MONTY PYTHON AND THE UNHOLY SCRIPT

Kelsey Troyer

PYTHON FACTS

- Guido van Rossum, Dutch programmer
- Named after the British comedy troupe Monty Python
- Considered to be one of the easier programming languages
- Free and open source
- Type “Import this” in for a poem by Tim Peters

```
import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

AUTOMATION

- You don't have to be an expert
- ArcMap was automated by Python2.X
- ArcGIS Pro is automated by Python3.X
- Python 3.X comes with your ArcGIS Pro license
- There are some formatting changes in Python3.X
  - print"string" to print("string")

```
>>> print("Hello World")
Hello World
>>> _
```

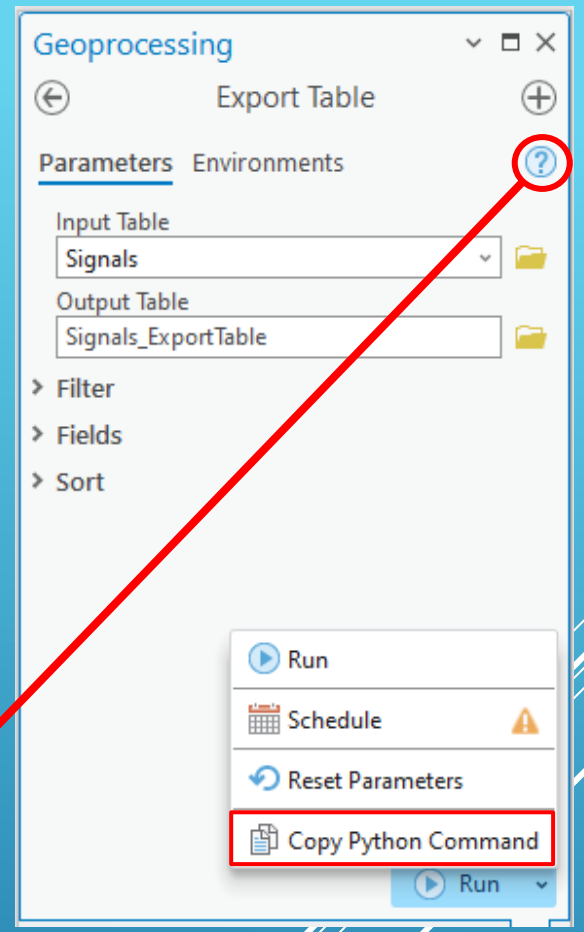
- Repetitive tasks
- Cumbersome tasks
- Data requests
- Reports
- And much more…

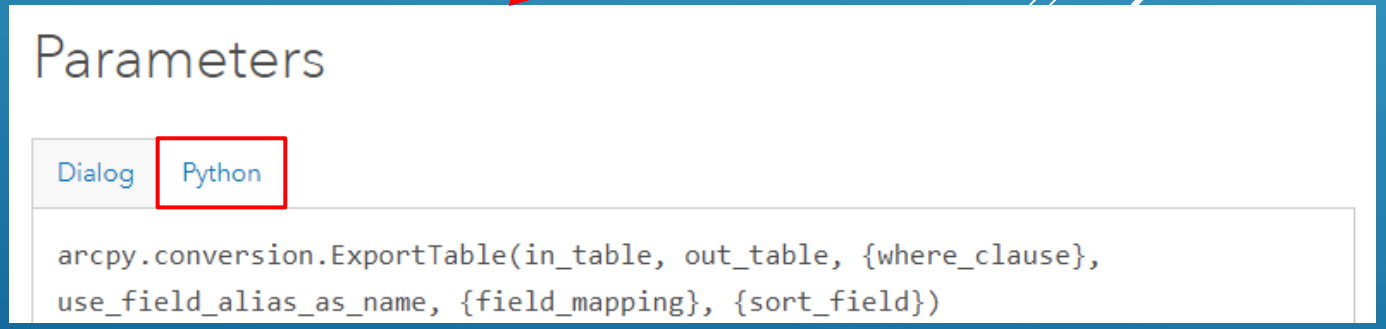
```
import arcpy, os, shutil, time
start_time = time.time()
#Asking for inputs of file location, destination, and name
arcpy.env.workspace = input("Provide the file path to the aerial files: ")
final_dest = input("Provide final destination file path: ")
grid = input("What do you want to name the final grid? (No Spaces): ")
use_extent = input("If you have a specific area you need the grid for, specify the file path to the shape area.\
Otherwise type 'No': ")
temp = "Temp_Folder"
path = os.path.join(final_dest, temp)
#Making temporary folder for individual layers
os.mkdir(path)
print("Starting grid creation...")
raz = arcpy.ListRasters()
#Creating individual vector layers of the raster boundaries for the entire folder of aerials
if use_extent == "No":
    for raster in arcpy.ListRasters():
        arcpy.ddd.RasterDomain(raster, os.path.join(final_dest, temp, os.path.basename(raster)), "POLYGON")
        print(os.path.basename(raster) + " complete")
#Creating individual vector layers of the raster boundaries based on provided extent
else:
    for raster in arcpy.ListRasters():
        rDesc = arcpy.Describe(raster)
        rExt  = rDesc.extent
        with arcpy.da.SearchCursor(use_extent,"SHAPE@") as sCur:
            IntersectsShape = False
            for ft in sCur:
                if not ft[0].disjoint(rExt):
                      IntersectsShape = True
                      break
            if IntersectsShape:
                arcpy.ddd.RasterDomain(raster, os.path.join(final_dest, temp, os.path.basename(raster)), "POLYGON")
                print(os.path.basename(raster) + " complete")
print("Merging each grid into one file...")
arcpy.env.workspace = os.path.join(final_dest, temp)
#Merging individual vector polygons to form grid in final folder
arcpy.management.Merge(arcpy.ListFeatureClasses(), final_dest + "\\" + grid, "", "ADD_SOURCE_INFO")
#Adding grid id field
arcpy.management.AddField(final_dest + "\\" + grid, "Grid", "TEXT")
#Populating the file name as the grid name
arcpy.management.CalculateField(final_dest + "\\" + grid, "Grid", 'os.path.basename(!MERGE_SRC!)','PYTHON3')
#Removing the .shp file extension from the grid id
arcpy.management.CalculateField(final_dest + "\\" + grid, "Grid", 'replace($feature.Grid,".shp","")','ARCADE')
#Deleting the MERGE_SRC field from table
arcpy.management.DeleteField(final_dest + "\\" + grid,"MERGE_SRC")
#Deleting Temp_Folder
shutil.rmtree(os.path.join(final_dest, temp))
print("Grid created")
print(str(((time.time() - start_time))//60) + " minutes run time")
```

PYTHON APPLICATIONS

ESRI WANTS TO HELP

- Even when you struggle there are tons of resources by ESRI to help you use Python
- Search ESRI's tools online for their descriptions or use the help button on the tool pane within ArcGIS Pro
  - Under Parameters there is a Python tab that will explain what you need and examples below
- You can also open the tool, set the parameters and copy an autogenerated Python code
- You can run these in a stand-alone IDE or use the built-in Jupyter Notebook within your APRX





OTHER RESOURCES

- W3Schools
- GeeksforGeeks
- GIS Stack Exchange
- ESRI Community Boards
- ChatGPT (AI)

SOME NIFTY TRICKS

- os.getlogin() – recognizes the users login name
  - Great for scripts to allow for all users to run the scripts
  - Assign the user name as a variable
- os.getcwd() – recognizes the folder the script is running in
- r"**file\path**" – the r at the beginning of a file path allows the file path string to be read as raw
  - Otherwise all \ will need to be \\
  - \ is an escape clause in Python
- f"**string {variable}**" the f at the beginning of a string allows for ease of formatting with involving variables with strings
- User input – you can ask for user input with the input function

```
path = r"C:\Users\Kelsey.Troyer\Downloads"

name = os.getlogin()

better_path = f"C:\\Users\\{name}\\Downloads"
```

```
answer = input("What  r favorite color? ")
print("Your favorite  is " + answer)
```

```
answer = input("What is your favorite color? ")
print(f"Your favorite color is {answer}")
```

```
What is your favorite color? Pink
Your favorite color is Pink
```

INCREASE EFFICIENCY

- For loops and lists are a great way to run a repetitive task through different parameters
- Lists are formatted with [] brackets
- You can also create lists of pairs [(),()]
- [#] indicates the index you would like to use

```
likes = ["dogs", "sunshine", "GIS"]
for x in likes:
    print(f"I like {x}")
```

```
I like dogs
I like sunshine
I like GIS
```

```
facts = [("sky","blue"),("frog","green"),("apple","red")]
for f in facts:
    print(f"The {f[0]} is {f[1]}")
```

```
The sky is blue
The frog is green
The apple is red
```

MULTI-PROCESSING

- Run multiple tasks simultaneously
- Export your atlases in a fraction of the time
- Define your function
- Use a dunder conditional block to trigger your processes

- To use any ArcGIS tools in Python your must import the arcpy library
  - Do this for any libraries or modules you may need
- listLayouts({wild_card}) is a method to use all or a specified layout
  - Can be used for exporting
  - Wildcards are used to call out a known value
    - Ex: I have layouts Wat100, Wat200, & Wat300
    - "*Wat100", "*100", "*1*", and "Wat1*" would all call the Wat100 layout

```
import arcpy
```

```
import arcpy, os, time
```

wild_card

Limits the results returned. If a value is not specified, all values are returned. The wildcard is not case sensitive.

Symbol	Description	Example
*	Represents zero or more characters.	Te* finds Tennessee and Texas.

ARCPY

Map - ESRI's documentation on managing layers

Layout - ESRI's documentation on exporting layouts

Map Series – ESRI's documentation on exporting map series

ArcGISProject - ESRI's documentation on managing workflows

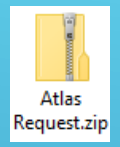
```
import arcpy, os, sys, time, shutil, multiprocessing

start_time = time.time()
print("Starting to export all Potable")
xlist = [(r"V:\Atlases\_WATER-POTABLE\Potable Water.aprx","*Wat100",r"V:\Atlases\_WATER-POTABLE"),\
        (r"V:\Atlases\_WATER-POTABLE\Potable Water.aprx","*Wat200",r"V:\Atlases\_WATER-POTABLE"),\
        (r"V:\Atlases\_WATER-POTABLE\Potable Water.aprx","*Wat200B&W",r"V:\Atlases\_WATER-POTABLE"),\
        (r"V:\Atlases\_WATER-POTABLE\Potable Water.aprx","*Wat300",r"V:\Atlases\_WATER-POTABLE")]
def export(aprx,layout,outpath):
    Atlas = arcpy.mp.ArcGISProject(aprx)
    for Atlaslyt in Atlas.listLayouts(layout):
        if not Atlaslyt.mapSeries is None:
            ms = Atlaslyt.mapSeries
            if ms.enabled:
                ms.exportToPDF(os.path.join(outpath, f"{Atlaslyt.name}\\", f"{Atlaslyt.name}"),\
                               multiple_files="PDF_MULTIPLE_FILES_PAGE_NAME", resolution= 300,\
                               image_compression="DEFLATE", layers_attributes="NONE", georef_info="FALSE")
        print(Atlaslyt.name + " Complete")
if __name__ == "__main__":
    processes = [multiprocessing.Process(target=export, args=(x[0],x[1],x[2])) for x in xlist]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
print("All Potable atlases exported")

print(str(((time.time() - start_time))//60) + " minutes run time")
```

```
import arcpy

aprx = r"The path and file name of your aprx, don't forget the file extension"
outpath = r"The path to where you want to export the map to and the namp of the export"
Atlas = arcpy.mp.ArcGISProject(aprx)
for Atlaslyt in Atlas.listLayouts():
    if not Atlaslyt.mapSeries is None:
        ms = Atlaslyt.mapSeries
        if ms.enabled:
            ms.exportToPDF(outpath, multiple_files="PDF_MULTIPLE_FILES_PAGE_NAME", resolution= 300)
```

EXPORTING A MAP SERIES

EXAMPLES

- Yearly inventory reports
- Grid creation from raster tiles
- Atlas page requests
- Data requests
- Data reports
- Copying files



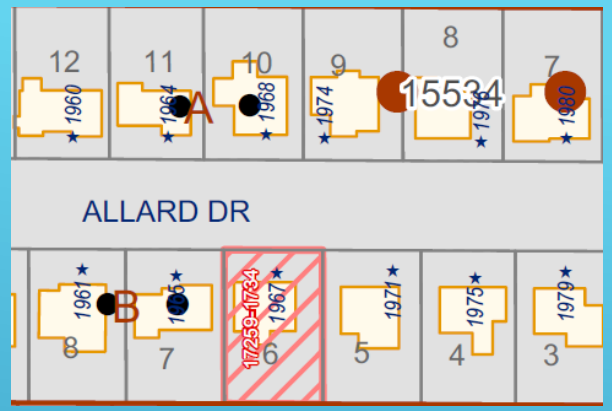

```
Miles of City Sanitary Sewer Gravity Mains: 372.29
Miles of City Sanitary Sewer Force Mains: 38.53
Sanitary Sewer Laterals: 31664
City Owned Sanitary Sewer Lift Station: 76
City Owned Sanitary Sewer Manhole: 8352
City Owned Sanitary Sewer Cleanout: 413
City Owned Sanitary Sewer Treatment Facilities: 3
Miles of City Stormwater Mains: 157.57
Miles of City Stormwater Underdrains: 117.16
Miles of City Stormwater Ditches: 39.98
Miles of City Stormwater Swales: 8.44
Miles of City Stormwater Creeks: 23.79
Miles of City Stormwater Lakes: 6.56
City Owned Stormwater Ponds: 184
City Owned Stormwater Sewer Baffle Boxes: 36
City Owned Stormwater Sewer Caps: 575
City Owned Stormwater Sewer Cleanouts: 1192
City Owned Stormwater Sewer Control Structures: 173
City Owned Stormwater Sewer Flared End Section: 108
City Owned Stormwater Sewer Grate Inlets: 1826
City Owned Stormwater Sewer Head Walls: 287
City Owned Stormwater Sewer Inlets: 4029
City Owned Stormwater Sewer Junction Boxes: 224
City Owned Stormwater Sewer Mitered End Section: 332
City Owned Stormwater Sewer Manholes: 1637
City Owned Stormwater Sewer Open Pipe: 156
City Owned Stormwater Sewer Percolation Inlets: 12
City Owned Stormwater Sewer Pump Station(s): 1
City Owned Stormwater Sewer Tideflex Valve: 18
Miles of City Potable Water Mains: 587.67
City Owned Potable Water Valves: 8779
City Owned Potable Water Left Turn Valves: 4
City Owned Potable Water Valves (Not Found): 97
City Owned Potable Water Inserta Valves: 23
City Owned Potable Water Air Release Valves: 13
City Owned Potable Water DCV: 633
```

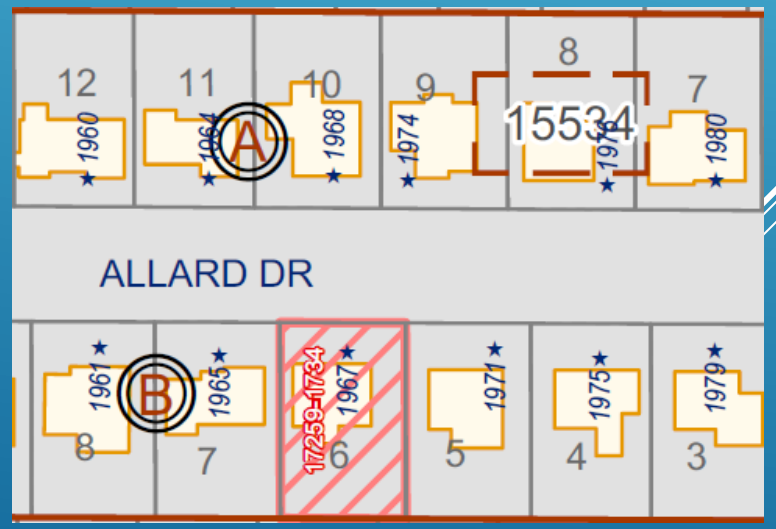
```python
import arcpy, os, datetime, time
start_time = time.time()
now = datetime.datetime.now()
five_hour = 16-int(now.strftime("%H"))
five_mins = 60-int(now.strftime("%M"))
total_secs = ((five_hour*60)+five_mins)*60
arcpy.env.workspace = r"V:\Atlases\PAO Data\PAO_Pinellas.gdb"
arcpy.env.overwriteOutput = True
#Determining user's name to use the correct file path to their downloads folder
name = os.getlogin()
#Determining if the script needs to sleep
print("If someone is using the data it will cause the clipping portion of the script to fail")
print("You can delay the clipping portion of this script until after hours to help with this")
sleeping = input("Would you like to delay part of this process until after 5pm? (Y/N) ").upper()
#Using today's date to find correct file
try:
    arcpy.env.workspace = r"C:\Users\\" + name + "\Downloads\PcpaCadastre_" + now.strftime("%d%b%y") + ".gdb"
    for feature in arcpy.ListFeatureClasses():
        arcpy.management.Copy(feature, r"V:\Atlases\PAO Data\PAO_Pinellas.gdb\\" + feature, "FeatureClass")
        print("Copied " + feature)
#If file is from a previous day, user can manually specify the file's date
except:
    wrksp = input("Provide the date associated with the gdb's name (i.e. 06Sep22) ")
    arcpy.env.workspace = r"C:\Users\\" + name + "\Downloads\PcpaCadastre_" + wrksp + ".gdb"
    for feature in arcpy.ListFeatureClasses():
        arcpy.management.Copy(feature, r"V:\Atlases\PAO Data\PAO_Pinellas.gdb\\" + feature, "FeatureClass")
        print("Copied " + feature)

#Copying county wide PCPA (PAO) data to V:\Atlases\PAO Data\PAO_Pinellas.gdb
for feature in arcpy.ListFeatureClasses():
    arcpy.management.Copy(feature, r"V:\Atlases\PAO Data\PAO_Pinellas.gdb\\" + feature, "FeatureClass")
    print("Copied " + feature)
arcpy.env.workspace = r"V:\Atlases\PAO Data\PAO_Pinellas.gdb"

#Shape of buffered service area used to clip data
service = r"V:\Atlases\PAO Data\Service_Area_Buffered.shp"

#If no sleep, clip to service area now
if sleeping == "Y":
    print(f"Sleeping for {five_hour} hour(s) and {five_mins} min(s)...")
    time.sleep(total_secs)
else:
    pass

#Starting the clipping
print("Starting clipping to Clearwater Service Area")
arcpy.env.workspace = r"V:\Atlases\PAO Data\PAO_Pinellas.gdb"
service = r"V:\Atlases\PAO Data\Service_Area_Buffered.shp"
for clip in arcpy.ListFeatureClasses():
    arcpy.analysis.Clip(clip, service, r"V:\Atlases\PAO Data\PAO_Basemap.gdb\\" + clip)
    print(clip + " clipped")

print("PAO Basemap data updated")
print(str(((time.time() - start_time))//60) + " minutes run time")
```

```
Which atlas would you like to export? Only choose one. (Wat, Rcw, Stm, San): Wat
What scale do you need? Only choose one. (100, 200, 300, all): 300
Enter the grid number(s) you need with a space between pages or 'all': 271a 271b 200a
Starting the export of the Wat300 atlas
You input grids ['271A', '271B', '200A']
Wat300 export complete
0.0 minutes run time
>>>
```

PYTHON LIMITATIONS

- We are currently in ArcGIS 3.1.x
  - Cannot specify “Convert character marker symbols to polygon” or “Simulate overprint”
    - Pinellas Font
- According to community boards they are trying to fix this by 3.2





THANK YOU FOR LISTENING