Get started with Python in ArcGIS Pro

Start learning Python in ArcGIS Pro. Write code to determine the number of features for all the feature classes in a workspace.

Author	Duration	Difficulty
Paul Zandbergen	30mins	Intermediate

Python scripting makes it possible to automate workflows in ArcGIS Pro. In this tutorial, you'll write code to determine the number of features for all the feature classes in the workspace. This also introduces some of the basics of Python syntax. You'll write code in the Python window in ArcGIS Pro. Code can be exported to a Python script, which be opened, modified, and run outside of ArcGIS Pro.

This tutorial was last tested on November 21, 2024, using ArcGIS Pro 3.4. If you're using a different version of ArcGIS Pro, you may encounter different functionality and results.

Requirements

- ArcGIS Pro (see options for software access)

Set up a project and review datasets

Before you begin writing Python code, you'll download the datasets, create a new project, and review the datasets to be used.

1. Download the data for this tutorial and extract the contents to the C:\ drive folder.

   The .zip file contains a folder named **PythonStart**.

   > **Note:**
   > You can use a different folder name, but you'll need to use that folder when creating you project in the following steps.

2. Start ArcGIS Pro. If prompted, sign in using your licensed ArcGIS organizational account.

   > **Note:**
   > If you don't have access to ArcGIS Pro or an ArcGIS organizational account, see options for software access.

3. Under **New Project**, click **Map**.

4. In the **New Project** window, for **Name**, type Python Getting Started. For **Location**, browse to the C:\PythonStart folder.

5. Uncheck the **Create a new folder for this local project** box.

6. Click **OK**.

   The project opens with a blank map.

7. If the **Catalog** pane is not already visible, on the **View** tab, click **Catalog Pane**. If necessary, dock the **Catalog** pane on the right side of the ArcGIS Pro interface.
8. In the **Catalog** pane, expand **Folders** and expand **PythonStart**.

   The folder contains six shapefiles. In this tutorial, you'll only work with a handful of datasets, but the same code can be used to work with a much larger number.

   You'll first manually determine the number of features for a single feature class, and then run a tool to obtain the same result. In the remainder of the tutorial, you'll use Python to determine the number of features for all the shapefiles.

9. Right-click **ambulances.shp** and choose **Add To Current Map**.

   The map zooms in to Toronto, Canada, where the points for the **ambulances** layer are located.

10. In the **Contents** pane, right-click the **ambulances** layer and choose **Attribute Table**.

    The attribute table appears. At the bottom of the table, the number of records is shown.

    The shapefile contains 48 records, which means there are 48 unique features. Next, you'll determine the same count using a tool.

11. Close the attribute table.

Run a tool using Python

Next, you'll run a geoprocessing tool in ArcGIS Pro. Then, you'll run the same tool using Python code.

1. On the ribbon, click the **Analysis** tab. In the **Geoprocessing** group, click **Tools**.

   The **Geoprocessing** pane appears.

2. In the search bar of the **Geoprocessing** pane, type count and press Enter.
3. Click the **Get Count** tool.
4. In the **Get Count** tool pane, for the **Input Rows** parameter, choose **ambulances**.

5. Click **Run**.

   When the tool finishes running, a message appears at the bottom of the pane.

6. Click **View Details**.

   The **Get Count (Data Management Tools)** window appears with the **Messages** tab open.

   The message reads **Row Count = 48**, which is the same count you determined manually by opening the attribute table. You can repeat these steps to determine the count for all the shapefiles, but that would be time consuming if you had many datasets. Instead, you'll develop a Python script to accomplish this task.

7. Close the **Get Count** window.

   Next, you'll use Python to run the same tool.

8. On the ribbon, on the **Analysis** tab, in the **Geoprocessing** group, click **History**.

   The **History** pane appears with a list of tools that have been run. The only entry is the **Get Count** tool that was just run.

9. On the **Analysis** tab, in the **Geoprocessing** group, click the drop-down menu for the **Python** button and choose **Python Window**.

   The **Python** window appears.

   > **Note:**
   > You can move or dock the **Python** window anywhere you like. The example images show the window docked under the map.

   The top section of the **Python** window is called the transcript and the bottom section is called the prompt. The transcript is initially blank. The transcript provides a record of previously entered code and its results.

   The prompt is where you type your code. When the **Python** window first appears, the message in the prompt reads **Initializing Python interpreter**, which means the window is getting ready to receive your code. After a

few seconds the message is replaced with **Enter Python code here**, which means you can start typing your code. After you have opened the **Python** window for the first time, these messages don't appear again in the current session.

10. In the **Python** window, type arcpy.management.GetCount("ambulances").

    ArcPy is a Python package that makes much of the functionality of ArcGIS Pro available from within Python. GetCount() is a function of ArcPy that runs the **Get Count** geoprocessing tool located in the **Data Management Tools** toolbox.

    > **Note:**
    > All of the help topics for geoprocessing tools include Python examples. For instance, in the Get Count help topic, under **Parameters**, click **Python** to see the topic's code sample.

11. Press Enter

    The code runs and the result is <Result '48'>.

    Although the formatting is a bit different, this is the same record count you determined previously. Running the line of code in the **Python** window produces the same results as running the tool using the tool pane. Running the code in the **Python** window also creates a new entry in the **History** pane.

    Other than the time stamp, these two entries are identical, and there is no difference between running a tool using the tool pane and using Python. When using Python, however, you can use code to control how tools are run, including running the same tool many times on different feature classes.

12. In the **Python** window, right-click the code and choose **Clear Transcript**.

    Anything that was run in the previous code remains in memory.

Run code in the Python window

The **Python** window is a convenient place to practice writing Python code.

1. In the **Python** window, click the prompt and type the following line of code:

   print("GIS is cool")

In this line of code, print() is a function. A function in Python carries out a specific task. In this case, the function prints text. Most functions in Python have arguments or parameters, which are provided in parentheses following the function. In this case, the argument is a string. A string in Python consists of a sequence of characters. You create a string by enclosing the characters in a pair of quotation marks.

Python uses both single and double quotation marks to identify strings, provided they are used consistently. Both print("GIS is cool") and print('GIS is cool') are therefore correct, but print("GIS is cool') results in an error. Quotation marks in Python are always straight (as opposed to slanted). They are automatically formatted this way when you type your code in the **Python** window or another coding environment. Sometimes, if you copy code that has been formatted or written in word processing software, the straight quotation marks may be replaced by slanted quotation marks. For example, print(“GIS is cool”) results in an error.

2. Click the end of the line of code and press Enter.

   The line of code is run and is copied to the transcript, followed by the result.

   The prompt is empty again, ready to receive the next line of code.

   This example is simple but illustrates a key aspect of running Python code this way. You write a single line of code that includes specific instructions to be carried out. When you press Enter, the line of code is run, and the instructions are carried out. In this example, the instructions consist of printing text to the screen, but the instructions can consist of many other tasks, as you already saw when running the **Get Count** tool.

   Next, you'll practice with a few more lines of code.

3. It the prompt, type the following line of code:

   x = 37

   In this line of code, x is a variable. A variable is like a container that stores a value. In this example, that value is the integer 37, but it could be another number, or text, or the name of a dataset. The line of code is called an assignment since the variable is assigned a value. The single equal sign is used in variable assignment to indicate that the variable is being set equal to the value. Variables are assigned so they can be used later in the code.

4. At the end of the line of code, press Enter.

   The line of code is run, but no result is printed. The variable is assigned a value, but there are no other instructions to carry out.

5. In the prompt, enter the following line of code and press Enter:

   y = 73

   This line is another assignment statement.

6. In the prompt, enter the following line of code and press Enter:

   x * y

   This line carries out a calculation, in this case multiplying the values of x and y. The result is printed.

   So far, you've entered the lines of code without worrying too much about exactly how to write each line. When using an assignment statement, the spaces are optional. Therefore, x = 37 is the same as x=37. Spaces are commonly added to improve legibility.

7. In the prompt, type the following line of code and press Enter:

   X * y

   This result is an error:

   NameError: name 'X' is not defined. Did you mean: 'x'?

   Python is case sensitive, so X is different from x. You defined x by assigning it the value 37, but capital X has not been assigned a value, which explains the error.

   As you continue running lines of code in the **Python** window, all the code previously run, and the results, appears in the transcript. You can clear the transcript, but this does not clear the values of any variables you've assigned, though restarting your ArcGIS Pro session will clear them.

8. Right-click in the transcript and choose **Clear Transcript**.

   Next, consider a more applied example: the conversion between temperature in Fahrenheit (F) and Celsius (C). The general formula is:

   F = 9/5 * C + 32

   This can be written in Python as a simple calculation.

9. In the prompt, enter the following lines of code and press Enter at the end of each line:

   temp_c = 17

   temp_f = temp_c * 9 / 5 + 32

   print(temp_f)

   The result of the calculation is printed.

The first line of the code assigns a numeric value to the variable temp_c. The second line performs a calculation using the variable temp_c and the result is assigned to a new variable temp_f. The third line prints the value of the variable temp_f. This third line can also be written as temp_f, but using the print() function is common since it provides better formatting options.

The same result can also be obtained without using variables.

10. In the prompt, enter the following line of code and press Enter:

    17 * 9 / 5 + 32

    The result is also 62.6.

    There are some good reasons for using variables. First, in many cases, you do not want to perform the calculation on a single value but on many values. Second, often you need to write code to perform the calculation, but you do not know in advance the values to be used. When you write the calculation as the relationship between two variables, you can reuse the calculation and it does not depend on a single value.

Get assistance writing code

The **Python** window includes several features to help you with writing code. Consider the earlier example of using the print() function.

1. Clear the transcript.
2. In the prompt, enter the following code:

    pri

    Even before you have completed typing print, a pop-up appears with the print() function. The blue icon with the letter F indicates that this a function. This is called autocompletion or code completion.

3. Point to the print() pop-up.

    This shows the syntax help for the function.

    The help for the print() function provides information about the parameters it takes and their order.

    You can continue typing, or you can click the pop-up to autocomplete this piece of code.

4. Click the print() pop-up.

    The print() function now appears in the prompt. A pair of parentheses has been added automatically. If you don't click the pop-up but continue typing, after you type print(, a closing parenthesis is added automatically. This is another example of code completion to assist you in writing correct syntax.

When your cursor is inside the parentheses, the syntax help for the function appears again.

5. Type a quotation mark between the parentheses of the print() function.

   A second quotation mark is added automatically.

   The **Python** window provides autocompletion pop-ups, interactive display of syntax help, and hints related to syntax errors.

6. Delete the code in the prompt.
7. Confirm that the active map still contains a feature layer named **ambulances**.
8. In the prompt, enter the following code:

   arc

   A pop-up appears with the ArcPy package, indicated by the red icon with the letter P.

   Packages are a way of extending the core set of Python code. ArcPy is a package that adds ArcGIS functionality to Python.

9. Click the arcpy pop-up.

   The arcpy package now appears in the prompt, followed by a dot. For very short pieces of code, such as this example, typing out all the characters is very quick and clicking the pop-ups does not save much time. However, for longer code elements, using the code completion saves time, avoids typos, and provides syntax assistance.

10. After arcpy., type Get, as in the following line of text:

    arcpy.Get

    A list appears of code elements that start with Get that logically follow arcpy.

    Code completion is context sensitive. For example, if you started typing Get at the start of a line of code (without arcpy.), the options displayed are very different compared to arcpy.Get.

11. Click **GetCount()management** in the list of options.

    Note:
    The word **management** after GetCount() indicates that GetCount() is a part of the set of **Management** tools.

    The prompt is populated with the following code:

    arcpy.management.GetCount()

This is identical to the code used earlier in this tutorial, where management refers to the **Data Management Tools** toolbox and GetCount() refers to the **Get Count** tool.

With the cursor between the parentheses, two pop-ups appear.

The lower pop-up shows the syntax of the **Get Count** tool. The syntax explains that the only parameter is named **in_rows**, which consists of an input table view or raster layer. The upper pop-up shows the name of the ambulances layer in the active map. This is a code completion prompt for the **in_rows** parameter of the **Get Count** tool. In other words, the **Python** window recognizes that the layer in the active map is a valid parameter of the tool. You can choose to use this layer or type something else.

12. Click the ambulances pop-up.

    The prompt is populated with the following code:

    arcpy.management.GetCount('ambulances')

    Code completion in this case automatically adds a pair of quotation marks around the name of the layer. Code completion in the **Python** window is a good way to learn the proper syntax. While code completion adds single quotation marks, you can also use double quotation marks here.

    There is no need to run this code, since it is identical to the code you ran before.

13. Delete the code in the prompt.

Iterate using a for loop

Next, you'll try iteration, which means repeating the same steps multiple times. You'll create a list of values and perform the same calculation on each element of the list.

1. In the prompt, enter the following line of code and press Enter:

   templist_c = [17, 19, 24, 21, 16]

   This code creates a Python list that contains five elements of the same type; in this case, they are centigrade temperature values. Lists are a very common data type in Python. A list consists of a sequence of elements surrounded by brackets [ ], sometimes referred to as square brackets, and the elements are separated by commas.

2. In the prompt, enter the following line of code and press Enter:

   for temp_c in templist_c:

   When you press Enter at the end of this line of code, the code does not run but the prompt moves to the next line. The line of code ends with a colon, meaning that more code is to follow, and the line itself cannot be run

on its own.

The line of code is the start of a for loop, which has the following general structure:

for <element> in <list>:

<execute one or more lines of code>

A for loop allows you to iterate over the elements of an existing list and repeat the same steps for each element. The line of code that contains the for keyword ends with a colon. The next line of code is indented, and all the lines of code that immediately follow with the same indentation will be run with each iteration. The block of code that repeats can be identified by its indentation.

The **Python** window recognizes the for loop because of the use of the colon, and therefore the next line of code is indented.

3. In the prompt, type the following line of code and press Enter:

   temp_f = temp_c * 9 / 5 + 32

   This code should remain indented. If you accidentally delete the indentation, you can indent the line by adding four spaces at the start. Four spaces is the default indentation level for a code block.

4. In the prompt, type the following line of code and press Enter:

   print(temp_f)

   The use of print() is not required here, but in many instances, it results in better formatting of the outputs.

   The for loop is complete and is ready to execute.

5. With the cursor still in the empty line of code, press Enter.

   The result prints.

   The for loop iterates over all the elements in the list and performs the same calculation. This is a very powerful concept in programming, since the effort to write the code is the same regardless of whether the list contains 5 elements or 5,000.

Create and iterate over a list of feature classes

Now that you've practiced writing and running code in the **Python** window, you'll return to the original task of determining the number of features for every feature class in the folder of interest.

You'll use the **Get Count** tool again.

1. Clear the transcript. Confirm that the active map still contains a feature layer named **ambulances**.
2. Run the following lines of code:

   ```
   count = arcpy.management.GetCount("ambulances")

   print(count)
   ```

   The result is the value 48. This code is almost identical to the code you ran earlier in this tutorial, but now the result from the **Get Count** tool is assigned to a variable. This makes it easier to work with the result, even though for now the only thing you're doing with the result is printing its value.

   The code so far uses the name of the feature layer that is open in the active map. While this is convenient, it is not very practical to have to add all the shapefiles to a map and run the code manually for each feature layer. Instead, you can point to the feature class on disk by specifying the full path.

3. Run the following lines of code:

   ```
   count = arcpy.management.GetCount("C:/PythonStart/ambulances.shp")

   print(count)
   ```

   The same result prints.

   > **Note:**
   > If you saved the **PythonStart** folder to a location on your computer other than your C:\ drive folder, you'll need to update the path accordingly for this line of code and future lines of code that refer to the folder's location.

   The path is in quotation marks because it is a string. A forward slash (/) is used in the path instead of a regular backslash (\). A backslash in Python is used as an escape character, which may change the meaning of the character following it. This can result in unintended consequences when you use backslashes in strings. Also, the feature class is referenced as ambulances.shp since the .shp file extension is part of the name. Using only ambulances would return an error since there is no feature class named ambulances in the folder.

   Instead of specifying the full path, you can set the workspace. A workspace is one of several environment settings that influence geoprocessing operations. In addition to the workspace, environments include the default output coordinate system, the default cell size for raster data processing, and several others.

4. Run the following lines of code:

   ```
   arcpy.env.workspace = "C:/PythonStart"

   count = arcpy.management.GetCount("ambulances.shp")

   print(count)
   ```

The same result prints.

Although setting the workspace requires an extra line of code, it is often effective to use a workspace since all following lines of ArcPy code will automatically use it. The second line of code no longer requires the full path. If no workspace is set in the code, the default workspace of the project is used.

The final step to accomplish the original task of determining the number of features for every dataset in the folder is to add some code to create a list of shapefiles in the workspace and run the **Get Count** tool on every element of this list. Once the workspace is set, you can create a list of feature classes in this workspace using a function of ArcPy.

5. Enter the following line of code and press Enter:

   fc_list = arcpy.ListFeatureClasses()

   The ListFeatureClasses() function creates a list of feature classes. By assigning this to a variable, you can use the list in other tasks.

   To check the contents of the list, you can print its values.

6. Run the following line of code:

   print(fc_list)

   The result is:

   ['ambulances.shp', 'boundary.shp', 'fire_stations.shp', 'fire_zones.shp', 'voting_divisions.shp', 'voting_sites.shp']

   Next, you'll create a for loop to iterate over the elements in the list.

7. Enter the following line of code and press Enter to start a block of code:

   for fc in fc_list:

8. Enter the following line of code and press Enter:

   count = arcpy.management.GetCount(fc)

   This line of code uses the same **Get Count** tool that you used before, but now it is using the variable fc instead of the name of a specific feature class. Since fc is updated to contain each feature class name in the feature class list, with every iteration the next feature class is used.

9. Enter the following line of code and press Enter twice:

   print(count)

   The result prints the number of features for each of the shapefiles in the workspaces.

The code you've developed accomplishes the task of counting the features in each shapefile. The code is not very polished, since if there was a large number of shapefiles it would be cumbersome to read the printout. You can add code to create a more meaningful output, such as including the feature class name after each count, writing the results to a text file, calculating a total number of features in the workspace, or determining the feature class or classes with the most features.

The same code used in the **Python** window can also be used in an ArcGIS Notebooks in ArcGIS Pro (or in a Jupyter Notebook, outside of ArcGIS Pro). Those steps are not covered here, but you can copy the code from the **Python** window and paste it into a cell in Notebooks.

You can also use the code outside of ArcGIS Pro in a Python editor, which you'll do next. You can save your code from the **Python** window to a Python script file to start your work in a Python editor.

10. Right-click the code in the transcript of the **Python** window and choose **Save Transcript**.
11. In the **Save Transcript** window, browse to the C:\PythonStart folder and save your work as count_features.py.
12. Save your project and close ArcGIS Pro.

You are now ready open the script in a Python editor.

Run a script in a Python editor

While running Python code in ArcGIS Pro using the **Python** window or inside a Notebook is convenient, sometimes you need to run code outside of ArcGIS Pro. A typical example is when you want to schedule a script to run at a predetermined time, but there are other scenarios as well. For example, more complex projects often require multiple code elements that work together, and typically this means organizing your code in several separate .py files. You can use a script as part of creating a custom tool, which also requires a .py file.

Running Python code outside of ArcGIS Pro requires a Python code editor, also referred to as an integrated development environment (IDE). You'll use an IDE called IDLE, which stands for Integrated DeveLopment Environment (notice the uppercase L). IDLE comes with every installation of Python, so if your computer has Python it also has IDLE.

There are many other Python IDEs, including PyCharm and Spyder, but IDLE is a good place to start. IDEs such as PyCharm and Spyder require additional steps to download, install, and configure, while IDLE is part of any Python installation and is ready to be used.

1. Start File Explorer.
2. Browse to the C:\PythonStart folder.
3. Right-click the **count_features.py** file and choose **Edit with IDLE (ArcGIS Pro)**.

The Python script opens in a script window in IDLE. You may also see the option to **Edit with IDLE** if you have ArcGIS Desktop 10.x installed. You should use **Edit with IDLE (ArcGIS Pro)** because this opens the script with the version of Python installed with ArcGIS Pro. ArcGIS Desktop 10.x uses an older version of Python.

> **Note:**

If **Edit with IDLE (ArcGIS Pro)** is not visible in the context menu, click **Start**, expand **ArcGIS**, and open **Python Command Prompt**. In the **Python Command Prompt** window, type idle and press Enter. The **IDLE** (**Python 3.9 Shell)** window appears. Click **File** and choose **Open**. Browse to and open the **count_features.py** file.

The top of the script window shows the version of IDLE used in Python. This is the version of Python that installs with your ArcGIS Pro application. A script written in Python 3.7 may not work in older (2.x) versions of Python.

The code includes the entire contents of the transcript of the **Python** window, including any results. Results are preceded by the # symbol or hash mark, which means they are interpreted as comments. Python skips anything in the comments, but they can be a very useful way to document what different parts of your code do.

Before working with this code, you'll take a quick look at how the Python Shell in IDLE works.

4. From the top menu in IDLE, click **Run** and choose **Python Shell**.

   The Python Shell, or interactive interpreter, opens. The symbol >>> is known as the prompt. While you are using the interactive interpreter, you can leave the script window with the **count_features.py** file open since you'll use it later.

5. Place your cursor after the prompt (after >>>) and type the following code:

   ```
   print("GIS is cool")
   ```

6. Press Enter.

   The result prints to the next line and a new prompt appears.

Running code in the interactive interpreter is similar to running code line by line in the **Python** window. Every time you press Enter, the line of code is run and the result, if any, prints to the next line. One key difference is that the Python Shell does not have separate sections for the transcript and the prompt, but otherwise the two approaches are nearly identical.

Next, you'll clean up the script and run the code.

7. Close the Python Shell window and return to the script window for the **count_features.py** script.
8. Clean up the code by removing all the lines before the line that begins with arcpy.env.workspace and delete all the lines that are results, indicated by the # symbol, which means it is a comment code.
9. Delete these two consecutive lines:

   ```
   count = arcpy.management.GetCount("ambulances.shp")
   ```

print(count)

10. Delete this line:

    print(fc_list)

    The remaining code is:

```
arcpy.env.workspace = "C:/PythonStart"
fc_list = arcpy.ListFeatureClasses()
for fc in fc_list:
    count = arcpy.management.GetCount(fc)
    print(count)
```

11. Ensure that the final line is indented by four spaces to match the line above.
12. Click **File** and choose **Save** to save the script.
13. Click **Run** and choose **Run Module** to run the script.

    The result prints to the interactive window.

    Instead of printing the feature counts, the interactive window prints the following error:

    NameError: name 'arcpy' is not defined

    The reason for the error is that you are using the ArcPy package, but the code is running outside of ArcGIS Pro. To use a package, it needs to be imported at the top of your script.

14. Close the Python Shell window.
15. Place your cursor at the first character position of the first line of script. Press Enter.

    A blank line is added to the beginning of the script.

16. Add the following code at the first line of the script:

    import arcpy

17. Save and run your script.

    The resulting counts print to the interactive window.

The first time you use ArcPy in your script, it may take a few moments for the results to appear because it takes a few seconds for Python to import ArcPy.

The result is identical to running the code in ArcGIS Pro, but ArcGIS Pro does not need to be open for the script to run (though it does need to be installed and licensed on the computer you are using).

There are several benefits to using a Python editor to work with your code. One of the key benefits is that you can write longer scripts and save the code as .py files. Saving the code as a file also makes the code easier to debug and reuse.

Comparisons of approaches to run Python code

You've seen two ways to write and run Python code. A summary of the pros and cons of each approach follows. Not all these elements were covered in depth in this tutorial, but these points will be helpful to remember as you continue learning Python.

Method	Pros	Cons
Python window in ArcGIS Pro	• Easy to get started • Interact with data and maps inside ArcGIS Pro • Provides intuitive code completion assistance	• Limited to relatively short pieces of code • Missing many features common in Python editors • Not designed to save and organize your scripts • Can only be used in ArcGIS Pro
Python editor (IDE)	• Code runs without ArcGIS Pro being open • Includes many features to assist with writing and testing more complex code • Work is saved as .py files to organize more complex code projects • Different IDEs cater to different types of users and skill levels • Some IDEs can be used for multiple programming languages	• Functionality varies with the specific IDE being used • Code completion prompts may be lacking depending on the IDE • Some IDEs require custom configuration before they can be used with ArcGIS Pro • Some IDEs can be complicated for beginners

Note:
For code that runs in ArcGIS Pro, including the **Python** window, you do not need to use import arcpy. For code that runs outside of ArcGIS Pro, such as in a Python editor, you do need to use import arcpy before you can use the functionality of the ArcPy package.

There are several other ways to run Python code. This includes running code in a Notebook, running a script using command line, or scheduling to run a script from the operating system. These are covered in other tutorials.

You have learned how to write and run Python code in the **Python** window and in IDLE. You learned about variables and loops, two very important concepts in Python. You also learned how to set the environment in Python, and how to use standard ArcGIS geoprocessing tools in Python. This is the first in a series of tutorials on how to use Python in ArcGIS Pro—stay tuned for more!

You may also be interested in Python Scripting for ArcGIS Pro andAdvanced Python Scripting for ArcGIS Pro by Dr. Paul A. Zandbergen, published by Esri Press.

You can find more tutorials in the tutorial gallery.

Acknowledgements

- Contains information licensed under the Open Government Licence – Toronto.
- Topographic map sources: Esri, HERE, Garmin, FAO, NOAA, USGS, Intermap, METI, OpenStreetMap contributors, and the GIS User Community

Send Us Feedback

Please send us your feedback regarding this tutorial. Tell us what you liked as well as what you didn't. If something in the tutorial didn't work, let us know what it was and where in the tutorial you encountered it (the section name and step number). Use this form to send us feedback.

Share and repurpose this tutorial