Exercise 9

Working with geometries

Exercise data

Exercise data for this book can be downloaded fromlinks.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 9 is posted as a zip file called PythonScripting_Ex09_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.

Work with geometry tokens

Working with full geometry objects can be costly in terms of time. Geometry tokens provide shortcuts to specific geometry properties of individual features in a feature class. In the next example, you will see how to work with geometry tokens in scripting.

1. **Start IDLE.**
2. **Click File > New File to create a new script file and save your script as print_length.py to the C:\PythonPro\Ex09 folder.**
3. **Enter the following lines of code:**

```
import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex09"

fc = "rivers.shp"
```

```
with arcpy.da.SearchCursor(fc, ["SHAPE@LENGTH"]) as cursor:
    length = 0
    for row in cursor:
        length = length + row[0]
    print(length)
```

4. **Save and run the script.**

Running this script prints the total length of all the river segments to the interactive interpreter:

```
2622733.446273685
```

The spatial reference object can be used to determine the units.

5. **Replace the last line of code with the following lines of code:**

```
desc = arcpy.da.Describe(fc)
units = desc["spatialReference"].linearUnitName
print(f"{length:.2f} {units}")
```

Your script should look like this:

```
import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex09"
fc = "rivers.shp"

with arcpy.da.SearchCursor(fc, ["SHAPE@LENGTH"]) as cursor:
    length = 0
    for row in cursor:
        length = length + row[0]
    desc = arcpy.da.Describe(fc)
    units = desc["spatialReference"].linearUnitName
    print(f"{length:.2f} {units}")
```

6. **Save and run the script.**

Running this script prints the total length of all the features, followed by the units, to the interactive interpreter, as follows:

```
2622733.45 Meter
```

In the `print()` function, the length is shortened to only two decimals using the .2f format code.

Read geometries

In addition to the geometry properties of a feature, a feature's individual vertices also can be accessed. You'll do that next.

1. **In IDLE, create a new script file, and save your script as points.py to the C:\PythonPro\Ex09 folder.**

2. **Enter the following lines of code:**

```
import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex09"

fc = "dams.shp"

with arcpy.da.SearchCursor(fc, ["SHAPE@XY"]) as cursor:

    for row in cursor:

        x, y = row[0]

        print("{0:.3f}, {1:.3f}".format(x, y))
```

Save and run the script.

Note: The code examples use both f-strings and the `format()` method to print strings. Both approaches are equally valid. Format codes to limit the number of decimals for display purposes can be used in both approaches.

Running this script prints a pair of x,y coordinates for each point, as follows:

```
852911.336, 2220578.935

792026.898, 2310863.768

784830.428, 2315171.539

...
```

Working with other geometry types, such as polylines and polygons, requires extra code because an array of point objects is returned for each feature. As a result, an extra iteration is required to interact with the array before you can get to the points that represent the vertices. Next, you will work with a polyline feature class.

3. **In IDLE, create a new script file, and save your script as vertices.py to the C:\PythonPro\Ex09 folder.**

4. **Enter the following lines of code:**

```
import arcpy
```

```
arcpy.env.workspace = "C:/PythonPro/Ex09"

fc = "rivers.shp"

with arcpy.da.SearchCursor(fc, (["OID@", "SHAPE@"])) as
cursor:

    for row in cursor:

        print("Feature {0}: ".format(row[0]))

        for point in row[1].getPart(0):

            print("{0:.3f}, {1:.3f}".format(point.X,
point.Y))
```

5. **Save and run the script.**

It will take time to print all the vertices, so you may choose to interrupt the execution once you have a sense of the operation.

Running this script prints a pair of x,y coordinates for each vertex in each feature, as follows:

```
Feature 0:

745054.499, 2324903.184

745122.149 2324835.941

...
```

```
Feature 1:

747821.019, 2317111.877

746909.057, 2317254.892

...

Feature 2:

753597.862, 2315541.916

753757.717 2319122.629

...
```

This script works for polyline and polygon feature classes but does not work for multipart features, which you will work with next.

Work with multipart features

For multipart features, cursors return an array containing multiple arrays of point objects. Working with multipart features therefore requires an extra iteration over the parts of each feature.

The `isMultipart` property of the geometry object can be used to determine whether a feature is multipart or not.

1. **In IDLE, create a new script file, and save your script as multipart.py to the C:\PythonPro\Ex09 folder.**

2. **Enter the following code:**

```
import arcpy
```

```
arcpy.env.workspace = "C:/PythonPro/Ex09"

fc = "dams.shp"

with arcpy.da.SearchCursor(fc, ["OID@", "SHAPE@"]) as
cursor:

    for row in cursor:

        if row[1].isMultipart:

            print("Feature {0} is
multipart.".format(row[0]))

        else:

            print("Feature {0} is single
part.".format(row[0]))
```

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

Running this script prints whether each feature in the features class is multipart or single part, as follows:

```
Feature 0 is single part.

Feature 1 is single part.

Feature 2 is single part.

Feature 3 is single part.

...
```

The results are single part for each feature because each feature consists of a single point.

4. **Modify line 3 of the script as follows:**

```
fc = "hawaii.shp"
```

5. **Save and run the script.**

Running this script confirms that the feature class representing Hawaii is a multipart polygon, as follows:

```
Feature 0 is multipart.
```

The `partCount` property can be used to determine the number of parts, which you'll do next.

6. **Modify the `print()` function following the `if` statement as follows:**

```
print("Feature {0} is multipart and has {1} "
      "parts.".format(row[0], str(row[1].partCount)))
```

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

Running this script prints the number of parts for every multipart feature, as follows:

```
Feature 0 is multipart and has 11 parts.
```

Because the geometry object for multipart features consists of an array containing multiple arrays of point objects, it is necessary to iterate over all parts of a feature to obtain their vertices.

8. **Save the script as multipart_vertices.py.**

9. **Modify the script as follows:**

```
import arcpy

arcpy.env.workspace = "C:/PythonPro/Ex09"

fc = "hawaii.shp"

cursor = arcpy.da.SearchCursor(fc, ["OID@", "SHAPE@"])

for row in cursor:

    print("Feature {0}: ".format(row[0]))

    partnum = 0

    for part in row[1]:

        print("Part {0}: ".format(partnum))

        for point in part:

            print("{0:.2f}, {1:.2f}".format(point.X,
point.Y))

        partnum += 1
```

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

Running this script prints the part number of each feature, followed by pairs of x,y coordinates of the vertices. In the case of the feature class hawaii.shp, there is one feature with 11 parts, as follows:

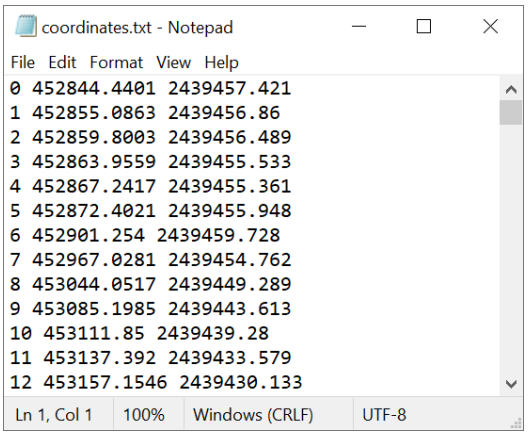
```
Feature 0:

Part 0:

829161.24, 2245088.49

829562.01, 2244825.55

...

Part 1:

757434.85, 2276341.38

757032.14, 2276024.29

...

Part 2:

710001.62, 2315999.27

712130.15, 2315404.38

...
```

Write geometries

Insert and update cursors can be used to create new features and update existing features, respectively. Point objects can be created to set the geometry of these new and updated features. In the next example, you will use an insert cursor to create a new polyline feature from a list of coordinates.

1. **In the search bar on the Windows task bar, type Notepad and click on the application to open it.**

2. **On the Notepad menu bar, click File > Open, and browse to the C:\PythonPro\Ex09 folder. Select the coordinates.txt file, and click Open.**



The file consists of 239 pairs of x,y coordinates, with each pair preceded by an ID number and separated by a space. First, you will create a new empty polyline feature class, and then use this list of coordinates to create a new polyline.

3. **Close Notepad.**

4. **In IDLE, create a new script file, and save your script as create_polyline.py to the C:\PythonPro\Ex09 folder.**

5. **Enter the following code:**

```
import arcpy

import fileinput

import os
```

```
ws = "C:/PythonPro/Ex09"

arcpy.env.workspace = ws

arcpy.env.overwriteOutput = True

newfc = "newpolyline.shp"

arcpy.CreateFeatureclass_management(ws, newfc,"Polyline",
spatial_reference = "dams.prj")
```

This code creates a new empty polyline feature class with the same spatial reference as an existing feature class—in this case, dams.shp.

6. **Save and run the script.**

   Running this script creates a new empty polyline feature class.

7. **Start ArcGIS Pro and click Start Without a Template to create a new blank project.**
8. **In the Catalog pane, 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\Ex09.**
9. **Navigate to the C:\PythonPro\Ex09 folder. Confirm that the file newpolyline.shp has been created, even though it does not contain any features yet.**
10. **Save the project as Exercise 9 to the C:\PythonPro\Ex09 folder.**
11. **Close ArcGIS Pro.**

    There is no need to save your work.

Next, you will read the text file.

12. In the create_polyline.py script, add the following line of code:

```
infile = os.path.join(ws, "coordinates.txt")
```

The full path is needed here because the workspace applies only to ArcPy functions and classes. The coordinates in this text file will be used to create the vertices for a polyline. This requires the use of an insert cursor to create new rows and an array object to contain the point objects.

13. Add the following lines of code:

```
with arcpy.da.InsertCursor(newfc, ["SHAPE@"]) as cursor:
    array = arcpy.Array()
```

Next, the script must read through the text file, parse the text into separate strings, and iterate over the text file to create a point object for each line in the text file.

14. Add the following lines of code:

```
    for line in fileinput.input(infile):
        ID, X, Y = line.split()
        array.add(arcpy.Point(X, Y))
    cursor.insertRow([arcpy.Polyline(array)])
    fileinput.close()
```

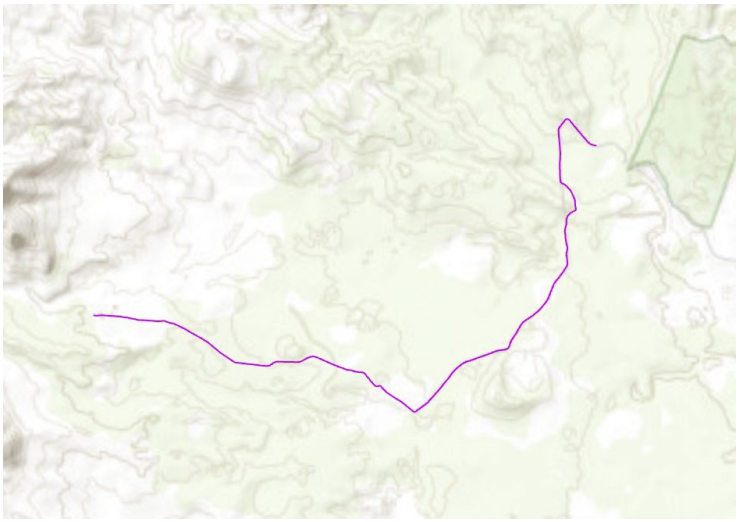
The completed script looks like the example in the figure.

15. **Save and run the script.**

16. **Start ArcGIS Pro and open your Exercise 9 project.**

17. **In the Catalog pane, navigate to the C:\PythonPro\Ex09 folder.**

18. **Add the file newpolyline.shp to a new map.**



The new feature class consists of only a single polyline feature, but a similar procedure can be carried out to create multiple features.

19. **Close ArcGIS Pro.**

There is no need to save your work.

End of Exercise 9.