

CHAPTER 3

Creating Python script tools

3.1 Introduction

This chapter describes the process of turning a Python script into a script tool. Script tools make it possible to integrate your scripts into workflows and extend the functionality of ArcGIS Pro. Script tools can be run as stand-alone tools using their tool dialog box, but they can also be used within a model or called by other scripts. Script tools have a tool dialog box, which contains the parameters that are passed to the script. Developing script tools is relatively easy and greatly enhances the experience of using a script. Tool dialog boxes reduce user error because parameters can be specified using drop-down lists, check boxes, combo boxes, and other mechanisms. This use of a tool dialog box provides substantial control over user input, greatly reducing the need to write a lot of code for error-checking. Creating script tools also makes it easier to share scripts with others.

3.2 Script tools versus Python toolboxes

Before getting into how script tools are created, it is important to distinguish between two types of tools that can be developed for using Python in ArcGIS Pro. The focus of this chapter is on how to create a **script tool**, also referred to as a Python script tool. The code for these tools is written as a Python script, and this script is called when the tool runs. The tool dialog box for the script tool is created from within ArcGIS Pro. Tool properties and parameters are created manually using the interface options of the ArcGIS Pro application. This approach provides an intuitive and easy-to-learn way to create and test a script tool. Although most script tools use Python, it is possible to use other scripting languages that accept arguments. For example, you could use a .com, .bat, .exe, or .r file instead of a .py file. A script tool calls a single script file, although other scripts can be called from the main script when the tool runs.

The second approach is to create a tool using a **Python toolbox**. In this approach, the entire tool dialog box is written in Python, and the script is saved as a .pyt file that is recognized as a Python toolbox in ArcGIS Pro. Creating a Python toolbox does not use any of the interface options in ArcGIS Pro, and the toolbox is created entirely in a Python editor. Python toolboxes can be written only in Python, and a single Python toolbox can contain multiple tools, all written in the same script file. Chapter 4 covers creating a Python toolbox in detail.

When you first learn how to create a tool for ArcGIS Pro using Python, you are recommended to start with a script tool because the process is more intuitive. Once you gain some experience in creating script tools, you can start using Python toolboxes as well. The same task can be accomplished using both approaches, and the choice is largely a matter of preference and experience.

The end of chapter 4 revisits some of the pros and cons of script tools and Python toolboxes.

3.3 Why create your own tools?

Many ArcGIS Pro workflows consist of a sequence of operations in which the output of one tool becomes the input of another tool. ModelBuilder and scripting can be used to automatically run these tools in a sequence. Any model created and saved using ModelBuilder is a tool because it is in a toolbox (a .atbx or legacy .tbx file) or a geodatabase. A model, therefore, is typically run from within ArcGIS Pro. A Python script (.py file), however, can be run in two ways:

Option 1: As a **stand-alone script**. The script is run from the operating system or from within a Python editor. For a script to use ArcPy, ArcGIS Pro must be installed and licensed, but ArcGIS Pro does not need to be open for the script to run. For example, you can schedule a script to run at a prescribed time directly from the operating system.

Option 2: As a tool within ArcGIS Pro. The script is turned into a tool to be run from within ArcGIS Pro. Such a tool is like any other tool: it is in a toolbox, can be run from a tool dialog box, and can be called from other scripts, models, and tools.

Using tools instead of stand-alone scripts has several advantages. These benefits apply to both script tools and Python toolboxes.

- A tool includes a **tool dialog box**, which makes it easier for users to enter the parameters using built-in validation and error checking.
- A tool becomes an integral part of geoprocessing. Therefore, it is possible to access the tool from within ArcGIS Pro. It is also possible to use the tool in ModelBuilder and in the Python window, and to call it from another script.
- A tool is fully integrated with the application it was called from. So, any environment settings are passed from ArcGIS Pro to the tool.
- The use of tools makes it possible to write tool messages.
- Documentation can be provided for tools, which can be accessed like the documentation for system tools.

- Sharing a tool makes it easier to share the functionality of a script with others.
- A well-designed tool means a user requires no knowledge of Python to use the tool.

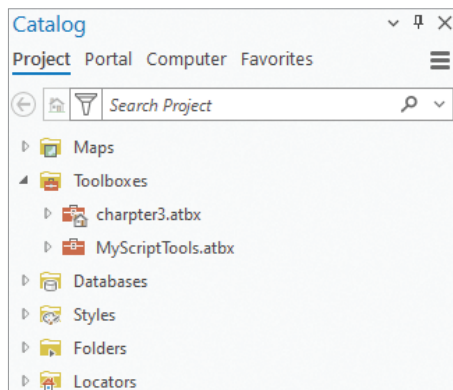
Despite the many benefits, developing a robust tool requires effort. If the primary purpose of the script is to automate tasks that are carried out only by the script's author, the extra effort to develop a script tool may not be warranted. On the other hand, scripts that are going to be shared with others typically benefit from being made available as a tool.

3.4 Steps to creating a script tool

A script tool is created using the following steps:

1. Create a Python script that carries out the intended tasks and save it as a .py file.
2. Create a custom toolbox (a .atbx file) where the script tool can be stored.
3. Add a script tool to the custom toolbox.
4. Configure the tool properties and parameters.
5. Modify the script so that it can receive the tool parameters.
6. Test that your script tool works as intended. Modify the script or the tool's parameters as needed for the script tool to work correctly.

You can create a new custom toolbox on the Project tab of the Catalog pane in ArcGIS Pro. Right-click Toolboxes and click New Toolbox. Select the folder where you want to save the toolbox and give the toolbox a name.



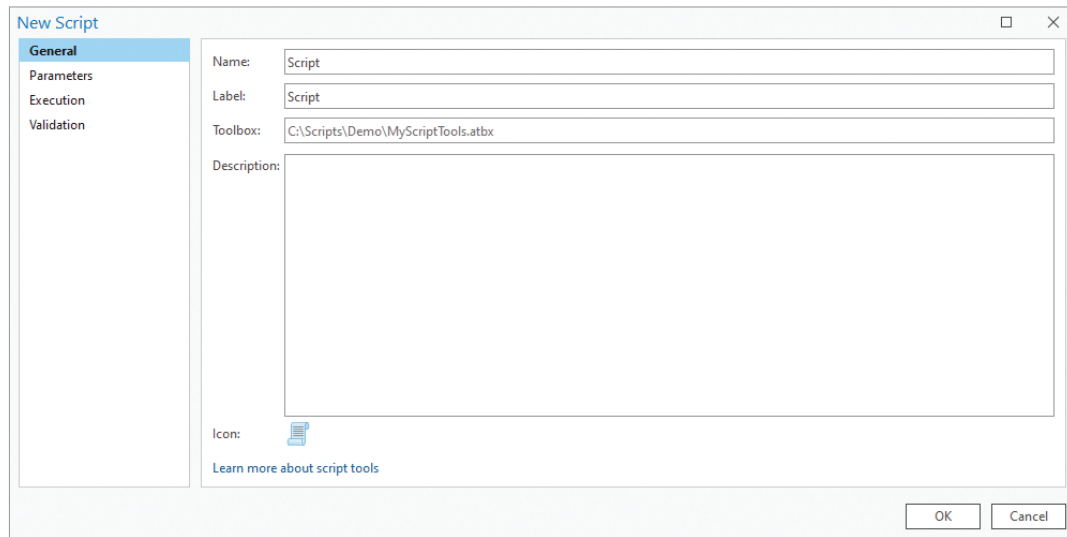
You can also create a new toolbox directly inside a folder or a geodatabase. In that case, you right-click the folder or geodatabase and click New > Toolbox. Although you can create a toolbox inside a geodatabase, the only way to share the toolbox is to share the entire geodatabase. A stand-alone toolbox is saved as a separate .atbx file and can be shared more easily.

Note: Starting with ArcGIS Pro 3.0, you can no longer create legacy toolboxes (.tbx). You can still add and edit existing legacy toolboxes. The Terrain Tools example in chapter 1 is a legacy toolbox.

A stand-alone toolbox can be located anywhere on your computer. For a given project, it makes sense to locate the toolbox in the same folder in which datasets and other files for a project are organized, but you can also have a separate folder for your custom toolboxes, especially if they are used in multiple projects. Stand-alone toolboxes have a file extension—e.g., C:\Demo\MyCoolTools.atbx. A toolbox inside a geodatabase, like other geodatabase elements, does not have a file extension—e.g., C:\Demo\Study.gdb\MyCoolTools.

To create a script tool, right-click a custom toolbox and click New > Script. Write access to the toolbox is needed to add a new script tool. As a result, you cannot add tools to any of the system toolboxes in ArcGIS Pro.

The New Script dialog box has four panels: General, Parameters, Execution, and Validation.



The General panel is used to specify the tool name, label, the toolbox location, and several options. The Parameters panel is used to specify the tool parameters, which will control how a user interacts with the tool. The Execution panel is where the Python code can be viewed and edited. The Validation panel provides further options to control the tool's behavior and appearance. Not all this information must be completed in one step. You can enter some of the basic information, save it, and then return to edit the tool properties later.

All the information needed to create a script tool is reviewed in detail in this chapter. First, however, it is important to consider the example script used for illustration. That way, the information has a context and is more meaningful.

The example script has been created as a stand-alone script. The script creates a random sample of features from an input feature class based on a user-specified count and saves the resulting sample as a new feature class. The complete code is shown next, followed by a figure of the same script. The syntax highlighting in the figure assists with reading the script.

```
# Python script: random_sample.py
# Author: Paul Zandbergen
# This script creates a random sample of input features based on
# a specified count and saves the results as a new feature class.

# Import modules.
import arcpy
import random

# Set inputs and outputs. Inputfc can be a shapefile or geodatabase
# feature class. Outcount cannot exceed the feature count of inputfc.
inputfc = "C:/Random/Data.gdb/points"
outputfc = "C:/Random/Data.gdb/random"
outcount = 5

# Create a list of all the IDs of the input features.
inlist = []
with arcpy.da.SearchCursor(inputfc, "OID@") as cursor:
    for row in cursor:
        id = row[0]
        inlist.append(id)

# Create a random sample of IDs from the list of all IDs.
randomlist = random.sample(inlist, outcount)

# Use the random sample of IDs to create a new feature class.
desc = arcpy.da.Describe(inputfc)
fldname = desc["OIDFieldName"]
sqlfield = arcpy.AddFieldDelimiters(inputfc, fldname)
sqlexp = f"{sqlfield} IN {tuple(randomlist)}"
arcpy.Select_analysis(inputfc, outputfc, sqlexp)
```

```

File Edit Format Run Options Window Help
# Python script: random_sample.py
# Author: Paul Zandbergen
# This script creates a random sample of input features based on
# a specified count and saves the results as a new feature class.

# Import modules.
import arcpy
import random

# Set inputs and outputs. Inputfc can be a shapefile or geodatabase
# feature class. Outcount cannot exceed the feature count of inputfc.
inputfc = "C:/Random/Data.gdb/points"
outputfc = "C:/Random/Data.gdb/random"
outcount = 5

# Create a list of all the IDs of the input features.
inlist = []
with arcpy.da.SearchCursor(inputfc, "OID@") as cursor:
    for row in cursor:
        id = row[0]
        inlist.append(id)

# Create a random sample of IDs from the list of all IDs.
randomlist = random.sample(inlist, outcount)

# Use the random sample of IDs to create a new feature class.
desc = arcpy.da.Describe(inputfc)
fldname = desc["OIDFieldName"]
sqlfield = arcpy.AddFieldDelimiters(inputfc, fldname)
sqlexp = f"{sqlfield} IN {tuple(randomlist)}"
arcpy.Select_analysis(inputfc, outputfc, sqlexp)

```

Ln: 31 Col: 49

A few points of explanation about the script are in order. First, it is important to understand the general logic of the script. The script creates a list of all the unique IDs of the input features and uses the `sample()` function of the `random` module to create a new list with the unique IDs of the random sample. This new list is used to select features from the input features, and the result is saved as a new feature class. Second, the input feature class, the output feature class, and the number of features to be selected are hard-coded in the script. Third, the script works for both shapefiles and geodatabase feature classes. This is accomplished by using `OID@` when setting the search cursor, by using the `OIDFieldName` property to read the name of the field that stores the unique IDs, and by using the `AddFieldDelimiters()` function to ensure correct SQL syntax regardless of the type of feature class.

The SQL syntax also warrants a bit of discussion. The `WHERE` clause uses the `IN` operator to compare the unique ID of the features to the list of randomly selected IDs, as follows:

```
sqlexp = f"{sqlfield} IN {tuple(randomlist)}"
```

In SQL, this list must be in a pair of parentheses, which is equivalent to a tuple in Python. To ensure proper string formatting, f-strings are used, but this formatting can also be accomplished using `.format()`. When testing the script, the following code can be added to check what the `WHERE` clause looks like:

```
print(sqlexp)
```

For a geodatabase feature class, the WHERE clause looks something like the following:

```
OBJECTID IN (1302, 236, 951, 913, 837)
```

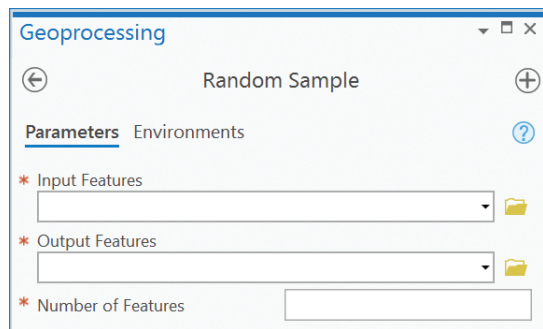
For a shapefile, the WHERE clause looks something like this:

```
"FID" IN (820, 1095, 7, 409, 145)
```

The actual ID values to be selected change with every run of the script because the `sample()` function creates a new random selection regardless of the previous result.

The WHERE clause is used as the third parameter of the Select tool, which creates the output feature class. The script does not work on stand-alone tables because the Select tool works only on feature classes. To work with stand-alone tables, the Select Layer By Attribute tool can be used instead.

Although the script works correctly, making changes to the inputs requires opening the script in a Python editor, typing the inputs, and running the script. A tool dialog box makes it a lot easier to share the functionality of this script. The goal of the script tool is for a user to be able to specify the input feature class to be used for sampling, the output feature class to save the result, and the count of features to be included in the random sample. In other words, the goal is to have a script tool in which the tool dialog box incorporates these features, as shown in the figure.



It is important to have an expectation or visualization of what the final tool dialog box should look like because it facilitates preparing your script, and it guides decisions during the creation of the script tool. Simply drawing out on a piece of paper what you expect the final tool dialog box to look like can help. Certain details may change along the way as you develop and test your script tool, but it helps to have a goal.

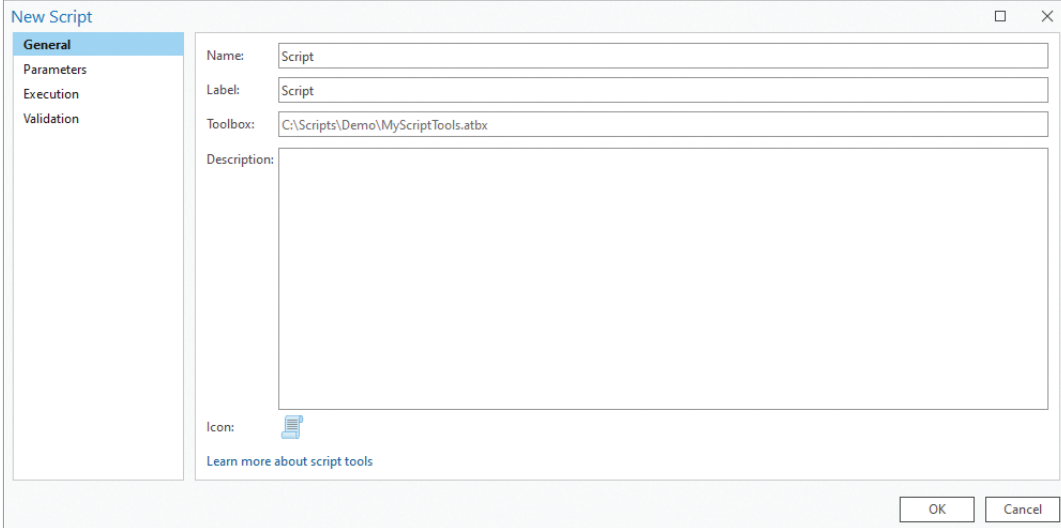
Returning to the code example, a few things are worth noticing about the script. First, the script is broken into sections, each preceded with **comments**, so the logic of the script is easier to follow. Comments are not required for a script tool to work, but if users of the script tool are likely to view the underlying code, comments will make it easier to understand. Second, the script uses several **hard-coded values**, including an input feature class, an output feature class, and a count of features. This type of hard coding is typical for stand-alone scripts. To facilitate using the script as a script tool, hard coding is limited to variables that will become tool parameters. No hard coding is used anywhere else in the script.

To prepare your script for use as a script tool, follow these guidelines:

- First, make sure your script works correctly as a stand-alone script. Working correctly will require the use of hard-coded values.
- Identify which values will become parameters in your script tool. Create variables for these values near the top of your script, and make sure that the hard-coded values are used only once to assign values to the variables. The rest of your script should not contain hard-coded values and use only variables.

Although the stand-alone script will run correctly if the input feature class exists, the script will require changes to be used as part of a script tool. These changes are facilitated by limiting the hard coding of values to the variables that are going to be used as tool parameters.

Returning to the New Script dialog box, it's time to complete the basic information about the script tool in the General panel.



The screenshot shows the 'New Script' dialog box with the following fields and values:

- Name:** Script
- Label:** Script
- Toolbox:** C:\Scripts\Demo\MyScriptTools.atbx
- Description:** (Empty text area)
- Icon:** (Blue document icon)

At the bottom right, there are 'OK' and 'Cancel' buttons. A link 'Learn more about script tools' is located below the icon field.