



# Python: Top 5 Tips and Tricks

Jack Horton

**GIS  
INSPIRING  
WHAT'S  
NEXT**

# Python is everywhere in ArcGIS

The image is a collage of several screenshots demonstrating Python's role in ArcGIS. At the top left is the ArcMap interface with a map of a city street grid. Below it is the PyCharm IDE showing a Python script named 'buffer.py' with the following code:

```
import arcpy
path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples"
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```

To the right, a browser window displays a page titled "Move existing user content to a new user" from developers.arcgis.com, with a "Download the samples" button. Below the browser is the Windows Task Scheduler, showing a task named "Buffer a feature class" with a trigger set for 8:02 PM on 6/30/2017. The task's actions are configured to run the Python script. At the bottom left, a terminal window shows the execution of the script: `>>> import arcpy`, `path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples"`, `input = path + "/railroads"`, `output = path + "/railbuf"`, `arcpy.Buffer_analysis(input, output, "100 Feet")`, and `print ("done")`. A notification at the bottom center states "Platform and Plugin Updates: PyCharm Community Edition is ready to update."

# ArcMap

The screenshot displays the ArcMap interface with a map of a city area. The map shows a network of streets and a railroad line. A purple buffer zone is applied around the railroad line. The interface includes a menu bar (File, Edit, View, Bookmarks, Insert, Selection, Geoprocessing, Customize, Windows, Help), a toolbar, and a Table of Contents pane on the left. The Table of Contents lists the following layers:

- Layers
  - Railroads
  - 
  - railbuf100
  - Climate
    - <all other values>
    - TYPE
      - Coastal
      - Maritime
      - Transitional
  - Basemap
    - World Topographic Map

The Python console at the bottom shows the following code:

```
>>> 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")
>>>
```

At the bottom of the Python console, the coordinates 6282052.96 1839029.715 Feet are displayed.



# ArcGIS Pro

The screenshot displays the ArcGIS Pro interface. The main map area shows a street map with a purple buffer zone around a railroad line. The interface includes a ribbon menu at the top with tabs for Project, Map, Inse, Ana, View, Edit, Ima, Sha, Appearance, Label, and Data. The View tab is active, showing tools for Global, Local, and Map. The Table of Contents on the left lists layers: Railroads, railbuf100, Climate (with sub-layers for Coastal, Maritime, and Transitional), and Basemap (with World Topographic Map). The Python console at the bottom left shows the following code:

```
>>> 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")
>>>
```

The Python console at the bottom right shows the same code with syntax highlighting:

```
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")
```

The map area shows a scale of 1:15,635 and coordinates 117.1433227°W 32.7005765°N. The status bar at the bottom indicates coordinates 6282052.96 1839029.715 Feet.

# ArcGIS Online or Enterprise (Portal)

The screenshot shows the ArcMap interface with a map of Petco Park. The Table of Contents on the left lists layers: Railroads, railbuf100, Climate, Basemap, and World Topographic Map. The Python console at the bottom shows the following code:

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

The screenshot shows a Jupyter Notebook titled "Move existing user content to a new user". The notebook content includes:

## 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 reassigning the old user's content to the new user connect 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 = GIS("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 = "georged"
new_userid = "gsd@esri.com"
```



# You can run it in the application

Table of Contents

- Layers
  - Railroads
  - railbuf100
  - Climate
    - <all other values>
    - TYPE
      - Coastal
      - Maritime
      - Transitional
  - Basemap
    - World Topographic Map

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")
>>>
```

6282052.96 1839029.715 Feet

ArcGIS Pro - Python... Feature Layer

Project Map Inse Ana View Edit Ima Sha Appearance Label Data JackHorton (Esri Ed Services Instructors)

Global Local Map Convert Link Views Catalog Contents Python Tasks Reviewer Rules Workflow Manager

View Link Windows Th... Ani... Scene Navigation

Catalog Map

1:15,635 117.1433227°W 32.7005765°N Selected Features: 0

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")
```

# You can run it in an IDE (Interactive Development Environment)

The image displays two IDE windows. The background window is PyCharm, showing a project named 'untitled [PyCharm Project]' with a file explorer on the left containing files like 'ArcGIS\_API.py', 'ArcGIS\_API\_Jupyter.ipynb', 'ArcMap\_Mapping\_Mo...', 'buffer\_pro.py', 'Pro\_mp\_Module.py', and 'TestPyVersion.py'. The main editor shows a Python script with the following code:

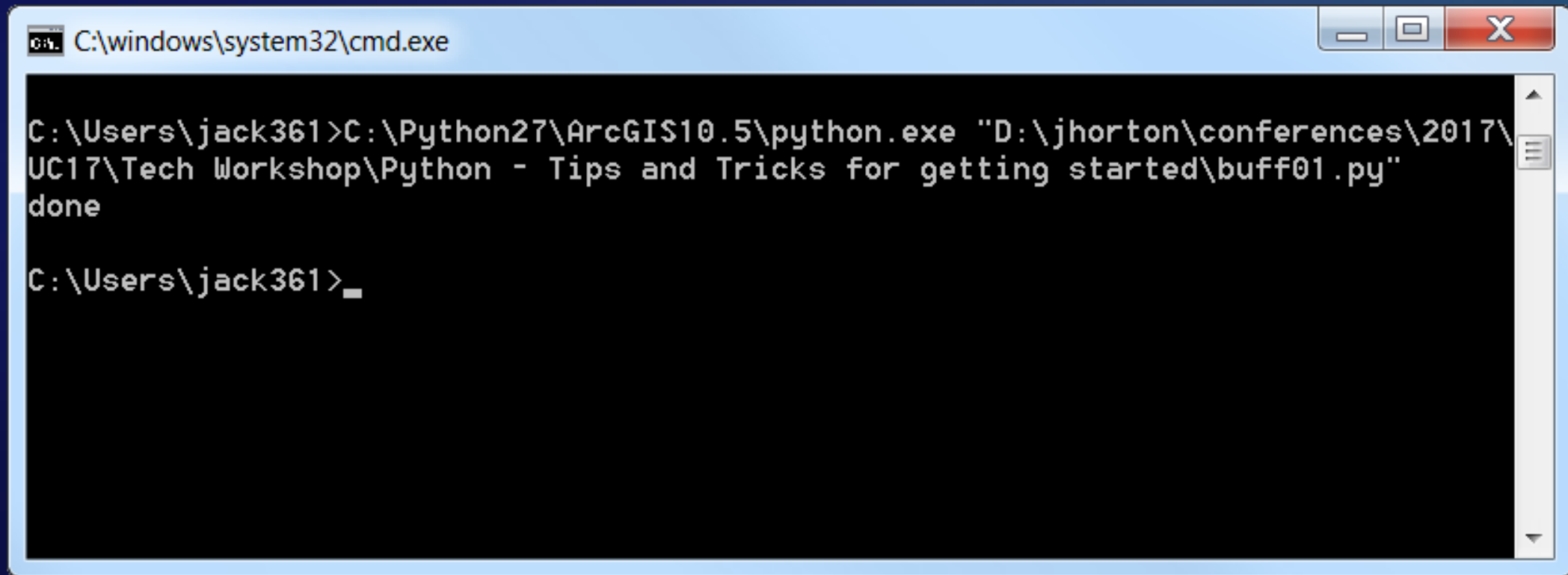
```
1 import arcpy
2 path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/100_Feet_Buffer/ArcMap_Mapping_Mo..."
3 input = path + "/railroads"
4 output = path + "/railbuf"
5 arcpy.Buffer_analysis(input, output, "100 Feet")
6 print ("done")
```

The foreground window is PyScripter, titled 'PyScripter - D:\jhorton\conferences\2017\UC17\Tech Workshop\Python - Tips and Tricks for getting started\buff.py'. It shows the same Python code in its editor. Below the editor is a 'Python Interpreter' console with the following output:

```
*** Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32. ***
*** Remote Python engine is active ***
>>>
```

The PyScripter window also includes a 'Project Explorer' on the left showing 'Untitled', 'Files', and 'Run Configurations'. The bottom status bar of PyScripter shows '1: 1' and 'Insert'.

## You can run it standalone



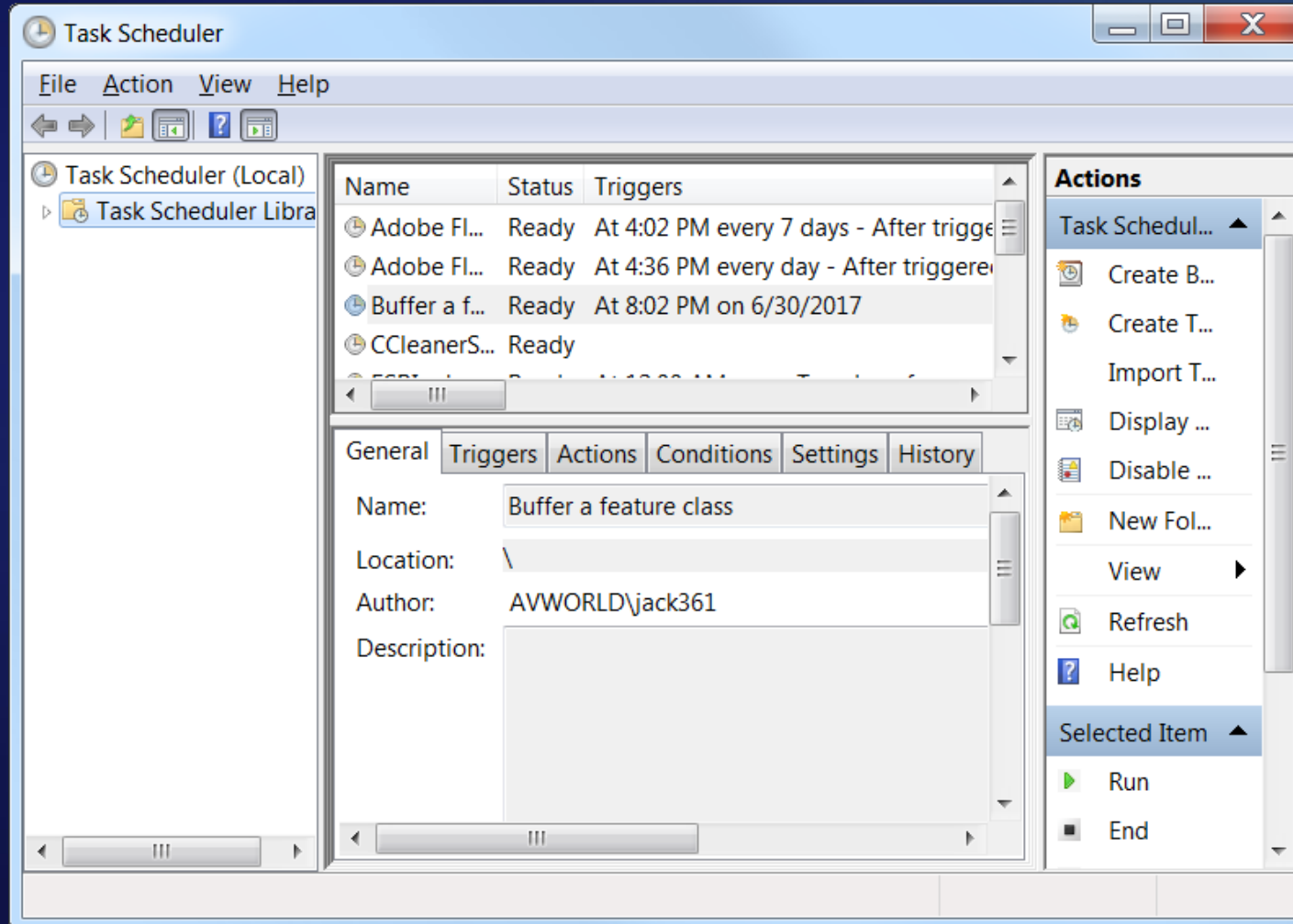
```
C:\windows\system32\cmd.exe

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

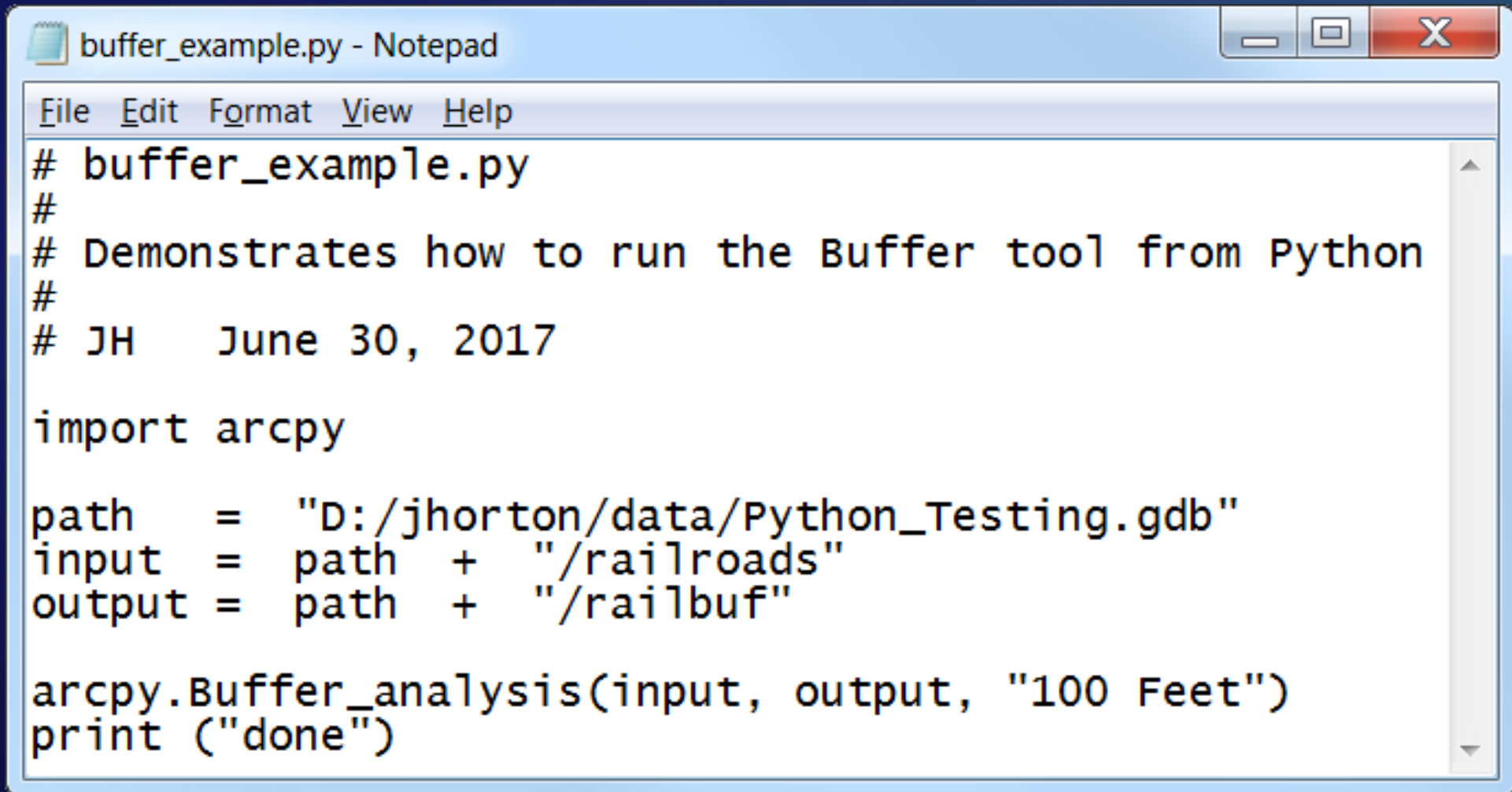
C:\Users\jack361>_
```



Or you can schedule it to run automatically



## A Python script is just a text file



```
buffer_example.py - Notepad
File Edit Format View Help
# buffer_example.py
#
# Demonstrates how to run the Buffer tool from Python
#
# JH    June 30, 2017

import arcpy

path      = "D:/jhorton/data/Python_Testing.gdb"
input     = path + "/railroads"
output    = path + "/railbuf"

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

## **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.

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

...Copy and paste them to get started

# arcpy contains the Esri Python code in ArcGIS Pro and ArcMap

We import the arcpy module so we can get to things like the buffer tool

```
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

In this script, we put a long pathname in a variable called path

```
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

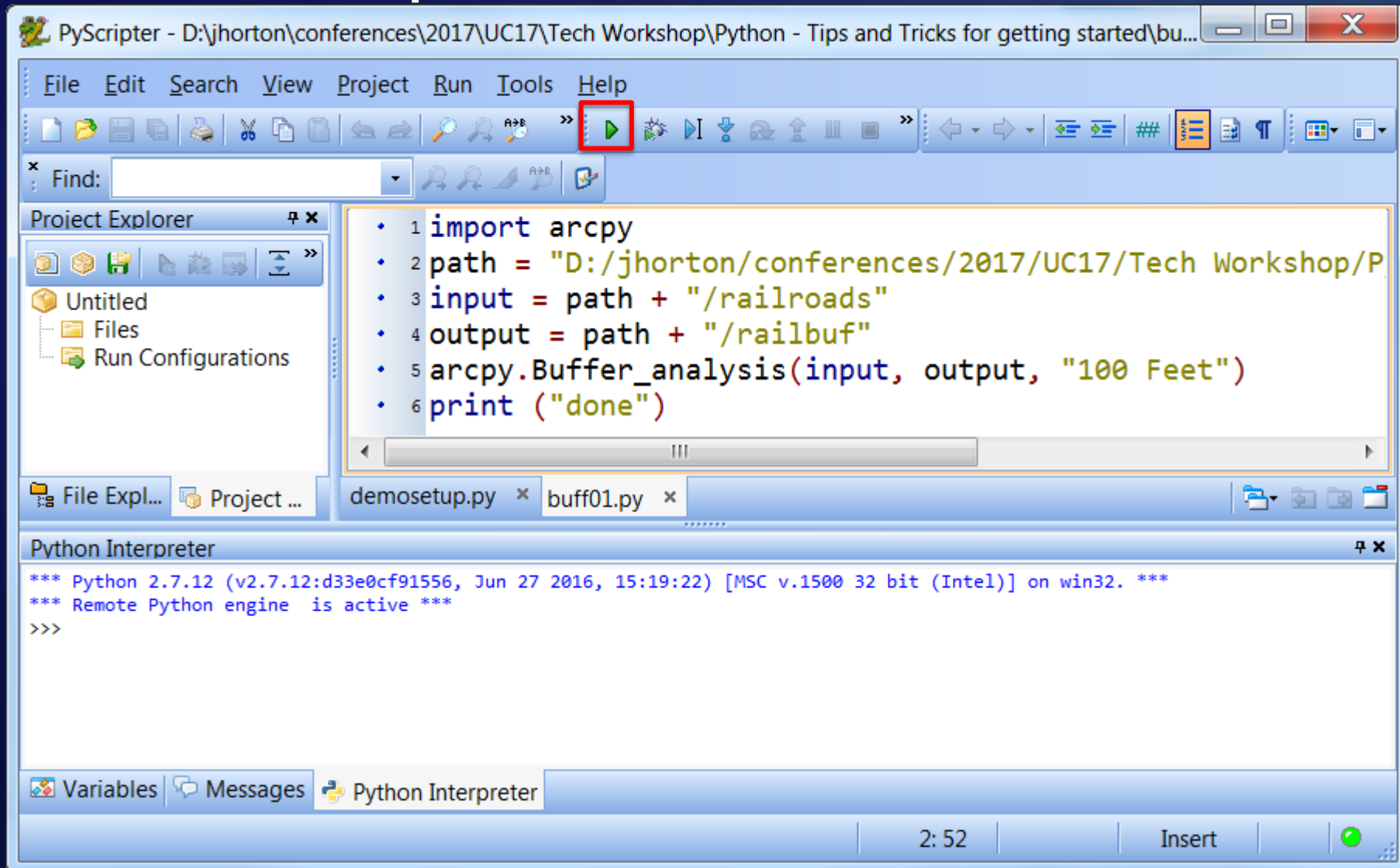
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
arcpy.Buffer_analysis(input, output, "100 Feet")
print ("done")
```

## Variables store data

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

```
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



The screenshot shows the PyScripter IDE interface. The main editor window displays a Python script with the following code:

```
1 import arcpy
2 path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/P
3 input = path + "/railroads"
4 output = path + "/railbuf"
5 arcpy.Buffer_analysis(input, output, "100 Feet")
6 print ("done")
```

The Python Interpreter window at the bottom shows the following output:

```
*** Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32. ***
*** Remote Python engine is active ***
>>>
```

The status bar at the bottom indicates the time is 2:52 and the mode is Insert.



# It worked! – It made the buffer, then printed “done”

The screenshot shows the PyScripter IDE interface. The main editor window displays a Python script with the following code:

```
1 import arcpy
2 path = "D:/jhorton/conferences/2017/UC17/Tech Workshop/P
3 input = path + "/railroads"
4 output = path + "/railbuf"
5 arcpy.Buffer_analysis(input, output, "100 Feet")
6 print ("done")
```

The line `print ("done")` is highlighted with a red box. The Python Interpreter window at the bottom shows the execution output:

```
*** Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32. ***
>>>
*** Remote Interpreter Reinitialized ***
>>>
done
>>>
```

The output `done` is also highlighted with a red box. The status bar at the bottom of the IDE shows the time as 2:52 and the mode as Insert.

Lets run it again – it fails, because the output feature class already exists

The screenshot shows the PyScripter application window. An error dialog box is open in the foreground, displaying the following text:

```
ExecuteError: Failed to execute. Parameters are not valid.  
ERROR 000725: Output Feature Class:  
D:\jhorton\conferences\2017\UC17\Tech  
Workshop\Python_Examples\Python_Examples.gdb\railb...  
already exists.  
Failed to execute (Buffer).
```

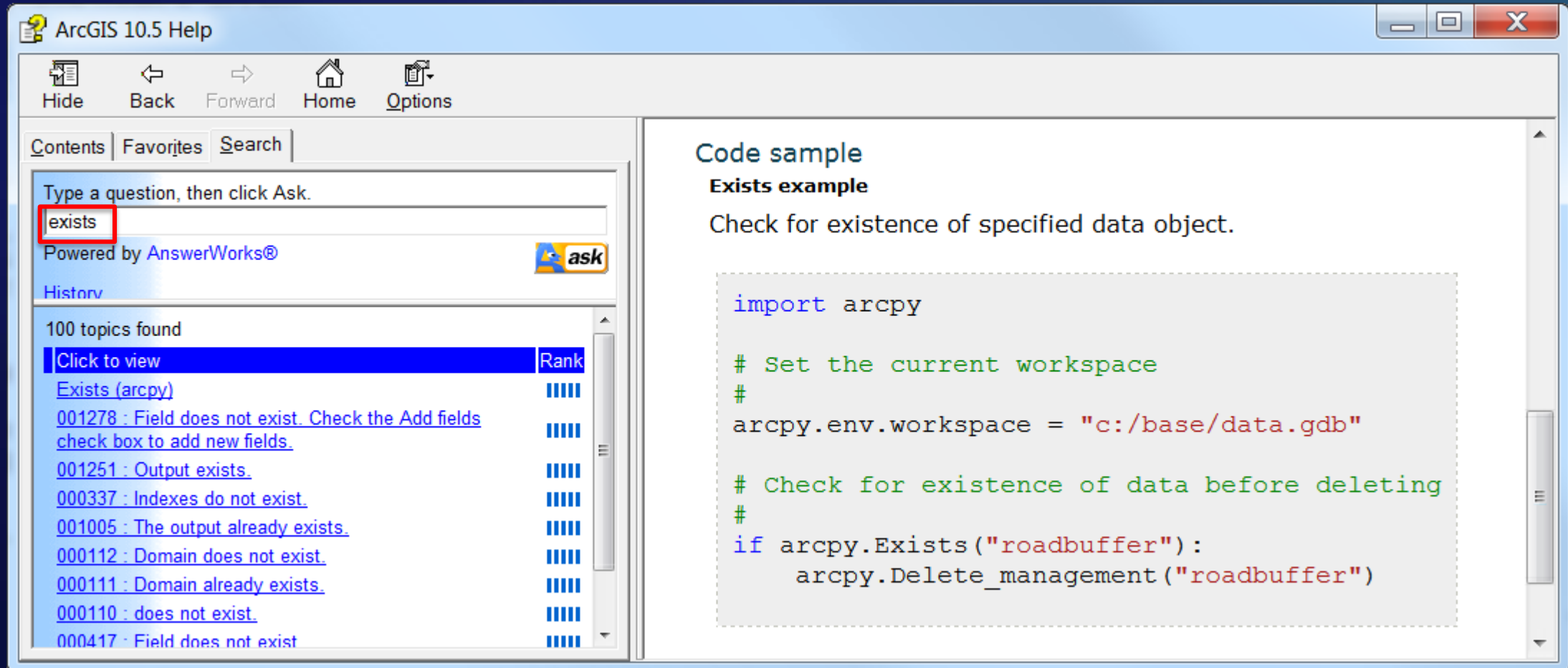
The error message is highlighted with a red border. The background shows the PyScripter interface with a code editor containing Python code. The Messages pane at the bottom displays a traceback:

Message	File Name	Line	Positi..
Traceback			
<module>	D:\jhorto...	5	
Buffer	C:\Progra...	767	
ExecuteError: Failed to execute. Parameters are not valid.ERROR 000725: Output Feature Clas...			

The Messages pane also shows the error message: "ExecuteError: Failed to execute. Parameters are not valid.ERROR 000725: Output Feature Clas...". The bottom status bar indicates "Running".

# Let's delete it if it already exists

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



The screenshot shows the ArcGIS 10.5 Help interface. The search bar contains the text "exists", which is highlighted with a red box. Below the search bar, there is a list of search results. The first result, "Exists (arcpy)", is highlighted in blue. To the right of the search results, there is a "Code sample" section titled "Exists example". This section contains a Python code snippet that demonstrates how to check for the existence of a data object before deleting it.

**Code sample**  
**Exists example**  
Check for existence of specified data object.

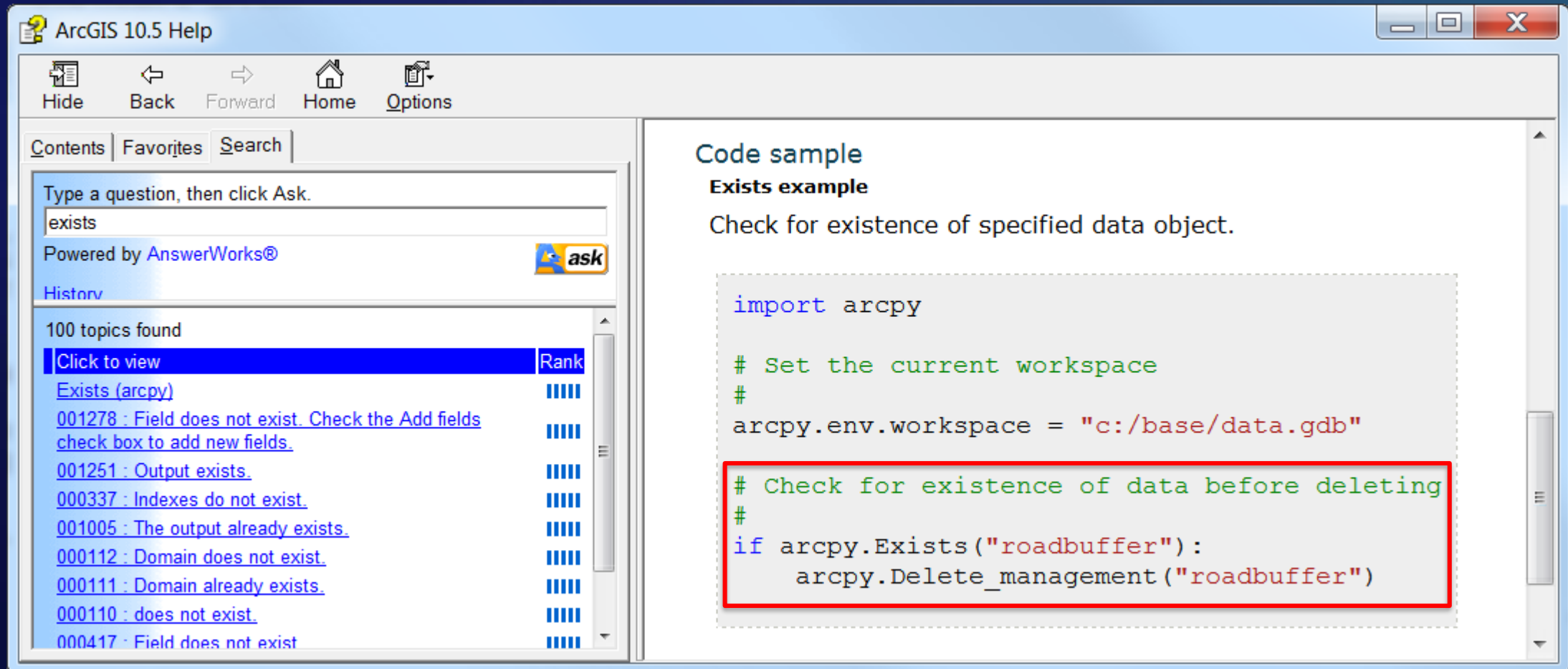
```
import arcpy

# 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



The screenshot shows the ArcGIS 10.5 Help interface. On the left, a search bar contains the text "exists". Below the search bar, a list of 100 topics is displayed. The first topic, "Exists (arcpy)", is highlighted in blue. To the right of the search results, a code sample titled "Exists example" is shown. The code sample includes a red box highlighting the following lines:

```
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")
```

Lets follow the pattern in the sample and put it in our code

```
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")
```



Lets follow the pattern in the sample and put it in our code

```
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")
```

Lets follow the pattern in the sample and put it in our code

```
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")
```

# Lets 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**

```
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

The screenshot shows the PyScripter IDE interface. The main editor displays a Python script with the following code:

```
• 3 input = path + "/railroads"  
• 4 output = path + "/railbuf"  
• 5 if arcpy.Exists(output):  
• 6     arcpy.Delete_management(output)  
• 7 arcpy.Buffer_analysis(input, output, "100 Feet")  
• 8 print ("done")
```

The `print ("done")` line is highlighted with a red box. Below the editor, the Python Interpreter window shows the following output:

```
>>>  
*** Remote Interpreter Reinitialized ***  
>>>  
done  
>>>
```

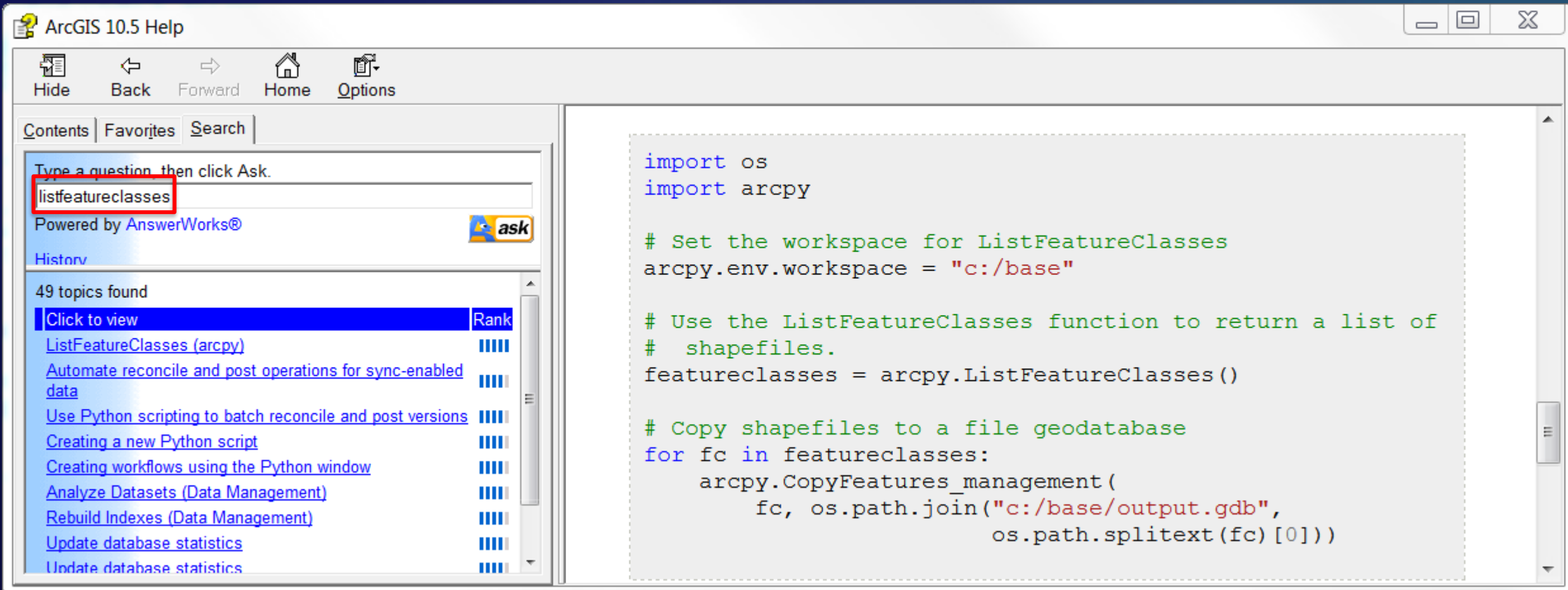
The `done` output is highlighted with a red box. The error message from the previous run is also visible:

```
.gdb\railbuf already exists.  
Failed to execute (Buffer).
```

The error message is highlighted with a red box. The text **Last time** is written in red next to the error message. The text **This time** is written in red next to the `done` output. The status bar at the bottom shows the time 8:15 and the Insert key.

# Lets buffer all the feature classes in the geodatabase

## Search the help for ListFeatureClasses



The screenshot shows the ArcGIS 10.5 Help window. The search bar contains the text 'listfeatureclasses', which is highlighted with a red box. Below the search bar, the text 'Powered by AnswerWorks®' and an 'ask' button are visible. The search results list 49 topics found, with 'ListFeatureClasses (arcpy)' selected and highlighted in blue. The main content area displays a Python code snippet for buffering feature classes in a geodatabase.

```
import os
import arcpy

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

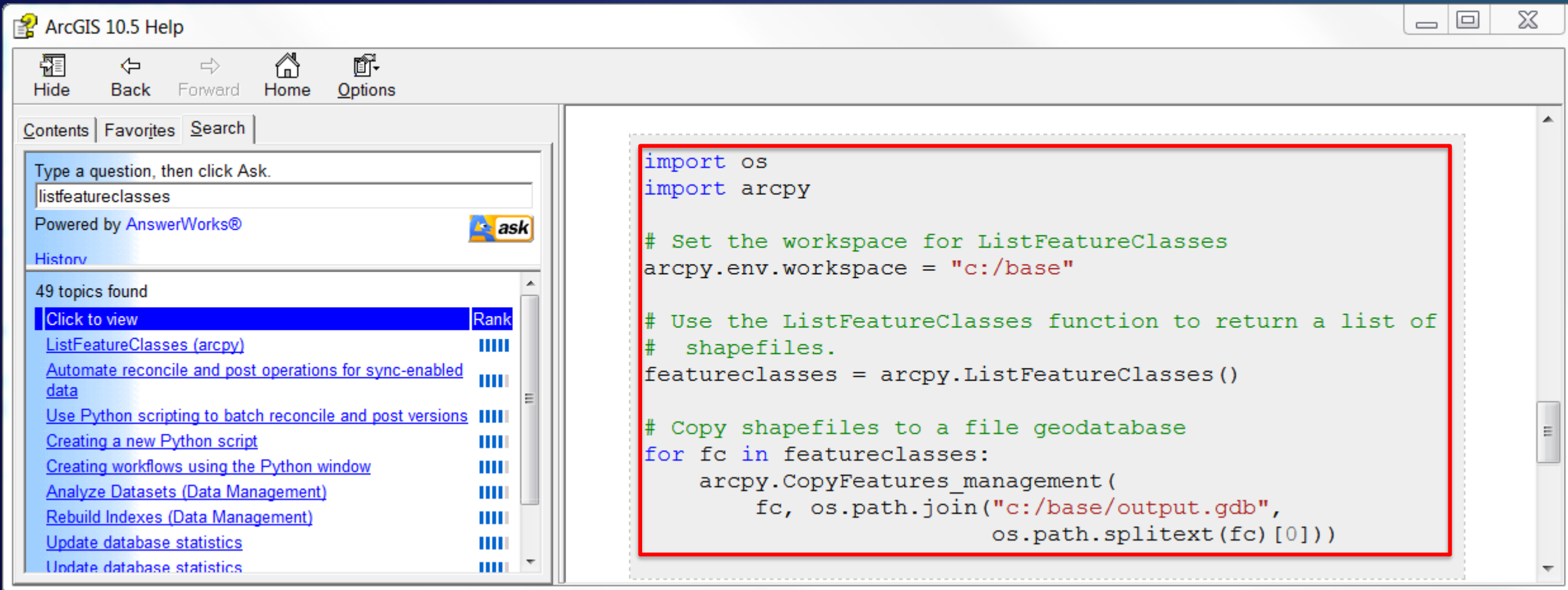
# Use the ListFeatureClasses function to return a list of
# shapefiles.
featureclasses = arcpy.ListFeatureClasses()

# 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



The screenshot shows the ArcGIS 10.5 Help window. The search bar contains the text 'listfeatureclasses'. The search results list 49 topics found, with 'ListFeatureClasses (arcpy)' highlighted. To the right, a Python script is displayed, which is highlighted with a red border. The script performs the following actions:

- Imports the `os` and `arcpy` modules.
- Sets the workspace for `ListFeatureClasses` to `c:/base`.
- Uses the `ListFeatureClasses` function to return a list of shapefiles.
- Iterates over each feature class in the list and copies it to a file geodatabase.

```
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

```
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

```
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 just pull out what we need**

**Lets remove that part...**

```
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:
```

# This sample is kind of fancy – lets just pull out what we need

... and just print out the feature classes for now

```
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
# 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)
```

## We must set the **workspace environment**

```
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

```
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)
```

## Lets remove some blank lines

```
# 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)
```



## Lets remove some blank lines

```
# 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

```
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

```
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

```
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)
```

## This script is still trying to use the data from the sample

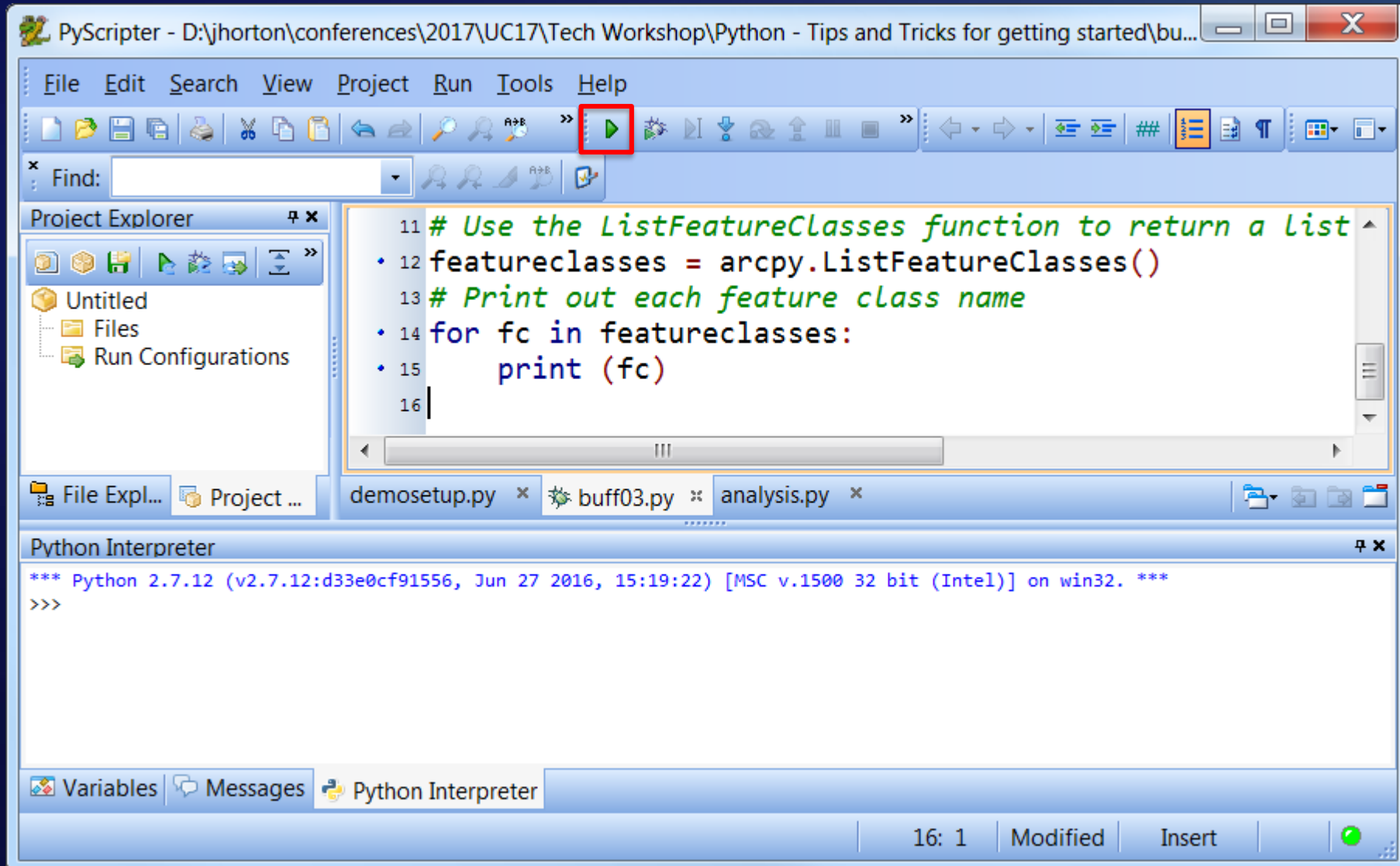
```
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
input = path + "/railroads"
output = path + "/railbuf"
# if arcpy.Exists(output):
#     arcpy.Delete_management(output)
# arcpy.Buffer_analysis(input, output, "100 Feet")
# print ("done")
# Set 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)
```

## Substitute our path variable for the workspace

```
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)
```



# Lets **test** what we have so far



# It prints out all of the feature classes

The screenshot shows the PyScripter IDE interface. The main editor window displays the following Python code:

```
• 10 arcpy.env.workspace = path
• 11 # Use the ListFeatureClasses function to return a list
• 12 featureclasses = arcpy.ListFeatureClasses()
• 13 # Print out each feature class name
• 14 for fc in featureclasses:
• 15     print (fc)
```

The code block for the for loop (lines 14 and 15) is highlighted with a red box. Below the code editor, the Python Interpreter window shows the output of the code:

```
MajorAttractions
Ocean
Railroads
Streams
railbuf100
railbuf
>>>
```

The output text is also highlighted with a red box. The status bar at the bottom of the IDE shows the current cursor position as 14: 1 and the mode as Insert.

## Now that we got this far, lets start setting it up to run the Buffer tool

```
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

```
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

```
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

```
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

```
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

```
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

```
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 ()
```

Lets **confirm** that it correctly sets our input and output variables

The screenshot shows the PyScripter IDE interface. The main editor window displays the following Python code:

```
• 14 for fc in featureclasses:  
• 15     print (fc)  
• 16     input = path + "/" + fc  
• 17     output = path + "/" + fc + "buff"  
• 18     print (input + " - " + output)  
• 19     print
```

The code is executed, and the Python Interpreter window at the bottom shows the following output:

```
Ocean  
D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb/Ocean - D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb/Oceanbuff  
  
Railroads  
D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb/Railroads - D:/jhorton/conferences/2017/UC17/Tech Workshop/Python_Examples/Python_Examples.gdb/Railroadsbuff
```

The IDE also shows a Project Explorer on the left with 'Untitled', 'Files', and 'Run Configurations'. The bottom status bar includes 'Variables', 'Messages', and 'Python Interpreter' tabs.

## Now alter the print statements and run the buffer

```
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

```
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

```
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 ()
```

## Get ready to copy and paste the buffer statements into the for loop

```
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 ()
```

## Remove the print() statements we used for testing

```
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 ()
```

## Remove the print() statements we used for testing

```
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"
```

## Alter the remaining print statement

```
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"
```

## Alter the remaining print statement

```
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"
```



## Paste in the lines that run the buffer tool

```
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

```
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

```
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

```
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?

```
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?

```
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?

```
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?

```
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")
```

## Answer: Don't run the buffer if the input ends in "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 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

```
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

```
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

```
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

```
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

```
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?

```
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?

**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?

**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:

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:

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:

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:

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"

That is how we use **if** to only run buffer if the last 4 characters are not “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 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...

```
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...

```
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")
```

## Add some header comments

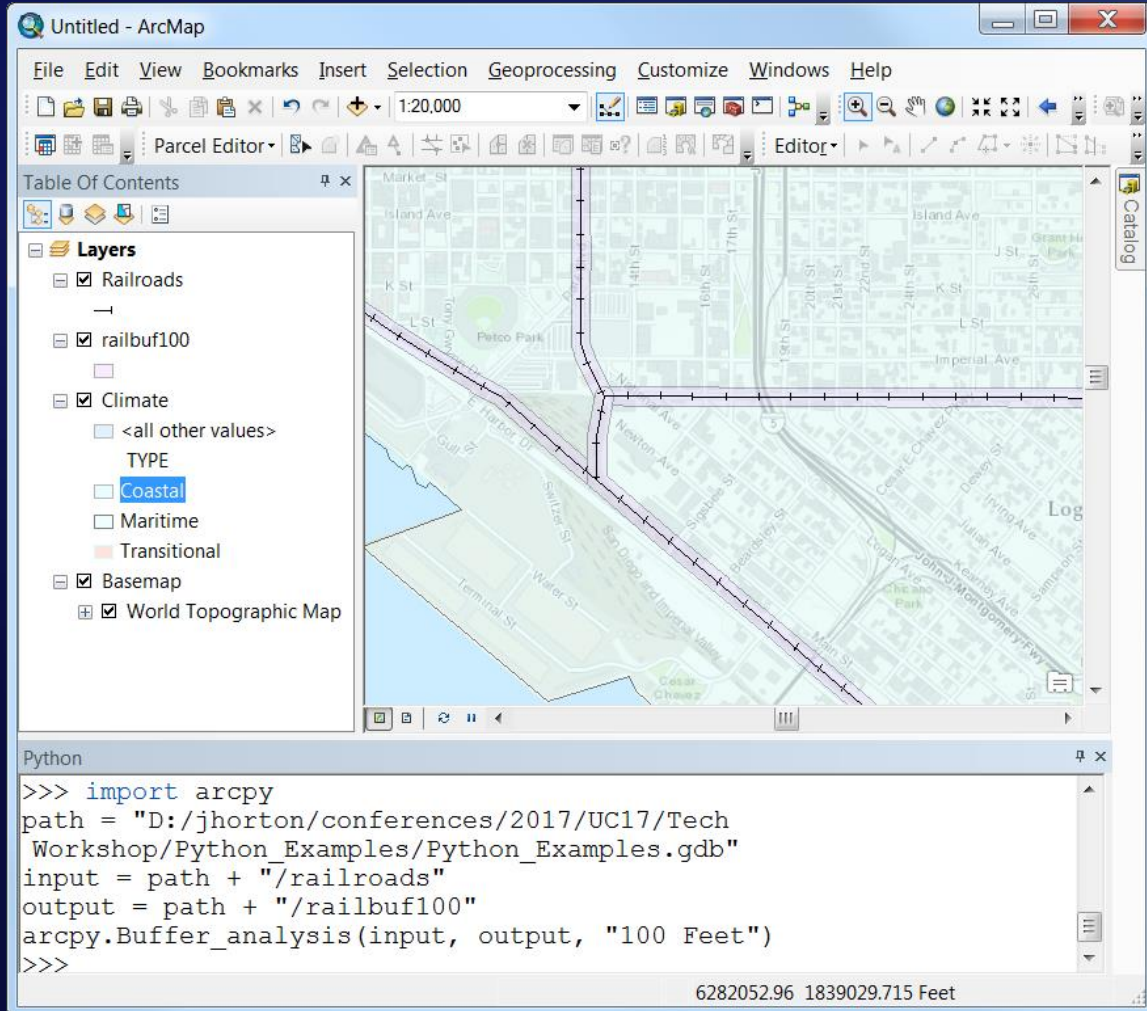
```
# BufferAll.py
# Jack Horton   June 30, 2018
# Buffers all the feature classes in a workspace, adding "buff" to each output
#
import arcpy
path = "D:/jhorton/conferences/2018/UC18/TechWorkshop/Python_Examples/Python_Examples.gdb"
# Set the workspace for ListFeatureClasses
arcpy.env.workspace = path
# 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")
```

## 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



Untitled - ArcMap

File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help

1:20,000

Parcel Editor Editor

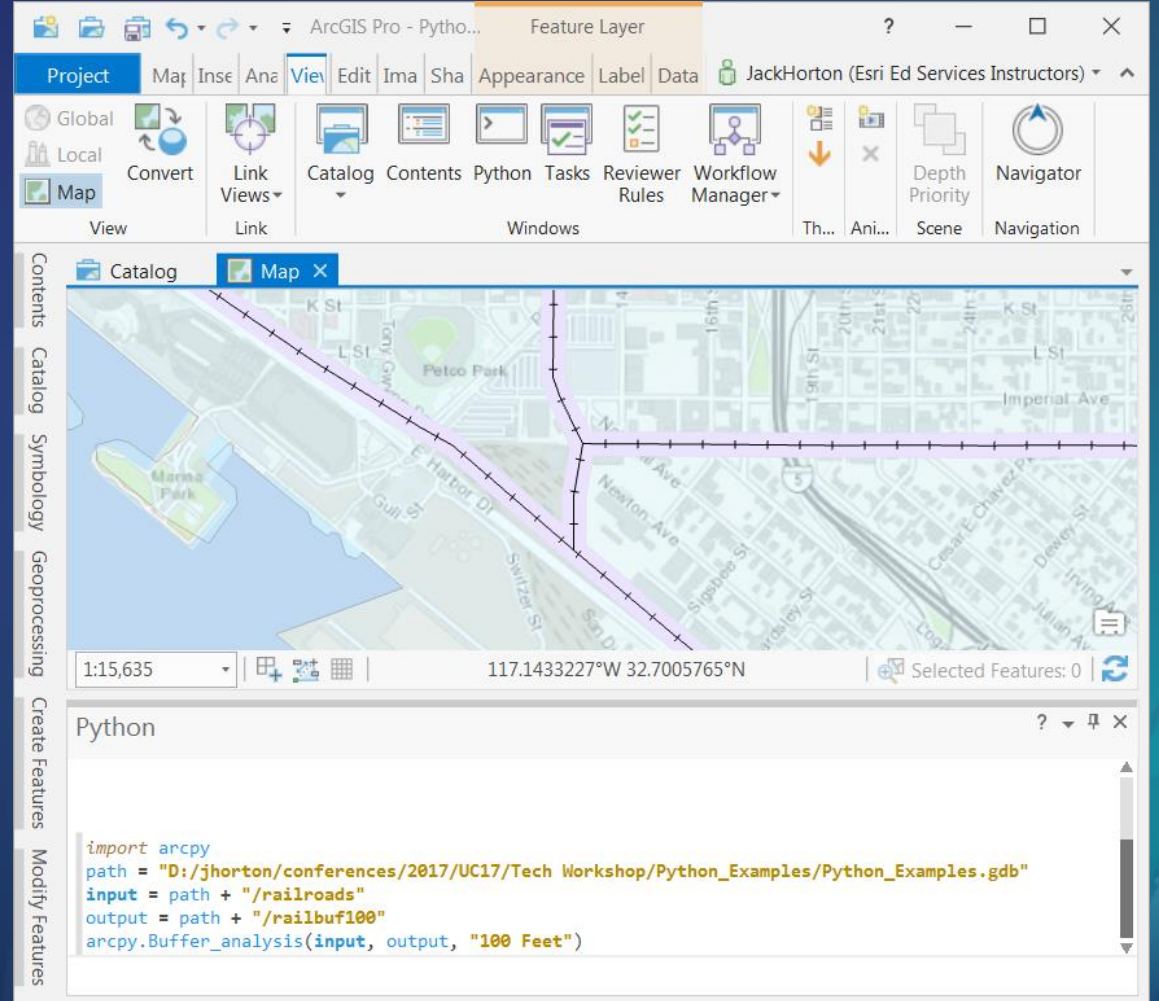
Table Of Contents

- Layers
  - Railroads
  - railbuf100
  - Climate
    - <all other values>
    - TYPE
      - Coastal
      - Maritime
      - Transitional
  - Basemap
    - World Topographic Map

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")
>>>
```

6282052.96 1839029.715 Feet



ArcGIS Pro - Python

Feature Layer

Project Map Inse Ana View Edit Ima Sha Appearance Label Data JackHorton (Esri Ed Services Instructors)

Global Local Convert Link Views Catalog Contents Python Tasks Reviewer Workflow Manager

View Link Windows Th... Ani... Scene Navigation

Catalog Map

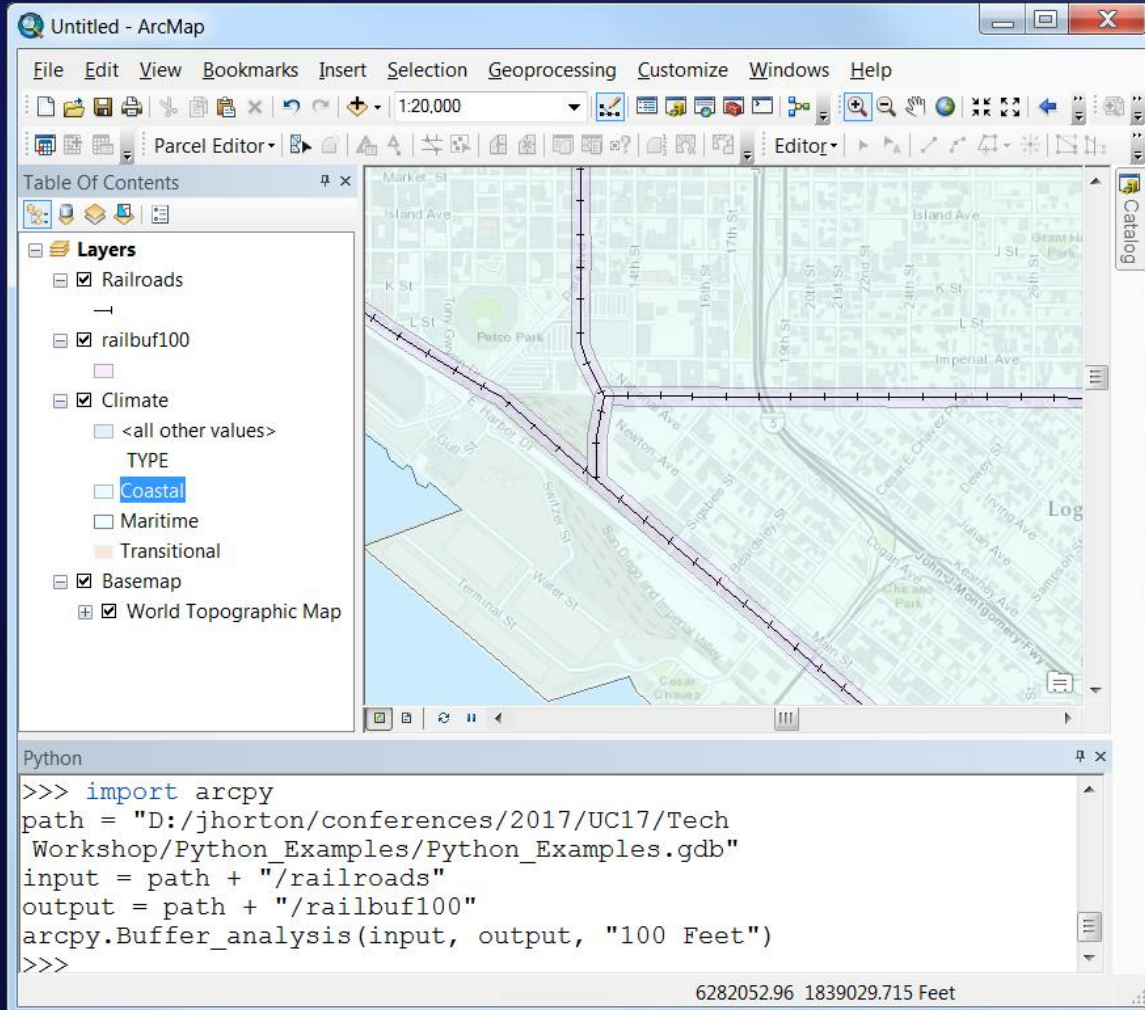
1:15,635 117.1433227°W 32.7005765°N Selected Features: 0

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")
```

# The same script runs in ArcMap or ArcGIS Pro

\* scripts that manipulate the user interface will, of course, be different



Untitled - ArcMap

File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help

1:20,000

Parcel Editor Editor

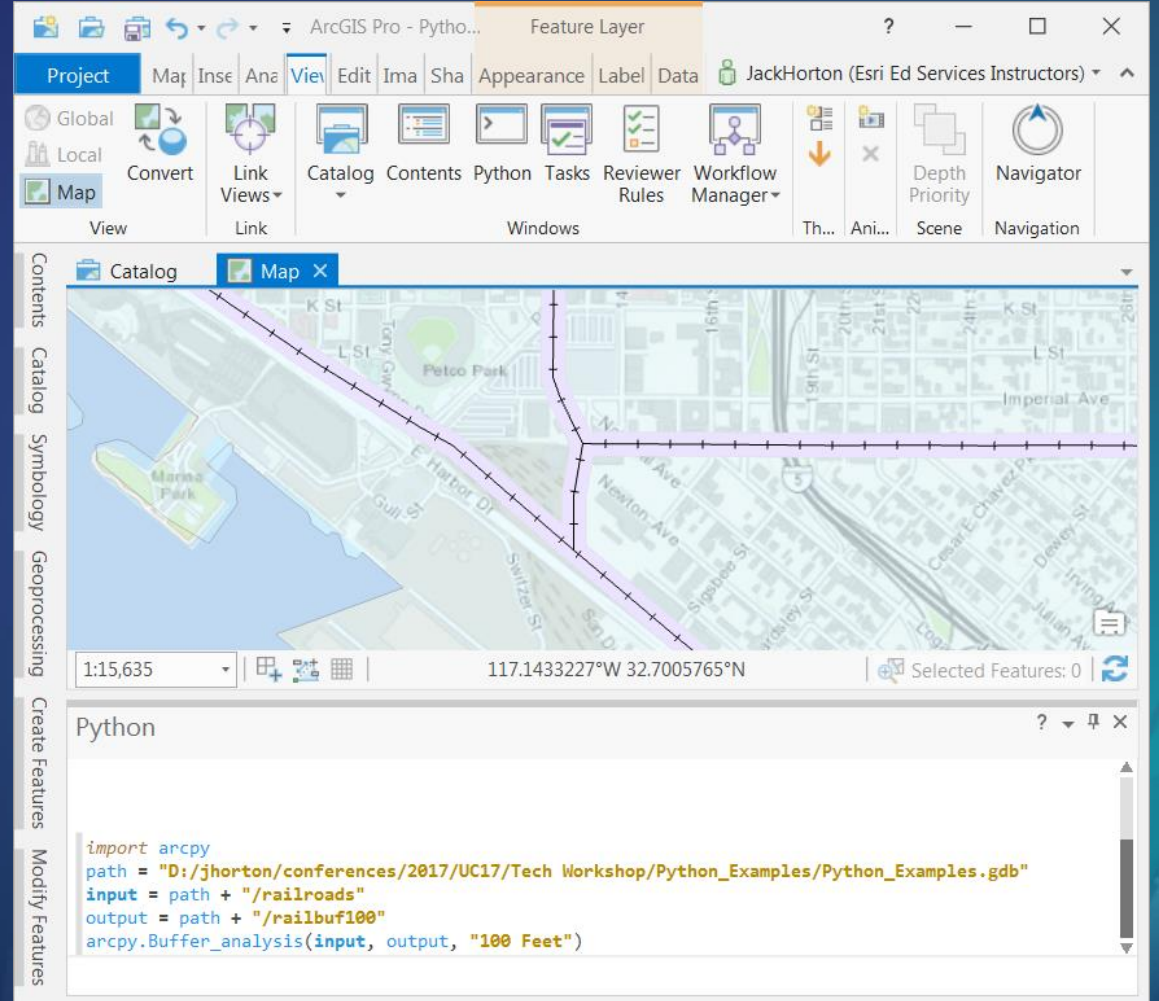
Table Of Contents

- Layers
  - Railroads
  - railbuf100
  - Climate
    - <all other values>
    - TYPE
      - Coastal
      - Maritime
      - Transitional
  - Basemap
    - World Topographic Map

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")
>>>
```

6282052.96 1839029.715 Feet



ArcGIS Pro - Python

Feature Layer

Project Map Inse Ana View Edit Ima Sha Appearance Label Data JackHorton (Esri Ed Services Instructors)

Global Local Convert Link Views Catalog Contents Python Tasks Reviewer Workflow Manager

View Link Windows Th... Ani... Scene Navigation

Contents Catalog Map

1:15,635 117.1433227°W 32.7005765°N Selected Features: 0

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")
```

# 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()`

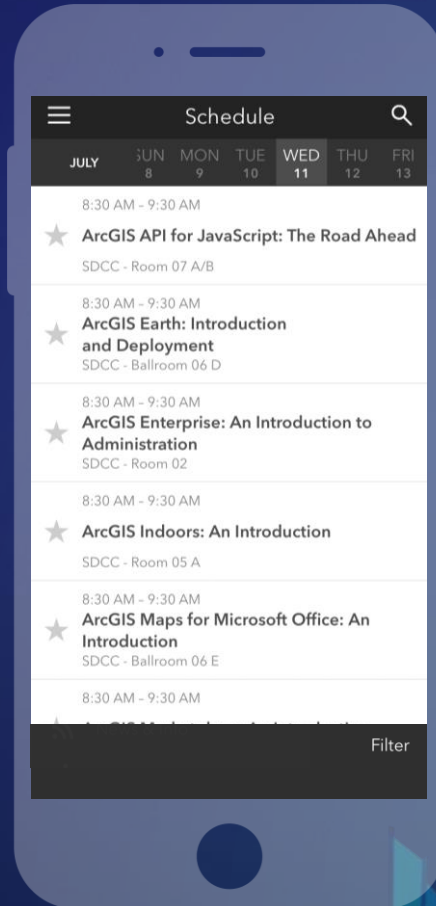
**5. Use `print()` functions or a debugger to figure out what your script is really doing and test as you go**

# Please Take Our Survey on the App

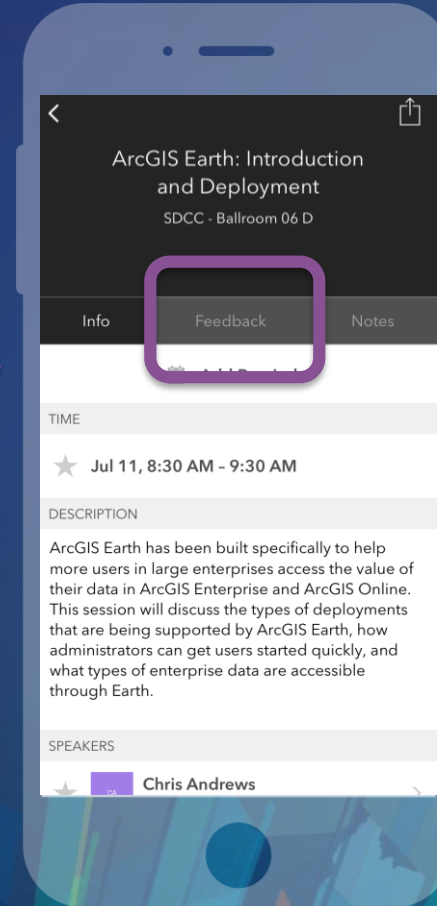
Download the Esri Events app and find your event



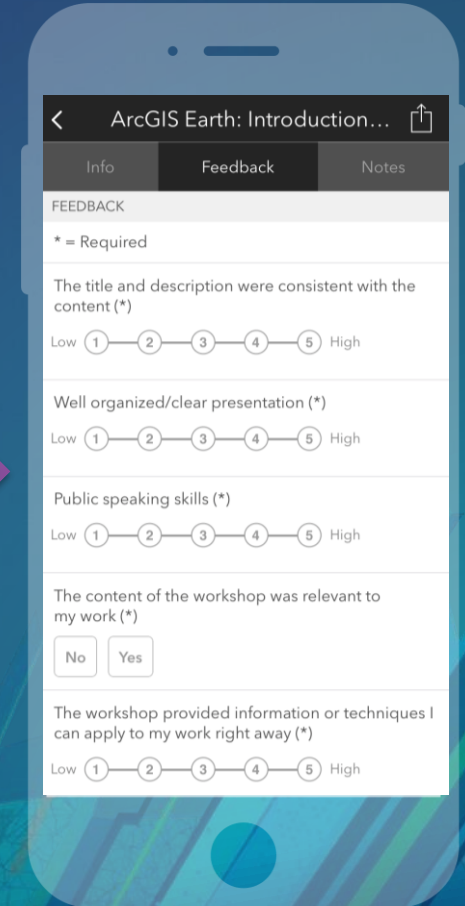
Select the session you attended



Scroll down to find the feedback section



Complete answers and select "Submit"







esri

THE  
SCIENCE  
OF  
WHERE