed and how many rows and field were written

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

# Read the feature class or standalone table data

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

# 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()


# Open the new output CSV file

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

# Python 2.x...For some reason, 'w' adds an extra line between rows, use 'wb' instead...
# with open(output_csv, 'wb') as csv_file:

# Python 3.x
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)))
```

Test the Script Tool in ArcGIS Pro

Parameters:

- Input Table: **Unconventional Wells**
- Output Fields: **Shape, PERMIT_NO, FARM_NAME, COUNTY, PROD_GAS_QUANT**
- Output CSV Table: **C:...\Python-for-ArcGIS-ILGISA-2020\CSV\UnconventionalWells_Tool.csv** (this will distinguish it from the previous CSV file for camparison)

Results

- 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**

Error Handling with Script Tools

Three types of errors in Python

- **Syntax Errors** - prevents code from running
- **Exceptions** - code will stop running midprocess
- **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 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...https://pro.arcgis.com/en/pro-app/arcpy/data-access/searchcursor-class.htm (https://pro.arcgis.com/en/pro-app/arcpy/data-access/searchcursor-class.htm)

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

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

''' 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

```
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.

If Time Permits...

Scenario Update:

> Some of your geodatabase feature classes and tables have field aliases that are much easier for the recipients 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.

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

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

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



''' Script '''

# Read the feature class or standalone table data

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

# 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(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'''

# Open the new output CSV file

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

# Python 2.x...For some reason, 'w' adds an extra line between rows, use 'wb' instead...
# with open(output_csv, 'wb') as csv_file:

# Python 3.x
with open(output_csv, 'w', newline='') as csv_file:
    # Creates CSV Writer object
    csv_writer = csv.writer(csv_file)

    ''' Modified 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 Modified 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)))
```

Scenario Update:

> Sometimes geodatabase feature classes and 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

Validation

- Validation (https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/understanding-validation-in-script-tools.htm)
  - 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

Let's Add Validation to the Script Tool

- Open the **Convert Table to CSV File** script tool **Properties**
- Click Validation
- Overwrite the Validation code with the code below:

In [ ]:
```
import arcpy
class ToolValidator(object):
    """Class for validating a tool's parameter values and controlling
    the behavior of the tool's dialog."""

    def __init__(self):
        """Setup arcpy and the list of tool parameters."""
        self.params = arcpy.GetParameterInfo()

    def initializeParameters(self):
        """Refine the properties of a tool's parameters.
        This method is called when the tool is opened."""
        # Disable the optional alias parameter
        self.params[3].enabled = False

    def updateParameters(self):
        """Modify the values and properties of parameters before internal
        validation is performed. This method is called whenever a parameter
        has been changed."""

        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 (op
tional)' 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):
        """Modify the messages created by internal validation for each tool
        parameter. This method is called after internal validation."""

    def isLicensed(self):
        """Set whether tool is licensed to execute."""
        return True
```

Automating Geoprocessing Tools with Python

- So far we haven't worked with many geoprocessing tools...So let's do one!
- First, we will run the tools in ArcGIS Pro, then export it as a Python command

Psuedo Code:

- Select Drilled and Producing Wells
  - Add the Expression as a parameter
  - Optional: Add this step earlier in the workflow so that the CSV file only includes these records
- Buffer each selected well at 100 m, dissolved (this will estimate the well pad area)
  - Add the output buffer feature class as a parameter
  - Add the buffer distance as a parameter

Add this code to the end of the script and modify as needed

Documenting Script Tools

- Right click on the the Script Tool
- Click Edit Metadata
  - Borrow and steal from Esri's script tools
  - Be very descriptive, but concise
- Save metadata edits
- Open the script tool and view the help

VIII. Conclusion and Moving Forward

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!')
```