Python: Top 5 Tips and Tricks

Jack Horton
Python is everywhere in ArcGIS
ArcMap

```python
>>> import arcpy
path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf100"
arcpy.Buffer_analysis(input, output, "100 Feet")
```
ArcGIS Pro

```python
>>> import arcpy
path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf100"
arcpy.Buffer_analysis(input, output, "100 Feet")
```
ArcGIS Online or Enterprise (Portal)

Move existing user content to a new user

This sample illustrates how to "move" a portal user's account to a new user account. This is accomplished by creating a new user account, assigning ownership/membership of this new user to all the applicable groups, and then assigning the old user's content to the new user account while maintaining folder structure under 'My Contents'.

For some customers, this is a useful utility when they have used one type of Identity store, e.g. Built-in Users, and then decided to switch to a different Identity provider, such as SAML or IWA. In these situations, it is highly likely new userids will be created as new user accounts get created. This Jupyter Notebook is an example of how to use the Python API to take a user's content and migrate it to a new userid while maintaining all group membership and content (including folders in My Content).

In [ ]: from arcgis.gis import *
Create a connection to the portal. In this case, we will exercise the verify_cert option to not validate the SSL certificate (True by default).

In [ ]: gis = GISS(“portal url”, “username”, “password”, verify_cert=False)
Establish variables for the current userid that is being transitioned and for the new userid to be created (e.g. a new Single Sign-on username).

In [ ]: orig_userid = ”georger”
new_userid = ”gso@esri.com”
You can run it in the application
You can run it in an IDE (Interactive Development Environment)
You can run it standalone

C:\Users\jack361>C:\Python27\ArcGIS10.5\python.exe "D:\jhorton\conferences\2017\UC17\Tech Workshop\Python - Tips and Tricks for getting started\buff01.py"
done

C:\Users\jack361>
Or you can schedule it to run automatically
A Python script is just a text file
Python automates the things you do by hand, such as

- Mapping
- Data management
- Analysis
- Publishing web services
- Administering your portal
- And much more

In this workshop, we will focus on simple analysis running geoprocessing tools.
The Help is full of code samples

**Code sample**

**Buffer example 1 (Python window)**
The following Python window script demonstrates how to use the Buffer tool.

```python
import arcpy
arcpy.env.workspace = "C:/data"
arcpy.Buffer_analysis("roads", "C:/output/majorrsBuffered", "100 Feet", "FULL", "ROUND", "LIST", "Distance")
```

...Copy and paste them to get started
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
In this script, we put a long pathname in a variable called path

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```
Variables store data

Then we use it to make two new variables containing the pathnames to our data

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```
Variables store data

And we finally use these two variables as input to the Buffer geoprocessing tool

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```
Let’s run our script

```python
import arcpy
path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python - Tips and Tricks for getting started/bu..."
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print("done")
```
It worked! – It made the buffer, then printed “done”
Let's run it again – it fails, because the output feature class already exists.
Let's delete it if it already exists

- Search for “Exists” and get another code sample from the help

```python
import arcpy

# Set the current workspace
arcpy.env.workspace = "c:/base/data.gdb"

# Check for existence of data before deleting
if arcpy.Exists("roadbuffer"):
    arcpy.Delete_management("roadbuffer")
```
Let’s delete it if it already exists

• Search for “Exists” and get another code sample from the help

```python
import arcpy

# Set the current workspace
arcpy.env.workspace = "c:/base/data.gdb"

# Check for existence of data before deleting
if arcpy.Exists("roadbuffer"):
    arcpy.Delete_management("roadbuffer")
```
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

input = path + "/railroads"
output = path + "/railbuf"

arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

if arcpy.Exists(output):
    arcpy.Delete_management(output)

arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
Let's follow the pattern in the sample and put it in our code

Note that we don’t use quotes, because our quoted string is in the variable called `output`

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

if arcpy.Exists(output):
    arcpy.Delete_management(output)

arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```
Now it works, because it deletes the output feature class if it already exists.
Let's buffer all the feature classes in the geodatabase

Search the help for ListFeatureClasses

```python
import os
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
arcpy.CopyFeatures_management(
    fc, os.path.join("c:/base/output.gdb", os.path.splitext(fc)[0]))
```
Lets buffer all the feature classes in the geodatabase
And copy the sample

```python
import os
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
arcpy.CopyFeatures_management(fc, os.path.join("c:/base/output.gdb", os.path.splitext(fc)[0]))
```
This sample is kind of fancy – lets remove what we don’t need

```python
import os
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
    arcpy.CopyFeatures_management(
        fc, os.path.join("c:/base/output.gdb", os.path.splitext(fc)[0]))
```
This sample is kind of fancy – lets remove what we don’t need

This example uses a Standard Python Module called OS to manipulate pathnames

```python
import os
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
    arcpy.CopyFeatures_management(
        fc, os.path.join("c:/base/output.gdb",
            os.path.splitext(fc)[0])))```
This sample is kind of fancy – let's just pull out what we need

Let's remove that part...

```python
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
```
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
    print (fc)
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# Copy shapefiles to a file geodatabase
for fc in featureclasses:
    print (fc)
This sample is kind of fancy – lets just pull out what we need… also update the comments to keep them relevant to what we are doing

import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of feature classes.
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of feature classes.
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
We must set the **workspace environment**

That is how `ListFeatureClasses` knows what workspace to list out

```python
import arcpy

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the `ListFeatureClasses` function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
```
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
...and merge in our existing script

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
if arcpy.Exists(output):
    arcpy.Delete_management(output)
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
```
...and merge in our existing script

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

if arcpy.Exists(output):
    arcpy.Delete_management(output)

arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
We will comment out the lines that run the Buffer tool for now

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "//railroads"
output = path + "//railbuf"
# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = "c:/base"
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
Let's **test** what we have so far.
It prints out all of the feature classes

```python
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print(fc)
```
Now that we got this far, let's start setting it up to run the Buffer tool.

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
# if arcpy.Exists(output):
#    arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
```
The input and output variables are currently hard-coded to our test data

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"

# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
```
Set up the input and output variables to use each feature class in the loop

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
# if arcpy.Exists(output):
#   arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
```
... and remove the original hard-coded input and output variables

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
```
... and remove the original hard-coded input and output variables

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# if arcpy.Exists(output):
    # arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
Print out our new variable values

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# if arcpy.Exists(output):
#    arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    print (input + " - " + output)
    print ()
```
Let's confirm that it correctly sets our input and output variables.
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    print (input + " - " + output)
    print ()
Get ready to copy and paste the buffer statements into the for loop

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

if arcpy.Exists(output):
    arcpy.Delete_management(output)

arcpy.Buffer_analysis(input, output, "100 Feet")

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    print (input + " - " + output)
```

print ()
Get ready to copy and paste the buffer statements into the for loop

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Print out each feature class name
for fc in featureclasses:
    print(fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    print(input + "  " + output)
print()
```
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    print (input + " - " + output)
    print ()
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print (fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    print (input + " - " + output)
    print ()
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print(fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print(fc)

    input = path + "/" + fc

    output = path + "/" + fc + "buff"
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Print out each feature class name
for fc in featureclasses:
    print ("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"

    # if arcpy.Exists(output):
    #     arcpy.Delete_management(output)

    arcpy.Buffer_analysis(input, output, "100 Feet")

    # print ("done")
Remove the comment characters (#) to activate these lines

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class
for fc in featureclasses:
    print ("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"

if arcpy.Exists(output):
    arcpy.Delete_management(output)

arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```
And indent them properly so they run inside the for loop

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"

    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    arcpy.Buffer_analysis(input, output, "100 Feet")

print("done")
```
Un-indent the final print() statement, so it only prints “done” at the end

```python
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Buffer each feature class
for fc in featureclasses:
    print ("Buffering " + fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
        arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")```
Challenge: How do you keep it from buffering a buffered feature class?

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    arcpy.Buffer_analysis(input, output, "100 Feet")

print("done")
```
Challenge: How do you keep it from buffering a buffered feature class?

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class
for fc in featureclasses:
    print ("Buffering " + fc)
    input  = path + "\" + fc
    output = path + "\" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
        arcpy.Buffer_analysis(input, output, "100 Feet")
    print ("done")
```
Challenge: How do you keep it from buffering a buffered feature class?

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    print ("Buffering " + fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"

    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```
Challenge: How do you keep it from buffering a buffered feature class?

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print("done")
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        arcpy.Buffer_analysis(input, output, "100 Feet")

print("done")
The print() is now misleading – we are not buffering every feature class

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    print("Buffering " + fc)
    input = path + "/" + fc
    output = path + "/" + fc + "buff"

    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    if input[-4:] <> "buff":
        arcpy.Buffer_analysis(input, output, "100 Feet")

print("done")
```
The print() is now misleading – we are not buffering every feature class

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:

    print ("Buffering " + fc)
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"

    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    if input[-4:] <> "buff":

        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```
Put it inside the if so it only prints if we actually buffer the feature class

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```
Put it inside the if so it only prints if we actually buffer the feature class

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in “buff”
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
```
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)

    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("done")
How did [-4:] specify the last four characters?

if input[-4:] <> "buff“:
    basically says:
    
    *if the last 4 characters of input are not “buff”*
How did [-4:] specify the last four characters?

```python
if input[-4:] <> "buff":
    basically says:

    *if the last 4 characters of input are not “buff”*
```

Strings are indexed between the letters like this:

```
0  1  2  3  4  5  6  7
a b c d e f g
-7 -6 -5 -4 -3 -2 -1
```
How did [-4:] specify the last four characters?

if input[-4:] <> "buff":
  basically says:
  **if the last 4 characters of input are not “buff”**

Strings are indexed between the letters like this:

```
  0 1 2 3 4 5 6 7
a b c d e f g
```

```
  -7 -6 -5 -4 -3 -2 -1
```

```
"abcdefg"[0:3] returns “abc”
"abcdefg"[:3]  also returns “abc”
"abcdefg"[3:6] returns “def”
"abcdefg"[-4:-1] also returns “def”
"abcdefg"[3:7] returns “defg”
"abcdefg"[3:]  also returns “defg”
"abcdefg"[-4:7] returns “defg”
"abcdefg"[-4:]  also returns “defg”
```
How did [-4:] specify the last four characters?

```python
if input[-4:] <> "buff":
    basically says:
    `if the last 4 characters of input are not "buff"`
```

Strings are indexed between the letters like this:

```
 0 1 2 3 4 5 6 7
\-----------------
<table>
<thead>
<tr>
<th>abcdefg</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcdefg</td>
</tr>
<tr>
<td>---------</td>
</tr>
</tbody>
</table>
```
How did [-4:] specify the last four characters?

if input[-4:] <> "buff":
    basically says:

    if the last 4 characters of input are not “buff”

Strings are indexed between the letters like this:

```
0 1 2 3 4 5 6 7
a b c d e f g
-7 -6 -5 -4 -3 -2 -1
```

- "abcdefg"[0:3] returns “abc”
- "abcdefg"[:3] also returns “abc”
- "abcdefg"[3:6] returns “def”
- "abcdefg"[-4:-1] also returns “def”
- "abcdefg"[3:7] returns “defg”
- "abcdefg"[3:] also returns “defg”
- "abcdefg"[-4:7] returns “defg”
- "abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

```python
if input[-4:] <> "buff":
    basically says:
    if the last 4 characters of input are not “buff”
```

Strings are indexed between the letters like this:

```
  0 1 2 3 4 5 6 7
-7 -6 -5 -4 -3 -2 -1
```

![Indexing diagram]

- "abcdefg"[0:3] returns “abc”
- "abcdefg"[:3] also returns “abc”
- "abcdefg"[3:6] returns “def”
- "abcdefg"[-4:-1] also returns “def”
- "abcdefg"[3:7] returns “defg”
- "abcdefg"[3:] also returns “defg”
- "abcdefg"[-4:7] returns “defg”
- "abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

```python
if input[-4:] <> "buff":
    basically says:
    *if the last 4 characters of input are not “buff”*
```

Strings are indexed between the letters like this:

```
0 1 2 3 4 5 6 7
a b c d e f g
```

```
-7 -6 -5 -4 -3 -2 -1
```

- "abcdefg"[0:3] returns “abc”
- "abcdefg"[:3] also returns “abc”
- "abcdefg"[3:6] returns “def”
- "abcdefg"[-4:-1] also returns “def”
- "abcdefg"[3:7] returns “defg”
- "abcdefg"[3:] also returns “defg”
- "abcdefg"[-4:7] returns “defg”
- "abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

if input[-4:] <> "buff":
    basically says:

    if the last 4 characters of input are not “buff”

Strings are indexed between the letters like this:

0 1 2 3 4 5 6 7

a b c d e f g

-7 -6 -5 -4 -3 -2 -1

"abcdefg"[0:3] returns “abc”
"abcdefg"[:3] also returns “abc”
"abcdefg"[3:6] returns “def”
"abcdefg"[-4:-1] also returns “def”
"abcdefg"[3:7] returns “defg”
"abcdefg"[3:] also returns “defg”
"abcdefg"[-4:7] returns “defg”
"abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

if input[-4:] <> "buff":
    basically says:

    *if the last 4 characters of input are not “buff”*

Strings are indexed between the letters like this:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td></td>
</tr>
</tbody>
</table>
```

"abcdefg"[0:3] returns “abc”
"abcdefg"[:3] also returns “abc”
"abcdefg"[3:6] returns “def”
"abcdefg"[-4:-1] also returns “def”
"abcdefg"[3:7] returns “defg”
"abcdefg"[3:] also returns “defg”
"abcdefg"[-4:-7] returns “defg”
"abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

if input[-4:] <> "buff":
basically says:

\textit{if the last 4 characters of input are not “buff”}

Strings are indexed between the letters like this:

\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 6 & \quad 7 \\
\text{abcdefg} \\
-7 & \quad -6 & \quad -5 & \quad -4 & \quad -3 & \quad -2 & \quad -1 \\
\end{align*}

"abcdefg"[0:3] returns “abc”
"abcdefg"[3:] also returns “abc”
"abcdefg"[3:6] returns “def”
"abcdefg"[-4:-1] also returns “def”
"abcdefg"[3:7] returns “defg”
"abcdefg"[3:] also returns “defg”
"abcdefg"[3:-7] returns “defg”
"abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

if input[-4:] <> "buff"
basically says:

*if the last 4 characters of input are not “buff”*

Strings are indexed between the letters like this:

```
 0 1 2 3 4 5 6 7
a b c d e f g
-7 -6 -5 -4 -3 -2 -1
```

- "abcdefg"[0:3] returns “abc”
- "abcdefg"[:3] also returns “abc”
- "abcdefg"[3:6] returns “def”
- "abcdefg"[-4:-1] also returns “def”
- "abcdefg"[3:7] returns “defg”
- "abcdefg"[3:] also returns “defg”
- "abcdefg"[-4:7] returns “defg”
- "abcdefg"[-4:] also returns “defg”
How did [-4:] specify the last four characters?

```python
if input[-4:] <> "buff":
    basically says:
    *if the last 4 characters of input are not “buff”*
```

Strings are indexed between the letters like this:

```
  0 1 2 3 4 5 6 7
 a b c d e f g
```

```plaintext
"abcdefg"[0:3] returns “abc”
"abcdefg"[:3] also returns “abc”
"abcdefg"[3:6] returns “def”
"abcdefg"[-4:-1] also returns “def”
"abcdefg"[3:7] returns “defg”
"abcdefg"[3:] also returns “defg”
"abcdefg"[-4:7] returns “defg”
"abcdefg"[-4:] also returns “defg”
```
That is how we use **if** to only run buffer if the last 4 characters are not “buff”

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
        if input[-4:] <> "buff":
            print ("Buffering " + fc)
            arcpy.Buffer_analysis(input, output, "100 Feet")
    print ("Done")
```
Almost done…

```python
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input_ = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input_[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input_, output, "100 Feet")
print ("Done")
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()
# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")
print ("Done")
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

arcpy.env.workspace = path

featureclasses = arcpy.ListFeatureClasses()

for fc in featureclasses:
    input = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        print("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")
print("Done")
Our completed script!

# BufferAll.py
# Jack Horton   June 30, 2018
# Buffers all the feature classes in a workspace, adding “buff” to each output
#
import arcpy

path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"

# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path

# Use the ListFeatureClasses function to return a list of feature classes
featureclasses = arcpy.ListFeatureClasses()

# Buffer each feature class that does not end in "buff"
for fc in featureclasses:
    input  = path + "/" + fc
    output = path + "/" + fc + "buff"
    if arcpy.Exists(output):
        arcpy.Delete_management(output)
    if input[-4:] <> "buff":
        print ("Buffering " + fc)
        arcpy.Buffer_analysis(input, output, "100 Feet")

print ("Done")
The same script runs in ArcMap or ArcGIS Pro
The same script runs in ArcMap or ArcGIS Pro

* scripts that manipulate the user interface will, of course, be different
Top 5 Python tips and tricks
Top 5 Python tips and tricks

1. Start with sample code
   from the help system or an internet search
Top 5 Python tips and tricks

1. Start with sample code from the help system or an internet search

2. Use meaningful names for your variables and keep it simple
Top 5 Python tips and tricks

1. Start with sample code
   from the help system or an internet search
2. Use meaningful names for your variables
   and keep it simple
3. **Python is Case Sensitive**
   so use copy/paste to reduce spelling errors
Top 5 Python tips and tricks

1. Start with sample code from the help system or an internet search
2. Use meaningful names for your variables and keep it simple
3. Python is Case Sensitive so use copy/paste to reduce spelling errors
4. Put parentheses after function calls, even if they take no parameters
   Example: arcpy.ListFeatureClasses()
Top 5 Python tips and tricks

1. Start with sample code
   from the help system or an internet search
2. Use meaningful names for your variables
   and keep it simple
3. Python is Case Sensitive
   so use copy/paste to reduce spelling errors
4. Put parentheses after function calls, even if they take no parameters
   Example: arcpy.ListFeatureClasses()
5. Use print() functions or a debugger to figure out what your script is really doing
   and test as you go
Top 5 Python tips and tricks

1. Start with sample code from the help system or an internet search
2. Use meaningful names for your variables and keep it simple
3. Python is Case Sensitive so use copy/paste to reduce spelling errors
4. Put parentheses after function calls, even if they take no parameters Example: arcpy.ListFeatureClasses()
5. Use print() functions or a debugger to figure out what your script is really doing and test as you go
Download the Esri Events app and find your event.

Select the session you attended.

Scroll down to find the feedback section.

Complete answers and select “Submit”.

Please Take Our Survey on the App.