Exercise 7

Debugging and error handling

Exercise data

Exercise data for this book can be downloaded from links.esri.com/PythonPro3rdEditionData . This is a link to the ArcGIS® Online group called *Python Scripting for ArcGIS Pro – 2024 (Esri Press)*. The data for exercise 7 is posted as a zip file called PythonScripting_Ex07_Data.zip. Download this file and extract it to a folder of your choice. The instructions use a folder called C:\PythonPro, but you can use a different folder provided you update any paths.

Examine syntax errors and exceptions

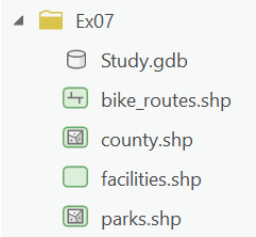
Before starting to work with the exercise data, you will preview the data in ArcGIS® Pro. When starting a new project in ArcGIS Pro, you can leave the home folder, default geodatabase, and default toolbox to their original settings or use the exercise folder—e.g., C:\PythonPro\Ex07). Saving your work is optional.

1. **Start ArcGIS Pro and click Start Without a Template to create a new blank project.**

2. **Make sure the Catalog pane is visible by clicking Catalog Pane on the View tab. Dock the Catalog pane to the right side of the ArcGIS Pro interface.**

3. **Create a new folder connection to the location of the exercise data by right-clicking Folders > Add Folder Connection and navigating to the folder—e.g., C:\PythonPro\Ex07.**

4. **Examine the contents of this folder.**



Notice that there are four shapefiles, including point, polyline, and polygon shapefiles, plus one empty file geodatabase. There are also two scripts in the C:\PythonPro\Ex07folder, which are not visible in ArcGIS Pro. One of these scripts contains errors, which you will fix in this exercise.

5. **Save the project as Exercise 7 to the C:\PythonPro\Ex08 folder. Then, close ArcGIS Pro.**

Syntax errors prevent code from being executed. In the following examples, you will identify some common syntax errors.

6. **Start IDLE.**

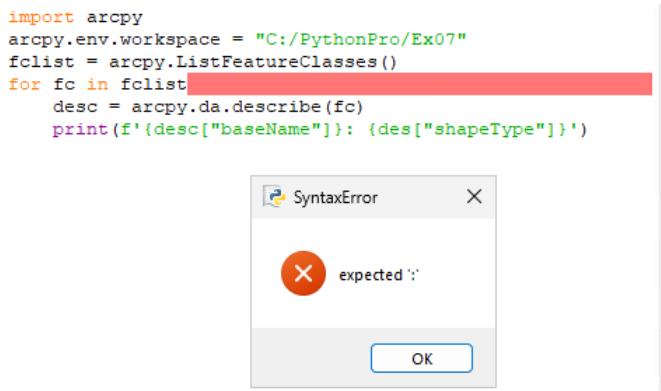
7. **Click File > Open, navigate to the C:\PythonPro folder, and open the script fc_list.py.**

```
import arcpy
arcpy.env.workspace = "C:/PythonPro/Ex07"
fclist = arcpy.ListFeatureClasses()
for fc in fclist
    desc = arcpy.da.describe(fc)
    print(f'{desc["baseName"]}: {des["shapeType"]}')
```

This script has several syntax errors. You may be able to identify them directly, but even so, it is useful to see how they can be found.

8. **Click Run > Check Module.**

This brings up a syntax error message: invalid syntax. A red bar in the script shows the location of the error.



The message reports where the error occurs and that it is a syntax error. You still must identify the exact nature of the syntax error. In this case, the error is a missing colon at the end of the line of code.

9. **Correct the code on line 4 as follows:**

```
for fc in fclist:
```

10. **Save and run the script.**

The script runs but an error message appears in the interactive interpreter, as shown in the figure.

```
========== RESTART: C:\PythonPro\Ex07\fc_list.py ==========
Traceback (most recent call last):
  File "C:\PythonPro\Ex07\fc_list.py", line 5, in <module>
    desc = arcpy.da.describe(fc)
AttributeError: module 'arcpy.da' has no attribute 'describe'. Did you mean: 'Describe'?
>>>
```

An AttributeError exception was raised at line 5 in the script. The module `arcpy` does not have an attribute named `describe`. The correct spelling of the function is `Describe`. Remember that Python is case sensitive, for the most part.

11. Correct the code on line 5 as follows:

```
desc = arcpy.da.Describe(fc)
```

12. Save and run the script.

Again, an error message appears in the interactive interpreter:

```
============================== RESTART: C:\PythonPro\Ex07\fc_list.py ==============================
Traceback (most recent call last):
  File "C:\PythonPro\Ex07\fc_list.py", line 6, in <module>
    print(f'{desc["baseName"]}: {des["shapeType"]}')
NameError: name 'des' is not defined. Did you mean: 'desc'?
>>>
```

A NameError exception was raised at line 6 in the script. The name `des` is not defined. The correct spelling of the variable is `desc`.

13. Correct the code on line 6 as follows:

```
print(f'{desc["baseName"]}: {desc["shapeType"]}')
```

14. **Save and run the script.**

This time the script runs correctly, and the result is printed to the interactive interpreter:

```
bike_routes: Polyline

county: Polygon

facilities: Point

parks: Polygon
```

This example illustrates some of the most common errors in Python scripts: punctuation, capitalization, and spelling.

15. **Save and close the script.**

Handle exceptions

Many different types of errors can occur when running a script. Rather than just letting a script cause a runtime error, you can gain more control using certain error-handling procedures. The most widely used error-handling technique uses a `try-except` statement.

1. **In the Python Shell, click File > Open, navigate to the C:\PythonPro folder, and open the script copy_fcs.py.**

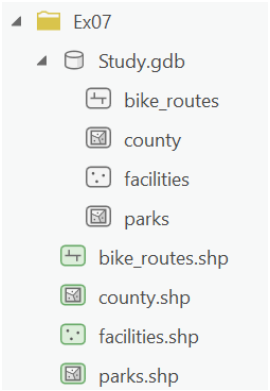
```
import arcpy
import os
arcpy.env.workspace = "C:/PythonPro/Ex07"
arcpy.env.overwriteOutput = True
fgdb = "Study.gdb"
fclist = arcpy.ListFeatureClasses()
for fc in fclist:
    desc = arcpy.da.Describe(fc)
    outfc = os.path.join(fgdb, desc["baseName"])
    arcpy.CopyFeatures_management(fc, outfc)
```

The script creates a list of feature classes in a workspace and copies each feature class to a file geodatabase Study.gdb. But what if the geodatabase Study.gdb does not exist? Before examining that scenario, you will first run the script without errors.

2. **Review the path used in line 3 of the code to specify the workspace. If necessary, modify this path to represent the location of the datasets for this exercise.**

3. **Save and run the script.**

4. **Start ArcGIS Pro and open Exercise 7.**



5. **In the Catalog Pane, confirm that the four feature classes have been copied to the Study.gdb geodatabase.**

Now that you have confirmed that the script works as intended, we'll introduce an error.

6. **Modify line 5 of the script as follows:**

```
fgdb = "Demo.gdb"
```

This geodatabase does not exist in the workspace.

7. **Save and run the script.**

This produces a lengthy error message printed to the interactive interpreter.

```
============================== RESTART: C:\PythonPro\Ex07\copy_fcs.py ==============================
Traceback (most recent call last):
  File "C:\PythonPro\Ex07\copy_fcs.py", line 10, in <module>
    arcpy.CopyFeatures_management(fc, outfc)
  File "C:\Program Files\ArcGIS\Pro\Resources\ArcPy\arcpy\management.py", line 4289, in CopyFeatures
    raise e
  File "C:\Program Files\ArcGIS\Pro\Resources\ArcPy\arcpy\management.py", line 4286, in CopyFeatures
    retval = convertArcObjectToPythonObject(gp.CopyFeatures_management(*gp_fixargs((in_features, out_fea
ture_class, config_keyword, spatial_grid_1, spatial_grid_2, spatial_grid_3), True)))
  File "C:\Program Files\ArcGIS\Pro\Resources\ArcPy\arcpy\geoprocessing\_base.py", line 512, in <lambda>
    return lambda *args: val(*gp_fixargs(args, True))
arcgisscripting.ExecuteError: ERROR 000210: Cannot create output Demo.gdb\bike_routes
Failed to execute (CopyFeatures).

>>>
```

The error message includes several references to scripts that are part of ArcPy, and it is a bit difficult to interpret. However, the end of the message reads as follows:

```
    ExecuteError: ERROR 000210: Cannot create output
Demo.gdb\bike_routes
    Failed to execute (CopyFeatures).
```

The rest of the error message is a bit confusing, which has resulted from the script being interrupted midprocess. Next, you will trap this error using a `try-except` statement to gain control of the error messages.

8. **Modify the code as follows:**

```
import arcpy
import os
arcpy.env.workspace = "C:/PythonPro/Ex07"
arcpy.env.overwriteOutput = True
fgdb = "Demo.gdb"
try:
    fclist = arcpy.ListFeatureClasses()
    for fc in fclist:
        desc = arcpy.da.Describe(fc)
        outfc = os.path.join(fgdb, desc["baseName"])
        arcpy.CopyFeatures_management(fc, outfc)
```

```
except arcpy.ExecuteError:
    print(arcpy.GetMessages(2))
except:
    print("There has been a nontool error.")
```

9. **Save and run the script.**

This time the script runs successfully, meaning that it finishes and is not interrupted midprocess by an error. The incorrect file geodatabase has not been fixed so there is still an error, but the error is caught using the `try-except` statement and only the specific error message prints.

```
========== RESTART: C:\PythonPro\Ex07\copy_fcs.py ==========
ERROR 000210: Cannot create output Demo.gdb\bike_routes
Failed to execute (CopyFeatures).

>>>
```

Although the specific error message is the same as before, there is an important difference: despite the error, the script ran successfully rather than resulting in a runtime error. This distinction is important. Say, for instance, that you called this script as a tool in ArcGIS Pro. If the script results in a runtime error, you may not find out what happened because the printed error message does not appear anywhere. With the use of the try-except statement, the script runs successfully, and the error message can be reported back to the tool that called the script.

End of Exercise 7.