WWU
MÜNSTER

Python in GIS

ArcGIS

living.knowledge

ifgi
Institut für Geoinformatik
Universität Münster

Learning goals

After this lesson you should be able to do the following with arcpy:

- Set up the working environment
- Manage licenses
- Run tools
- Access attribute values of features via numpy
- Convert rasters from and to numpy arrays
- if time left: Build script tools with Python

Acknowledgements:

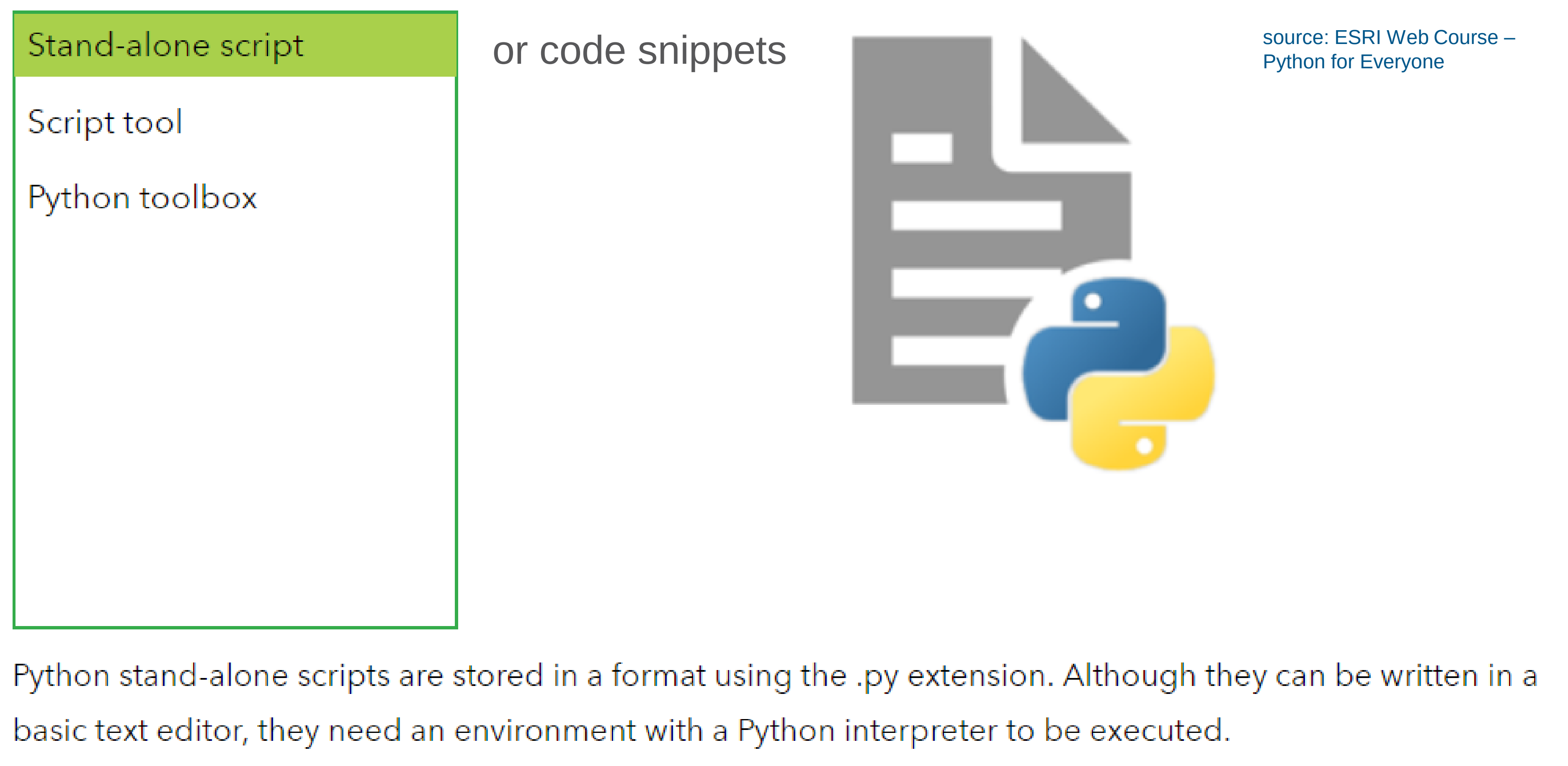
- http://pro.arcgis.com/en/pro-app/arcpy/main/arcgis-pro-arcpy-reference.htm
- ESRI free training: Python for Everyone

Python for ESRI products

Two separate libraries:

- arcpy for Desktop GIS applications: ArcGIS Pro (Python 3) and ArcMap (Python 2)
  - **so for data management, processing etc.**
- arcgis.gis, for Web GIS applications: ArcGIS Online/Portal for ArcGIS
  - **so for user administration, web mapping etc.**

Using arcpy within ArcGIS Pro



Stand-alone script

Script tool

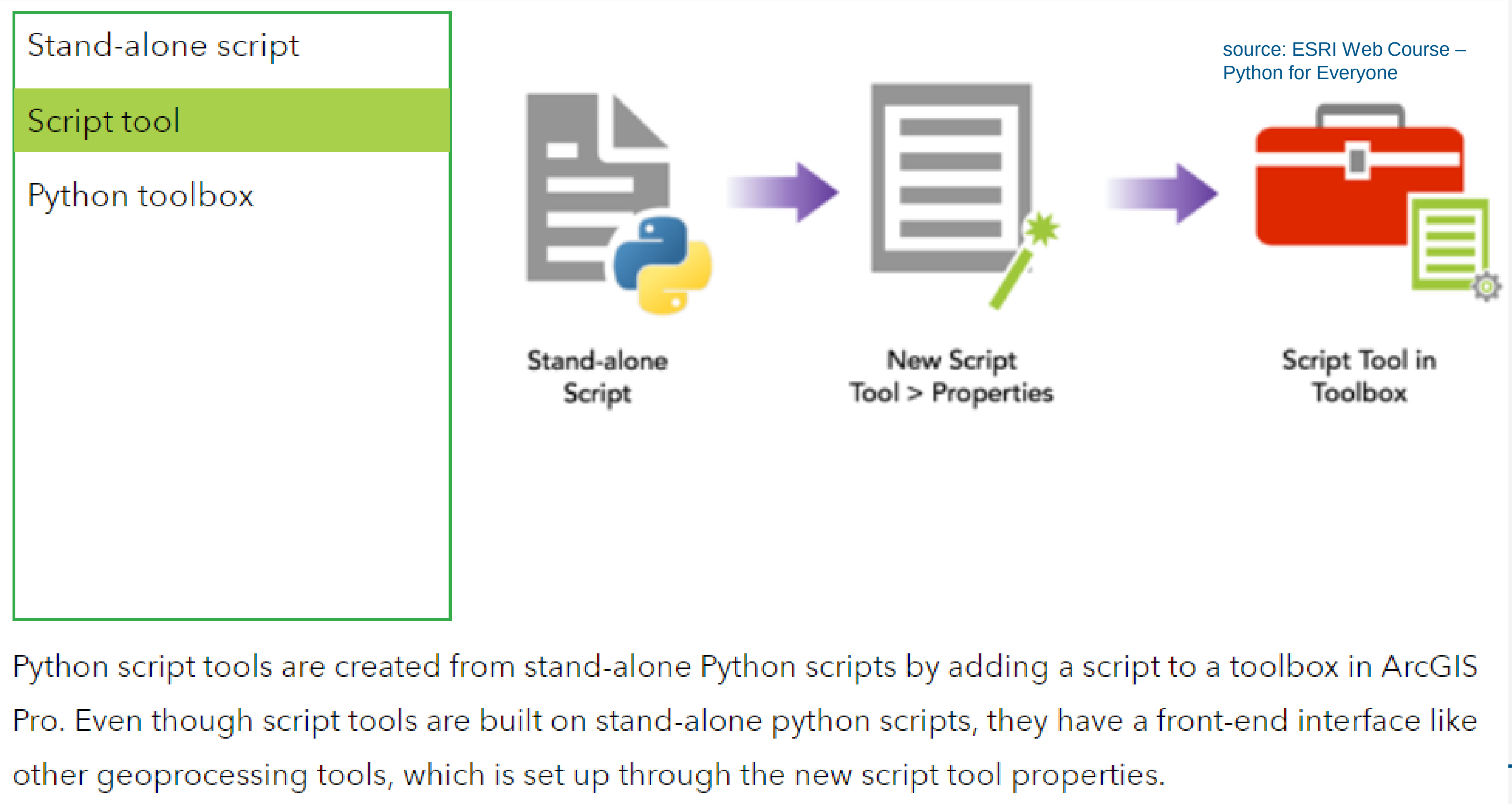
Python toolbox

or code snippets

source: ESRI Web Course – Python for Everyone

Python stand-alone scripts are stored in a format using the .py extension. Although they can be written in a basic text editor, they need an environment with a Python interpreter to be executed.

Using arcpy with ArcGIS Pro

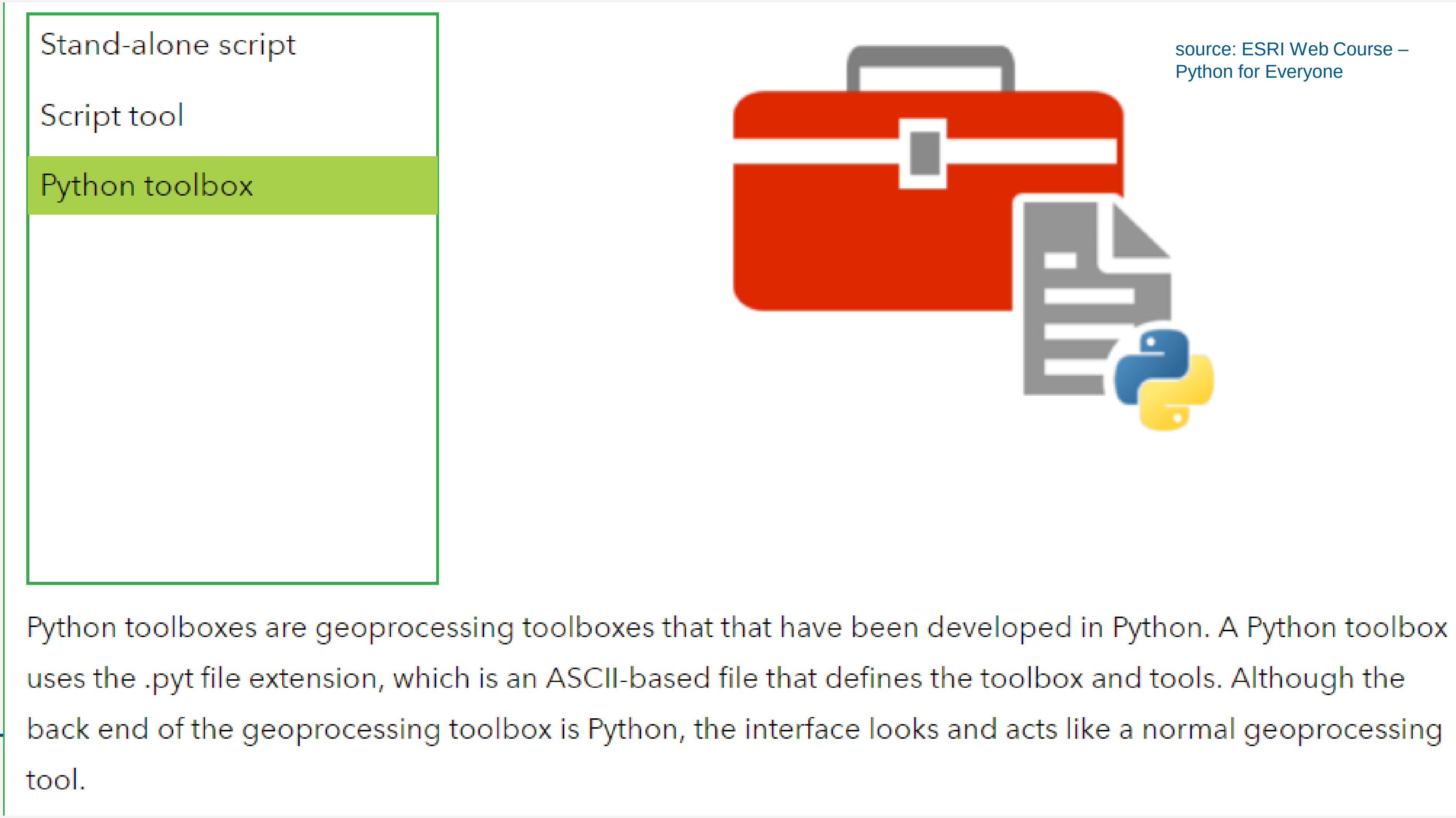


source: ESRI Web Course – Python for Everyone

- Stand-alone script
- Script tool
- Python toolbox

Python script tools are created from stand-alone Python scripts by adding a script to a toolbox in ArcGIS Pro. Even though script tools are built on stand-alone python scripts, they have a front-end interface like other geoprocessing tools, which is set up through the new script tool properties.

Using arcpy with ArcGIS Pro



Stand-alone script

Script tool

Python toolbox

Python toolboxes are geoprocessing toolboxes that that have been developed in Python. A Python toolbox uses the .pyt file extension, which is an ASCII-based file that defines the toolbox and tools. Although the back end of the geoprocessing toolbox is Python, the interface looks and acts like a normal geoprocessing tool.

source: ESRI Web Course – Python for Everyone

Environment

The Environment class

Geoprocessing environment settings can be thought of as parameters that you want to be common among all tools.

They differ from normal tool parameters in that they are set separately from the tool and are used by tools when they are run.

Examples of environment settings are area of interest, the coordinate system of the output dataset, the output folder/GDB, and the cell size of a new raster dataset.

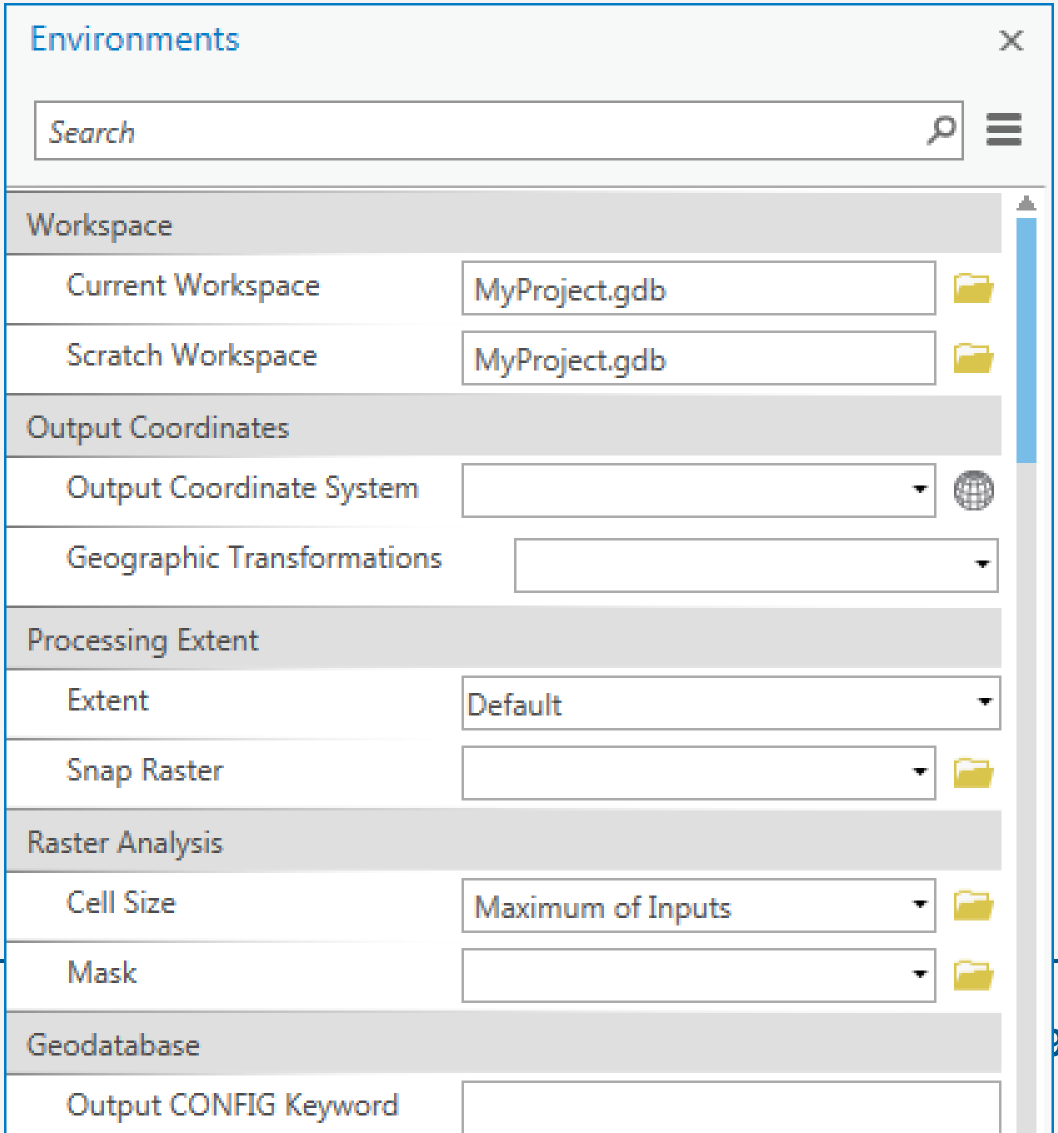
Environment settings are available from the `env` class as properties. These properties can be used to get the current environment values and set them.

To check all variables in the `env`, call:

```
print(arcpy.ListEnvironments())
```

See also: https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/using-environment-settings.htm

So, it is what you normally set in Environments



Environments

Search

Workspace

Current Workspace	MyProject.gdb
Scratch Workspace	MyProject.gdb

Output Coordinates

Output Coordinate System	
Geographic Transformations	

Processing Extent

Extent	Default
Snap Raster	

Raster Analysis

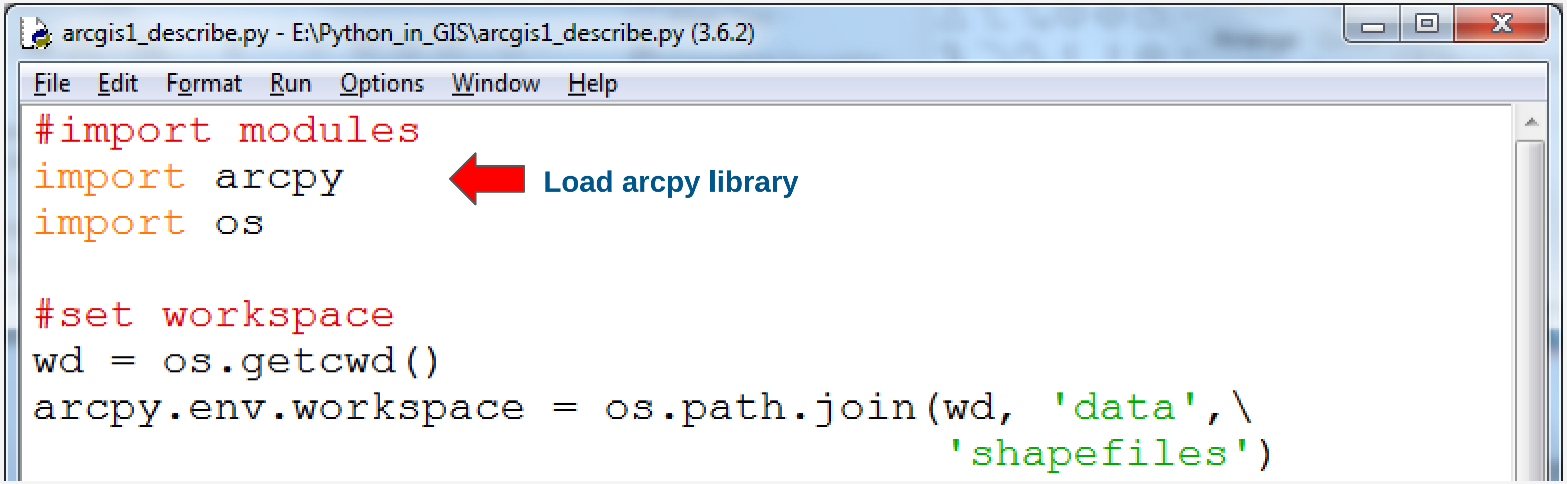
Cell Size	Maximum of Inputs
Mask	

Geodatabase

Output CONFIG Keyword	

load arcpy and set workspace

Environment settings are properties of ArcPy's `env` class. These properties can be used to retrieve (get) current environment settings or to change (set) them.



arcgis1_describe.py - E:\Python_in_GIS\arcgis1_describe.py (3.6.2)
File Edit Format Run Options Window Help

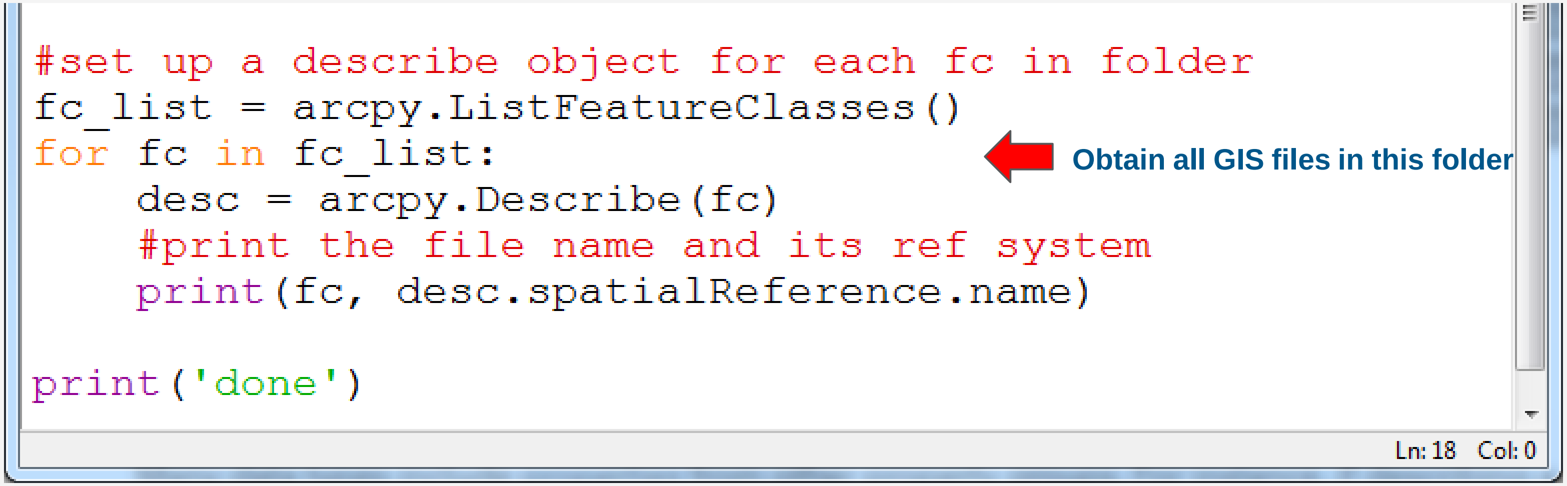
```
#import modules
import arcpy
import os

#set workspace
wd = os.getcwd()
arcpy.env.workspace = os.path.join(wd, 'data',\
                                   'shapefiles')
```

Load arcpy library

Then add rest of script (e.g. get ref system)

Describe function:

- Provides the meta data of a layer.
- Returns a Describe object, with multiple properties, such as data type, fields, indexes, and many others.
- Its properties are dynamic, depending on what data type is described.



```
#set up a describe object for each fc in folder
fc_list = arcpy.ListFeatureClasses()
for fc in fc_list:
    desc = arcpy.Describe(fc)
    #print the file name and its ref system
    print(fc, desc.spatialReference.name)

print('done')
```

Obtain all GIS files in this folder

Running tools

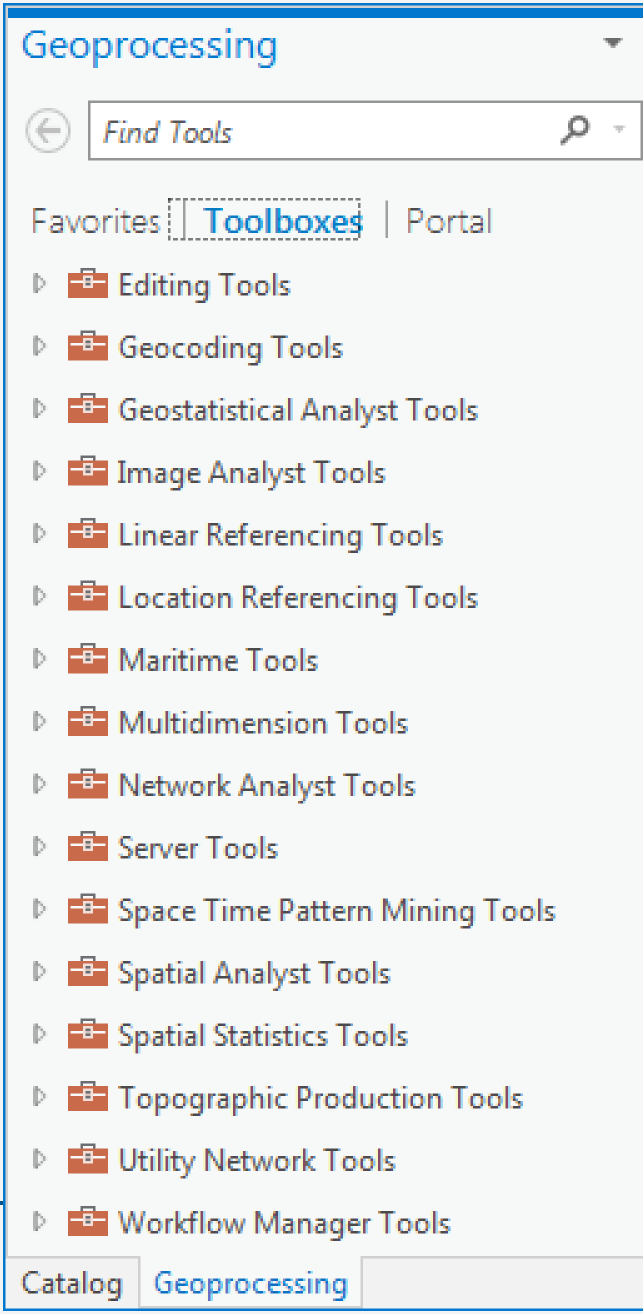
Modules

arcpy includes a series of modules (sub-libraries), such as:

- a data access module (arcpy.da), and
- a mapping module (arcpy.mp),

In addition, all toolboxes are available as modules!

- e.g. "Network Analyst" module (arcpy.na)
- e.g. “Spatial Analyst" module (arcpy.sa)



Geoprocessing

Find Tools

Favorites | Toolboxes | Portal

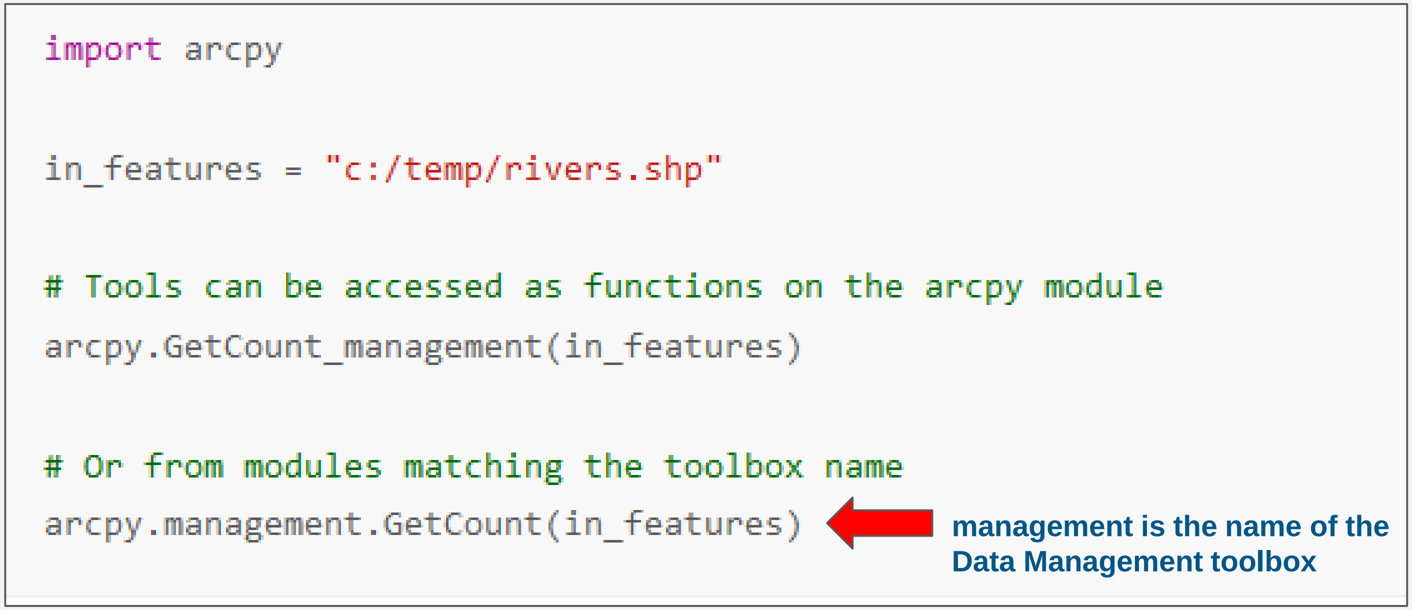
- Editing Tools
- Geocoding Tools
- Geostatistical Analyst Tools
- Image Analyst Tools
- Linear Referencing Tools
- Location Referencing Tools
- Maritime Tools
- Multidimension Tools
- Network Analyst Tools
- Server Tools
- Space Time Pattern Mining Tools
- Spatial Analyst Tools
- Spatial Statistics Tools
- Topographic Production Tools
- Utility Network Tools
- Workflow Manager Tools

Catalog | Geoprocessing

Two ways to access tool functions

Geoprocessing tools are organized in two different ways (matter of preference):

- as functions directly on arcpy
- as functions in the (toolbox) modules



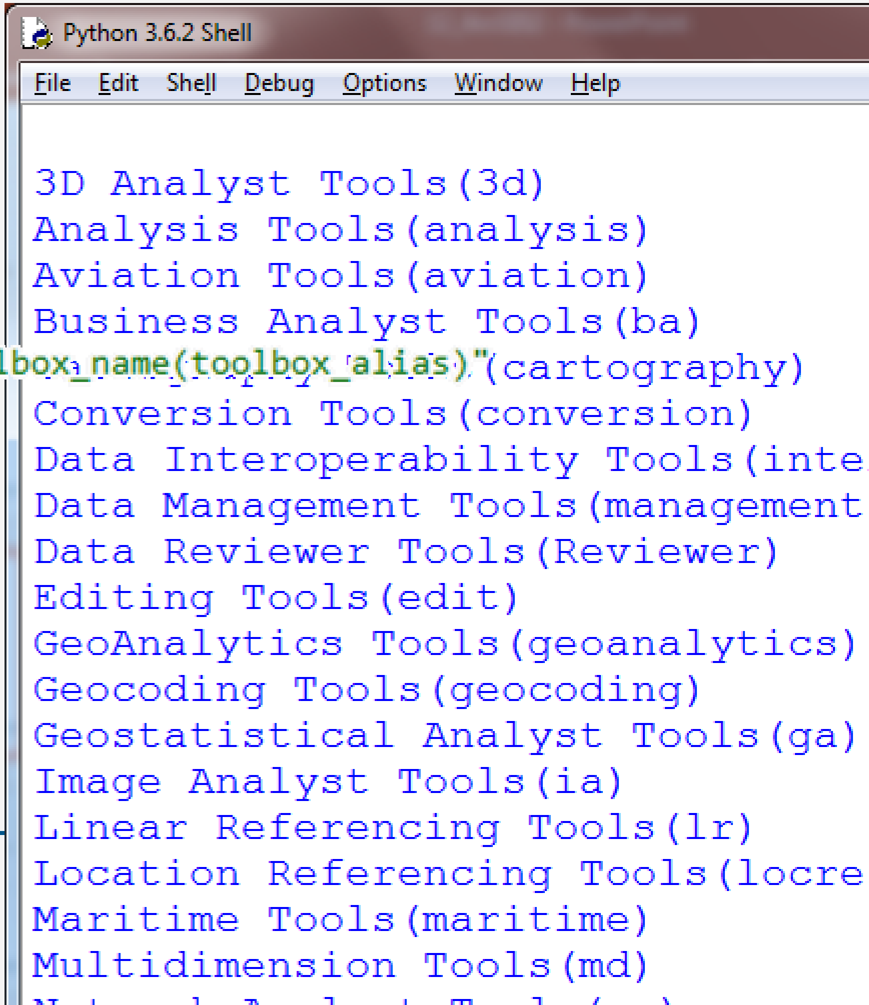
```
import arcpy

in_features = "c:/temp/rivers.shp"

# Tools can be accessed as functions on the arcpy module
arcpy.GetCount_management(in_features)

# Or from modules matching the toolbox name
arcpy.management.GetCount(in_features)
```

management is the name of the Data Management toolbox

If you need to know the toolbox name:

Printed in brackets is the name to use in Python

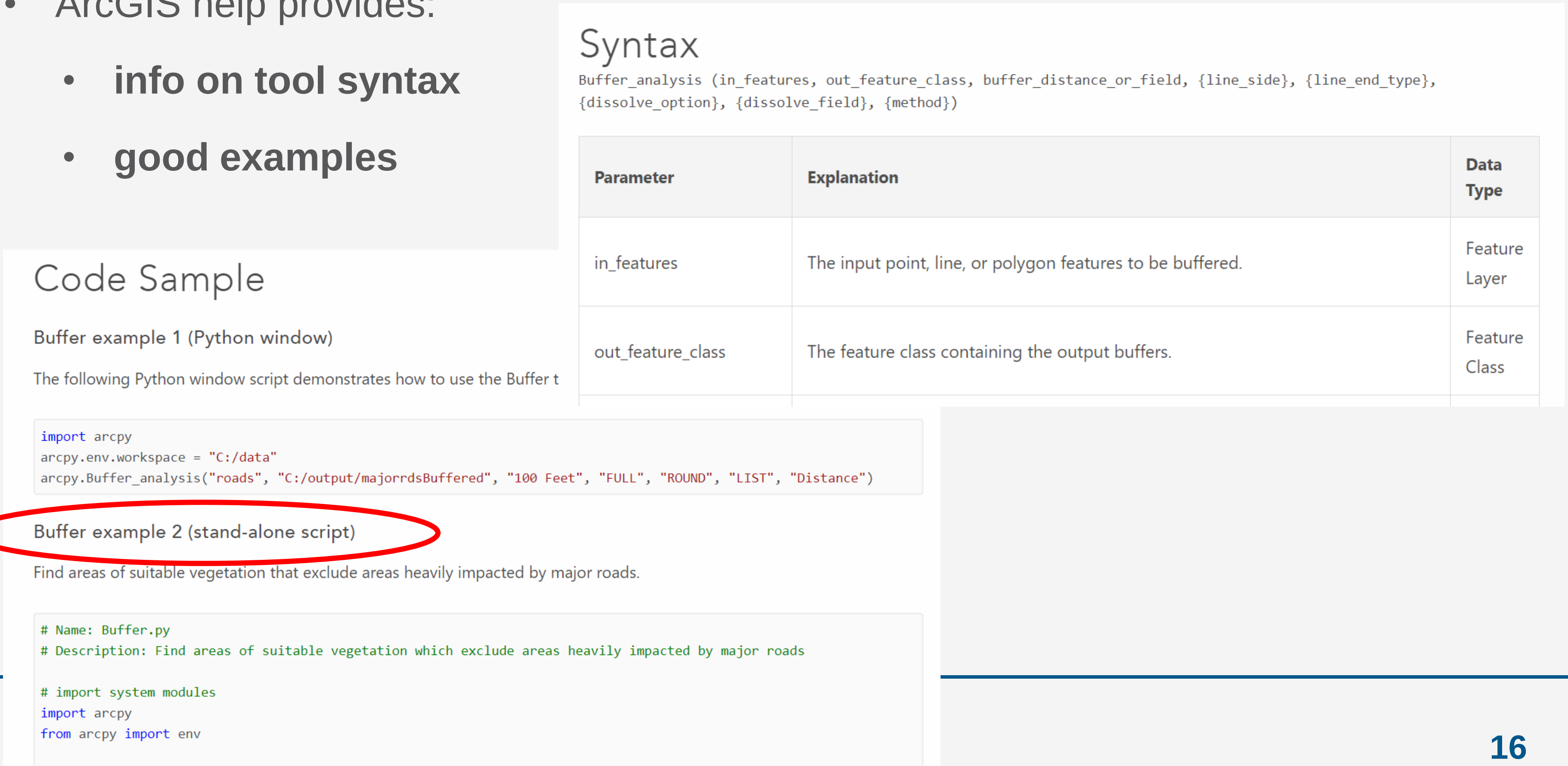
```
import arcpy

# Print all current toolboxes
#
for toolbox in arcpy.ListToolboxes():
    # Toolboxes are printed in the form of "toolbox_name(toolbox_alias)"
    print(toolbox)
```



```
3D Analyst Tools(3d)
Analysis Tools(analysis)
Aviation Tools(aviation)
Business Analyst Tools(ba)
Cartography Tools(cartography)
Conversion Tools(conversion)
Data Interoperability Tools(inte
Data Management Tools(management
Data Reviewer Tools(Reviewer)
Editing Tools(edit)
GeoAnalytics Tools(geoanalytics)
Geocoding Tools(geocoding)
Geostatistical Analyst Tools(ga)
Image Analyst Tools(ia)
Linear Referencing Tools(lr)
Location Referencing Tools(locre
Maritime Tools(maritime)
Multidimension Tools(md)
```

Running tools

- The tools that you normally use in ArcMap / ArcGIS Pro are available in Python
- ArcGIS help provides:
  - **info on tool syntax**
  - **good examples**



Syntax

```
Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field, {line_side}, {line_end_type},
{dissolve_option}, {dissolve_field}, {method})
```

Parameter	Explanation	Data Type
in_features	The input point, line, or polygon features to be buffered.	Feature Layer
out_feature_class	The feature class containing the output buffers.	Feature Class

Code Sample

Buffer example 1 (Python window)

The following Python window script demonstrates how to use the Buffer t

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

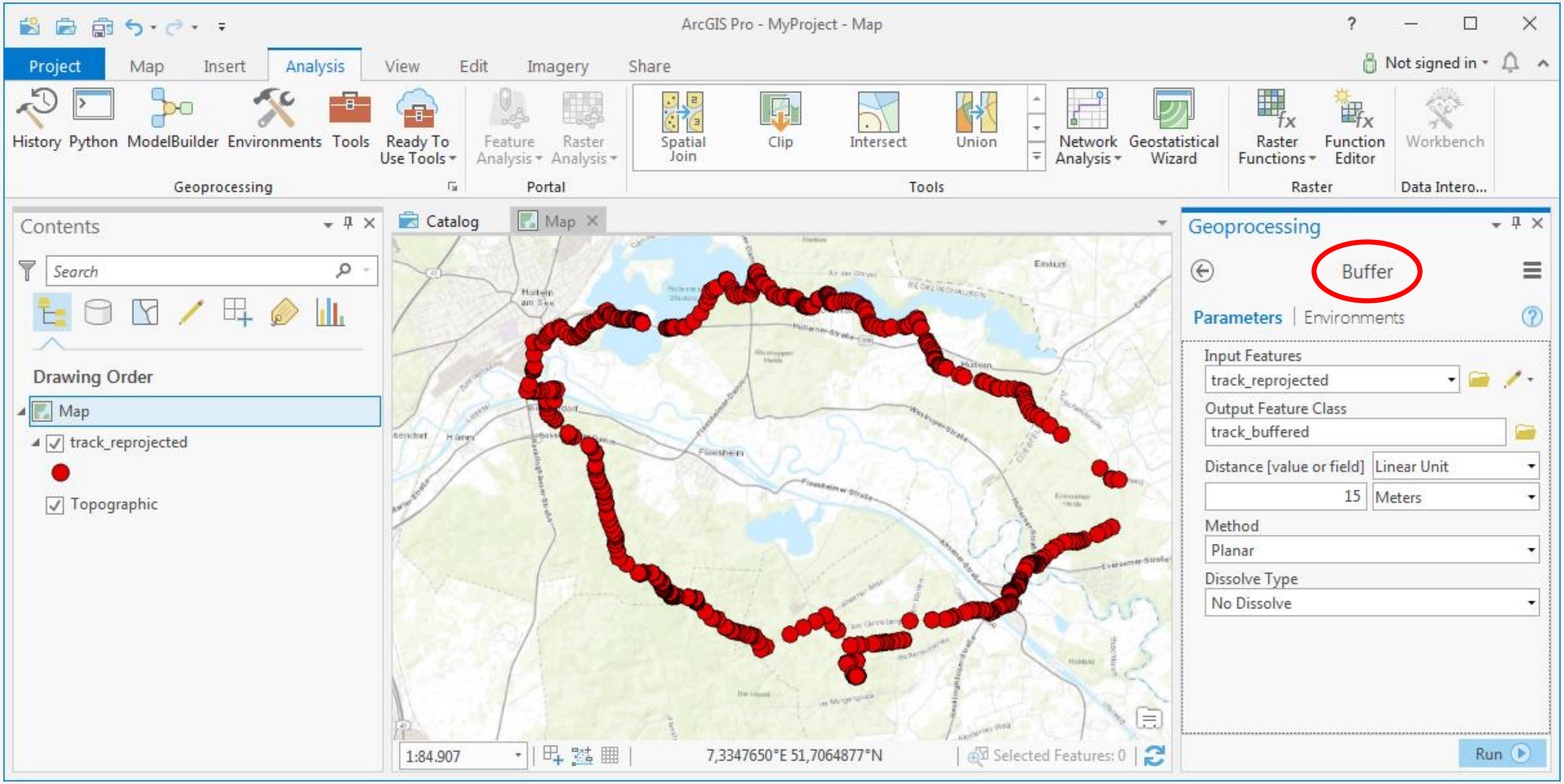
Buffer example 2 (stand-alone script)

Find areas of suitable vegetation that exclude areas heavily impacted by major roads.

```
# Name: Buffer.py
# Description: Find areas of suitable vegetation which exclude areas heavily impacted by major roads

# import system modules
import arcpy
from arcpy import env
```

Handy: obtain Python code snippets from tool results

Run a tool in the GUI, e.g. a buffer



ArcGIS Pro - MyProject - Map

Geoprocessing – Buffer

Parameters | Environments

Input Features: track_reprojected

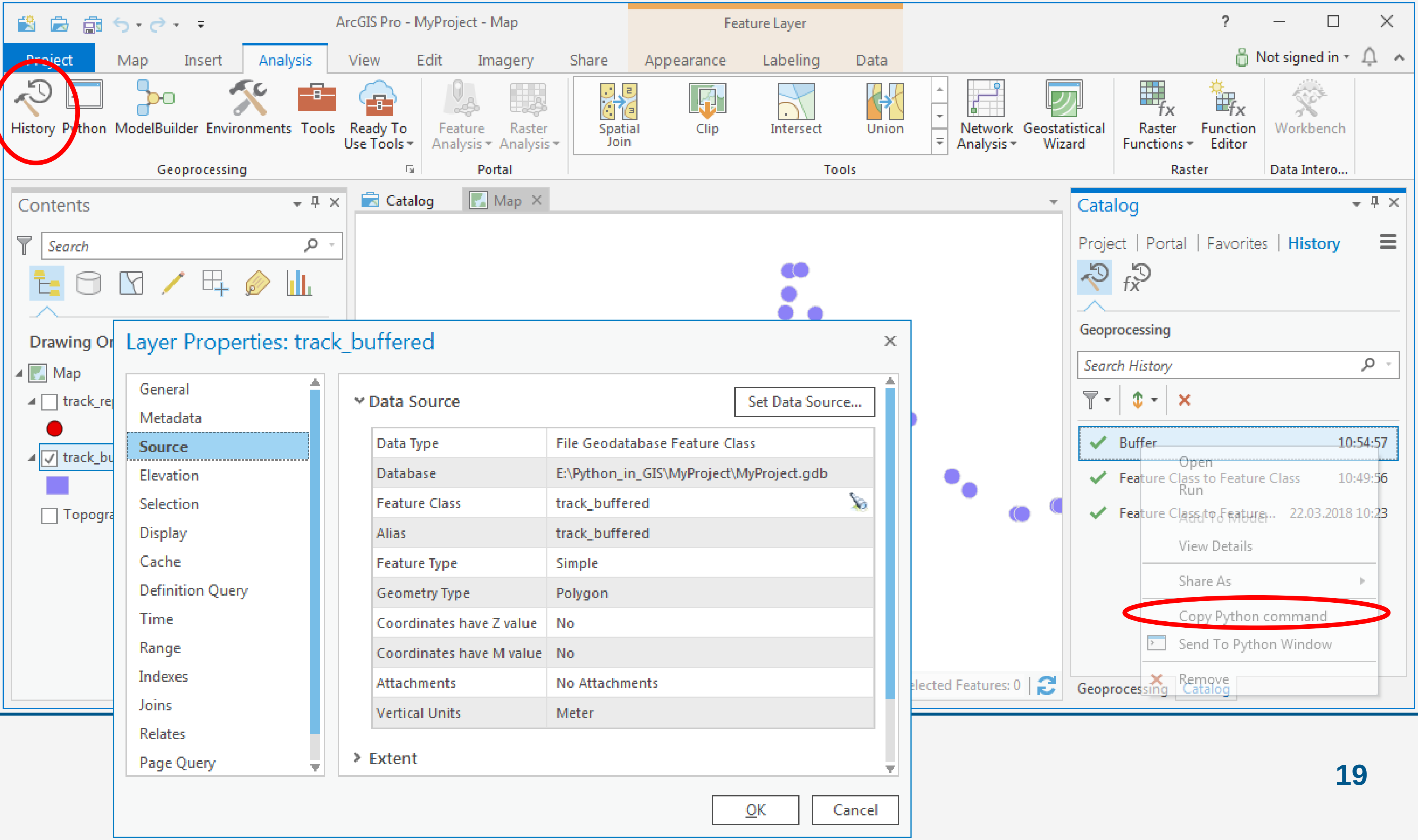
Output Feature Class: track_buffered

Distance [value or field]: Linear Unit – 15 Meters

Method: Planar

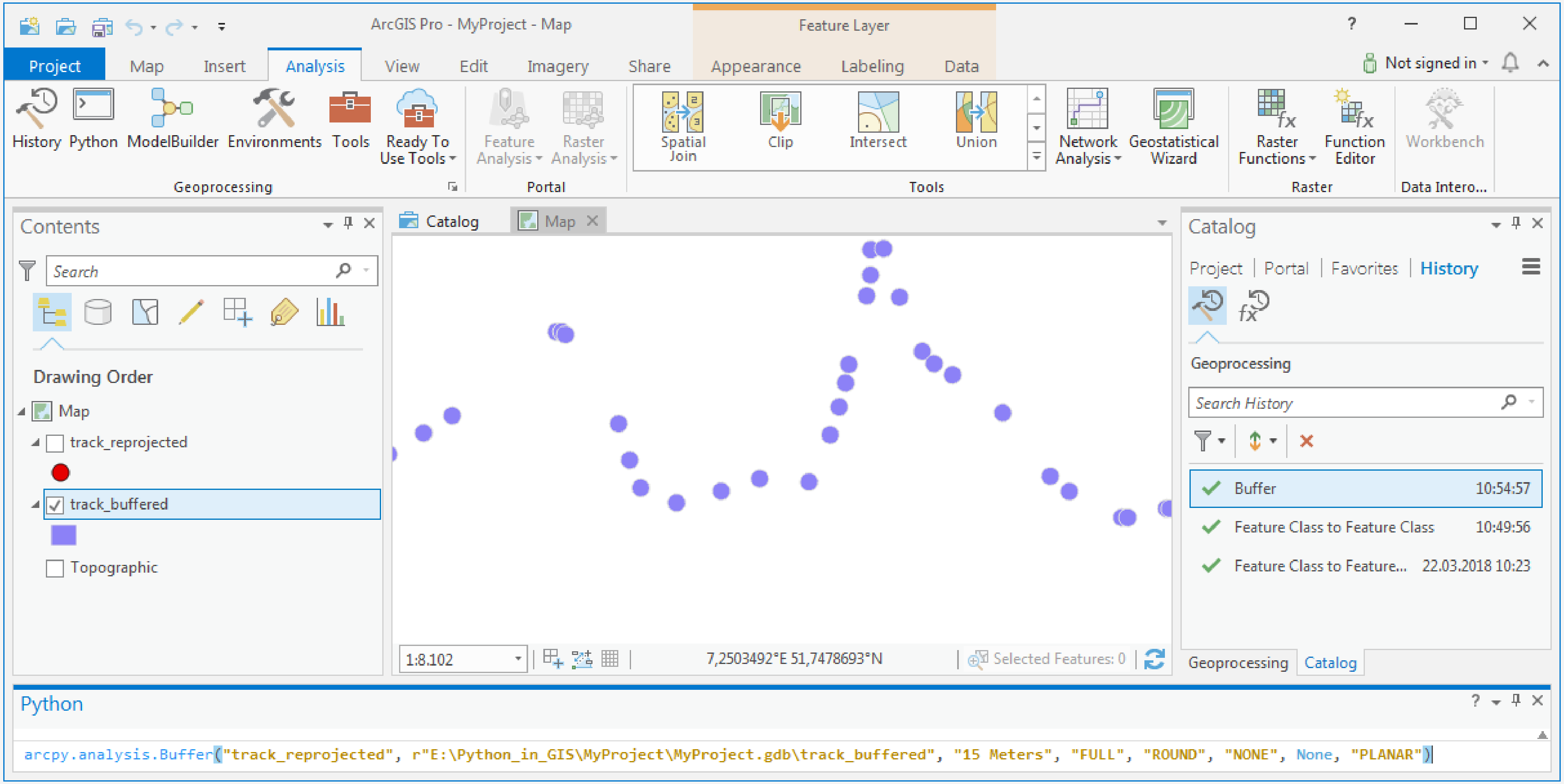
Dissolve Type: No Dissolve

Check, go to History, Copy Python command



Data Type	File Geodatabase Feature Class
Database	E:\Python_in_GIS\MyProject\MyProject.gdb
Feature Class	track_buffered
Alias	track_buffered
Feature Type	Simple
Geometry Type	Polygon
Coordinates have Z value	No
Coordinates have M value	No
Attachments	No Attachments
Vertical Units	Meter

Paste code somewhere (e.g. Python window)

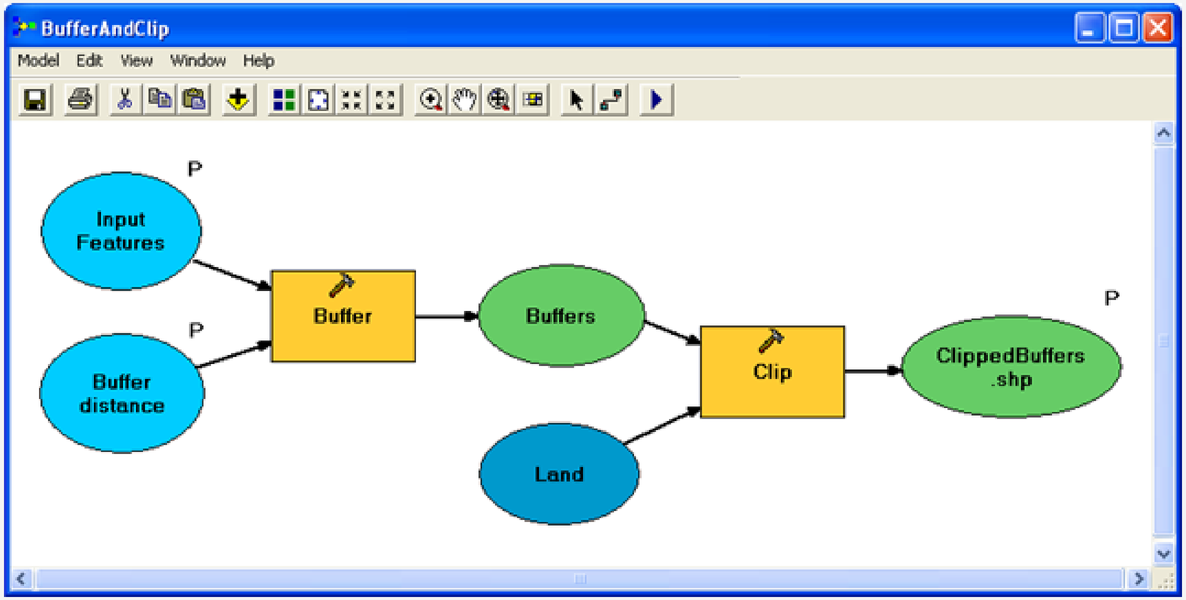


✓ Buffer	10:54:57
✓ Feature Class to Feature Class	10:49:56
✓ Feature Class to Feature...	22.03.2018 10:23

```
arcpy.analysis.Buffer("track_reprojected", r"E:\Python_in_GIS\MyProject\MyProject.gdb\track_buffered", "15 Meters", "FULL", "ROUND", "NONE", None, "PLANAR")
```

From model builder to Python script

- In ArcMap, it was possible to generate Python scripts from the model builder
- This ability was removed from ArcGIS Pro 1.1 onward and ESRI does not plan to add it back...



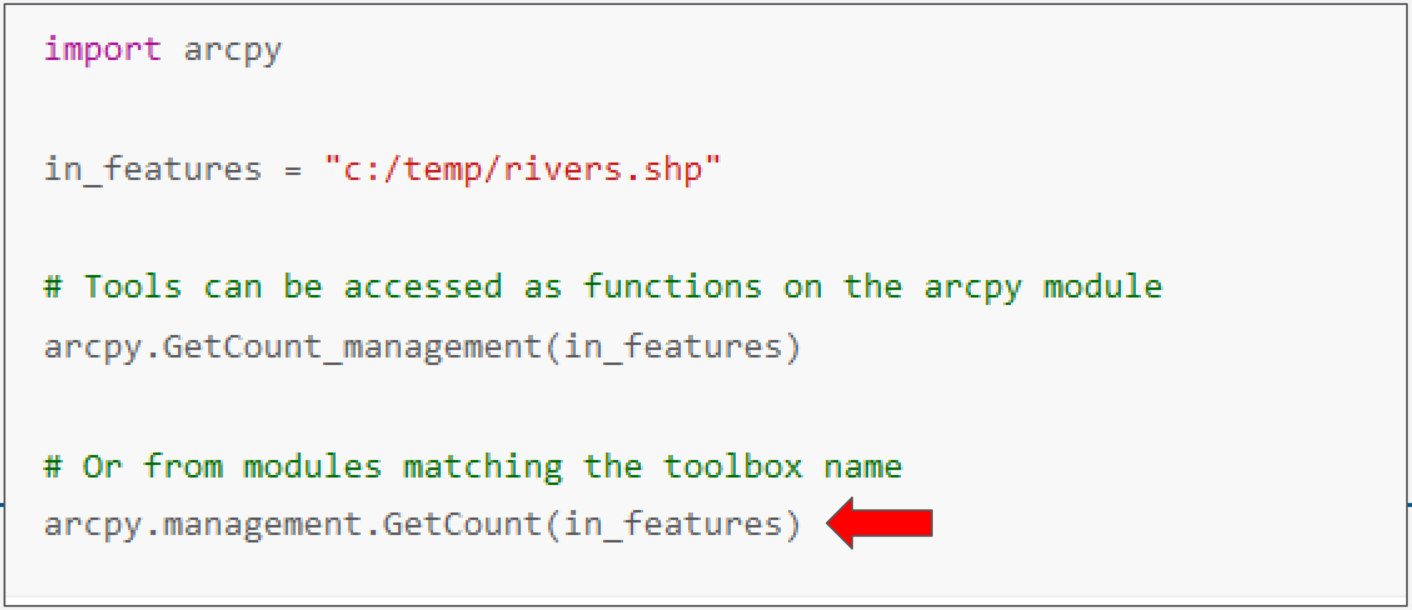
Licenses

Toolboxes that require a license

Tools from ArcGIS extensions, such as the Spatial Analyst toolbox, require an additional license.

If the necessary licenses are not available, a tool fails and returns error messages.

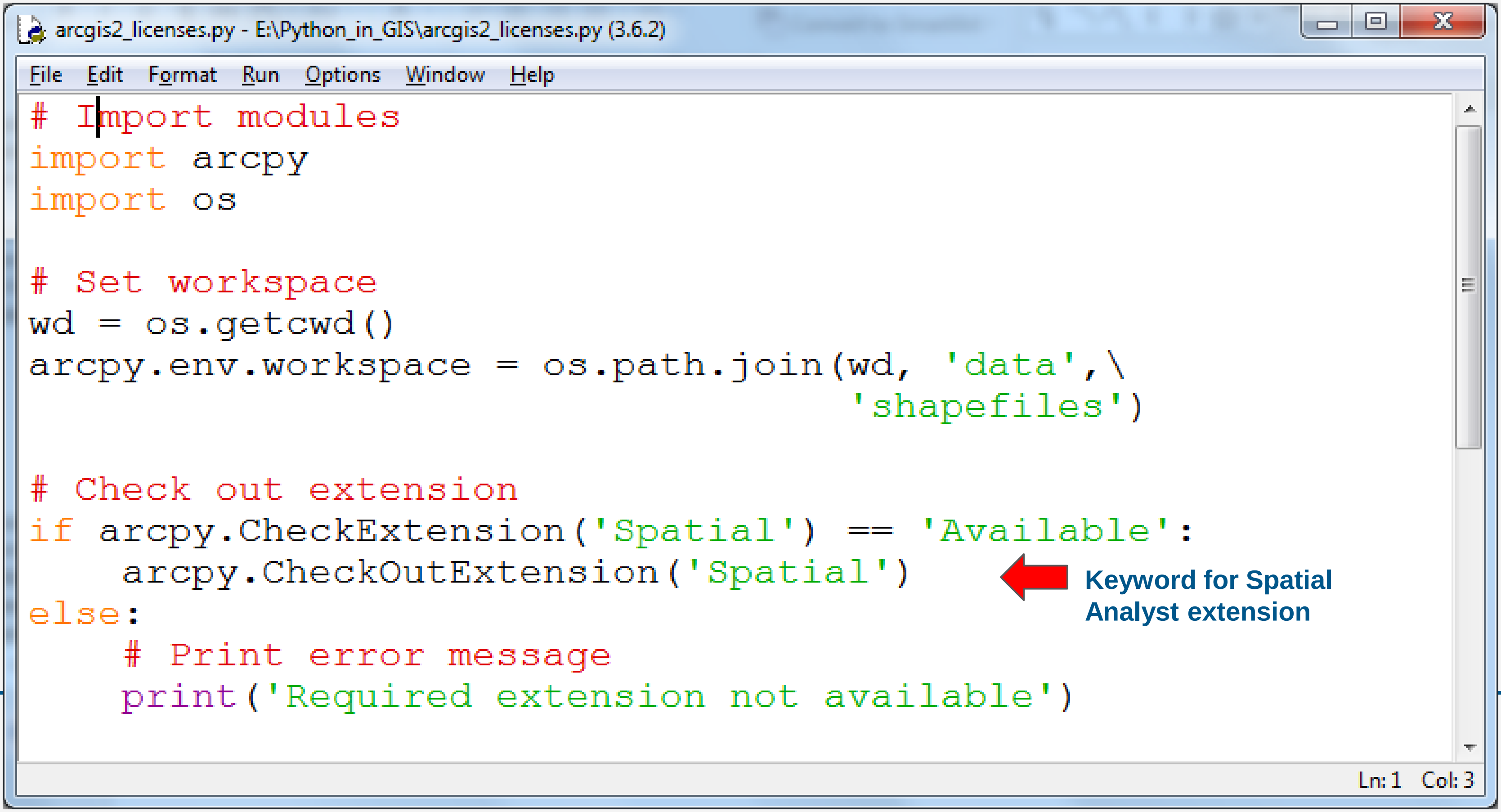
Tools from licensed toolboxes are available in the toolbox module only, not from arcpy directly!



```
import arcpy

in_features = "c:/temp/rivers.shp"

# Tools can be accessed as functions on the arcpy module
arcpy.GetCount_management(in_features)

# Or from modules matching the toolbox name
arcpy.management.GetCount(in_features)
```

Checking extensions out and in (1)

Find the correct keyword for the extension here: https://pro.arcgis.com/en/pro-app/arcpy/functions/checkoutextension.htm



arcgis2_licenses.py - E:\Python_in_GIS\arcgis2_licenses.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 1 Col: 3

```python
# Import modules
import arcpy
import os

# Set workspace
wd = os.getcwd()
arcpy.env.workspace = os.path.join(wd, 'data',\
                                   'shapefiles')

# Check out extension
if arcpy.CheckExtension('Spatial') == 'Available':
    arcpy.CheckOutExtension('Spatial')
else:
    # Print error message
    print('Required extension not available')
```

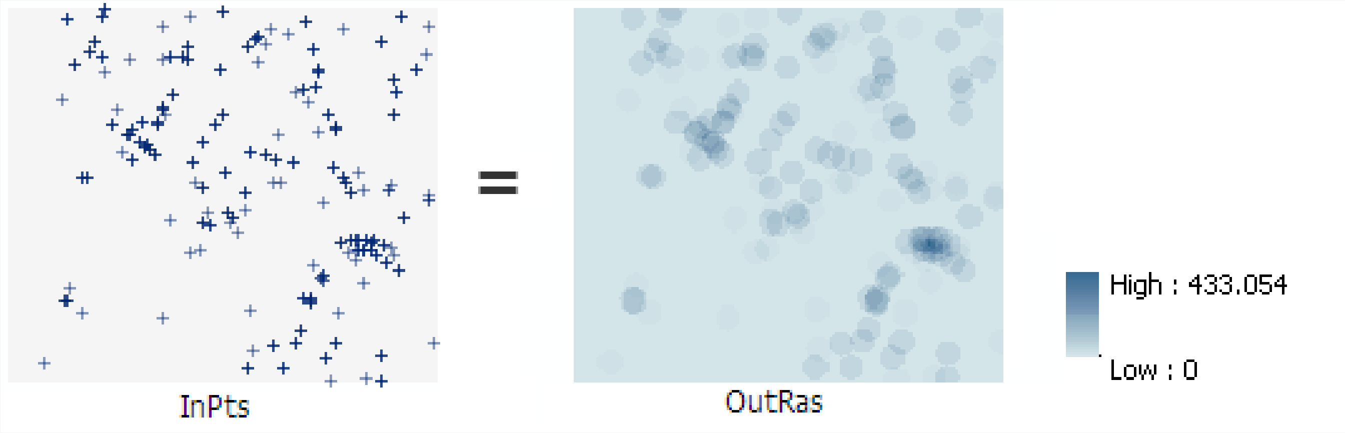
Keyword for Spatial Analyst extension

Checking extensions out and in (2)

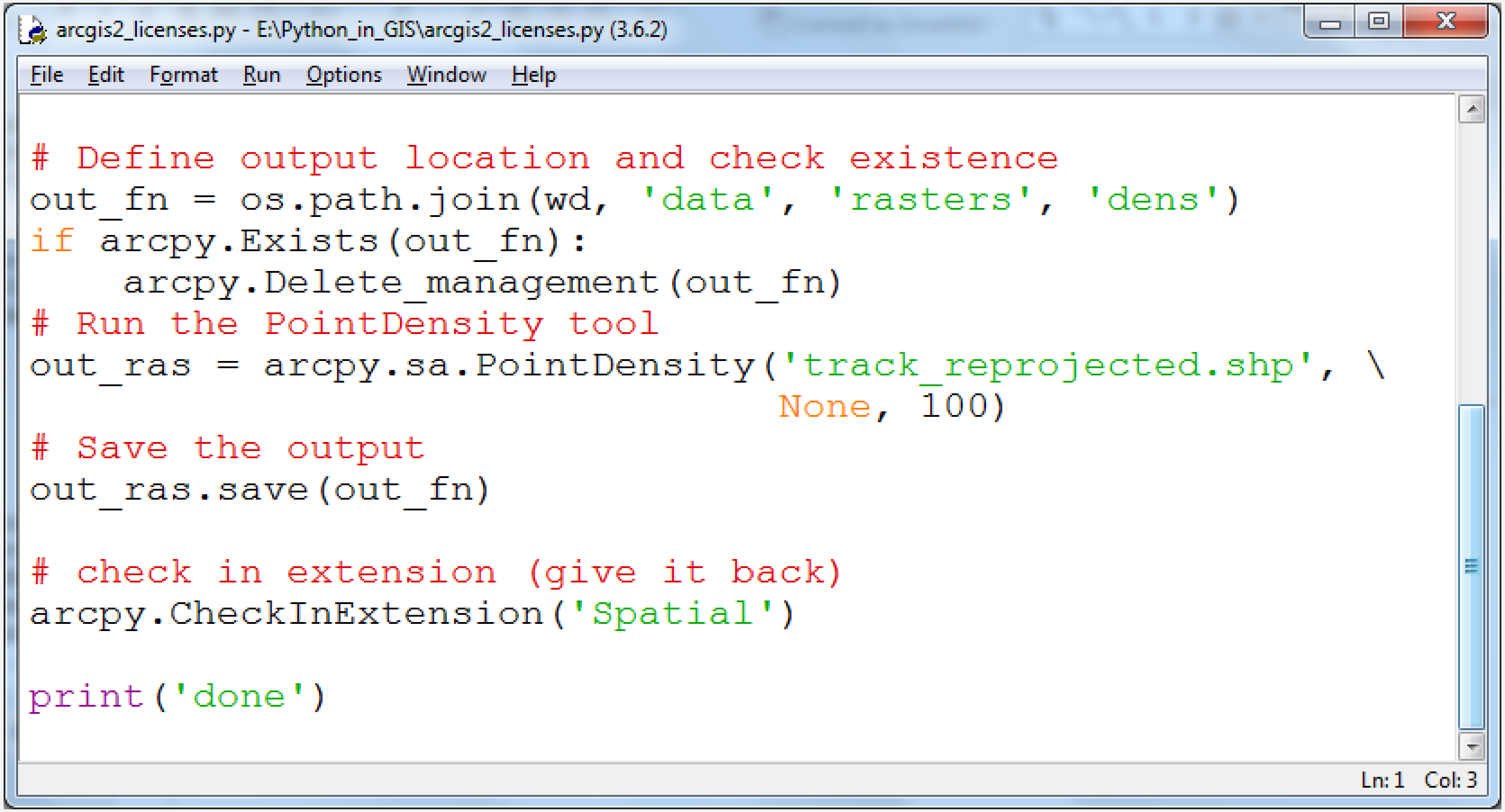
`Point Density` calculates a magnitude-per-unit area from point features that fall within a neighborhood around each cell → heat map

Syntax:

```
PointDensity (in_point_features, population_field, {cell_size},
{neighborhood}, {area_unit_scale_factor})
```



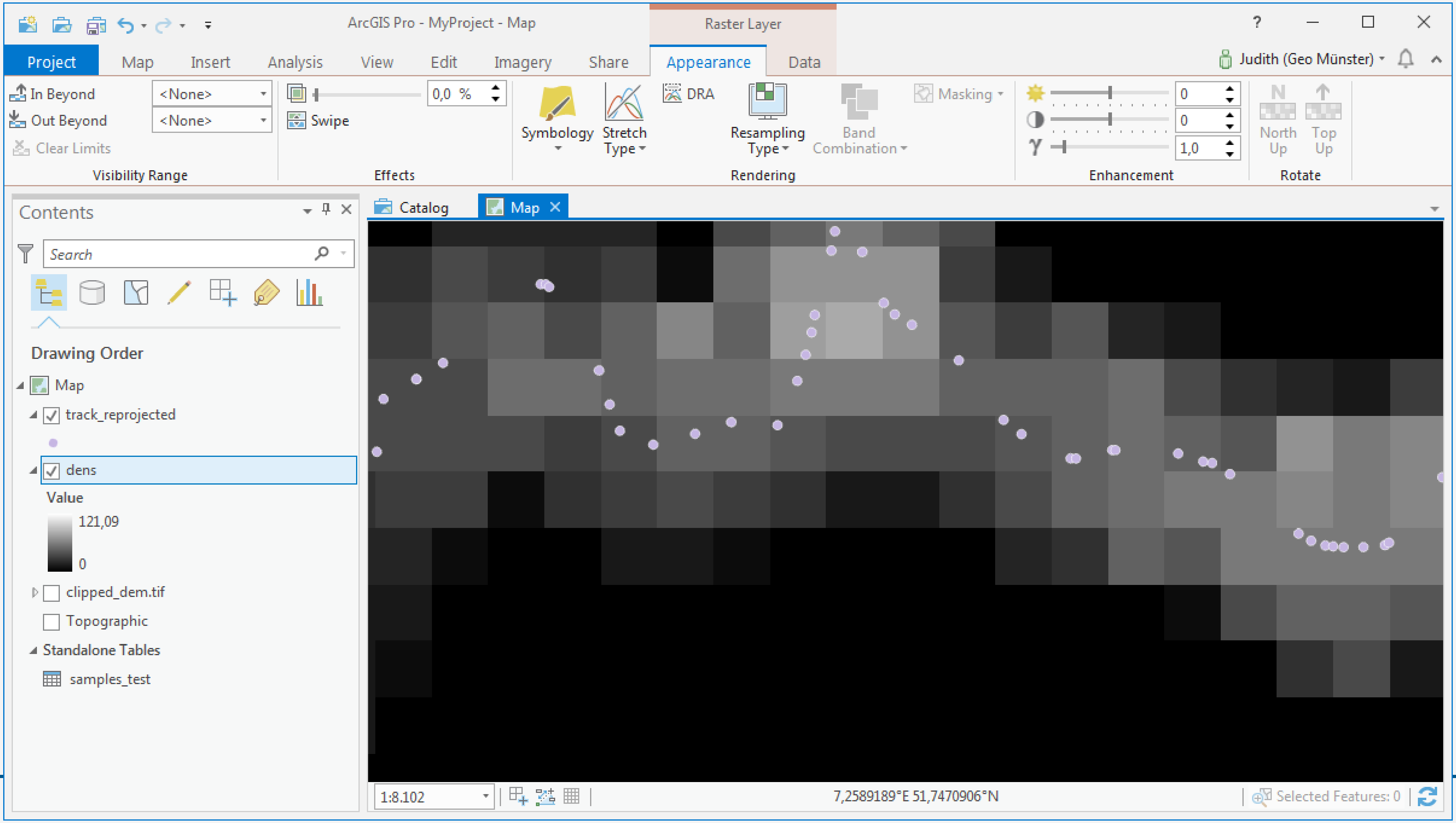
Checking extensions out and in (3)



arcgis2_licenses.py - E:\Python_in_GIS\arcgis2_licenses.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 1 Col: 3

```
# Define output location and check existence
out_fn = os.path.join(wd, 'data', 'rasters', 'dens')
if arcpy.Exists(out_fn):
    arcpy.Delete_management(out_fn)
# Run the PointDensity tool
out_ras = arcpy.sa.PointDensity('track_reprojected.shp', \
                                 None, 100)
# Save the output
out_ras.save(out_fn)

# check in extension (give it back)
arcpy.CheckInExtension('Spatial')

print('done')
```

Result


ArcGIS Pro - MyProject - Map
Raster Layer
Project
Map
Insert
Analysis
View
Edit
Imagery
Share
Appearance
Data
Judith (Geo Münster)
In Beyond
Out Beyond
Clear Limits
<None>
<None>
Visibility Range
0,0 %
Swipe
Effects
Symbology
Stretch Type
DRA
Resampling Type
Band Combination
Masking
Rendering
0
0
1,0
Enhancement
North Up
Top Up
Rotate
Contents
Search
Drawing Order
Map
track_reprojected
dens
Value
121,09
0
clipped_dem.tif
Topographic
Standalone Tables
samples_test
Catalog
Map
1:8.102
7,2589189°E 51,7470906°N
Selected Features: 0

Note

When ArcGIS Pro uses licensing through the organization, the available product level and extensions are set by your account and available automatically from arcpy.

With a Concurrent Use license in ArcGIS Pro and with other ArcGIS Desktop applications, specific functions (including `CheckOutExtension` and `CheckInExtension`) are required to access certain tools.

For transferability reasons, it is always better to explicitly otain the license.

Exercise #1

We want to compare the elevation in the GPS track (measured by the GPS) with the elevation from the DEM at the same locations.

Make an arcpy tool script to add the elevation at the GPS points from the DEM to the shapefile as a field, such that the attribute table of the shapefile looks something like:

FID	…	ele	…	ele_raster
0		86		90
1		81		85
2		79		80
3		79		80
4		78		80
5		80		78

Hint: First search for an ArcGIS tool to do this, and test it manually if you want

Cursors

What is a cursor?

A cursor is a data access object that can be used to

- iterate over the set of rows in a table and read the attribute values
- edit rows in a table.

Cursors have three forms:

Aim of the cursor	ArcPy function
retrieve rows	`SearchCursor()`
insert rows	`InsertCursor()`
update or delete rows	`UpdateCursor()`

Cursors move only forward

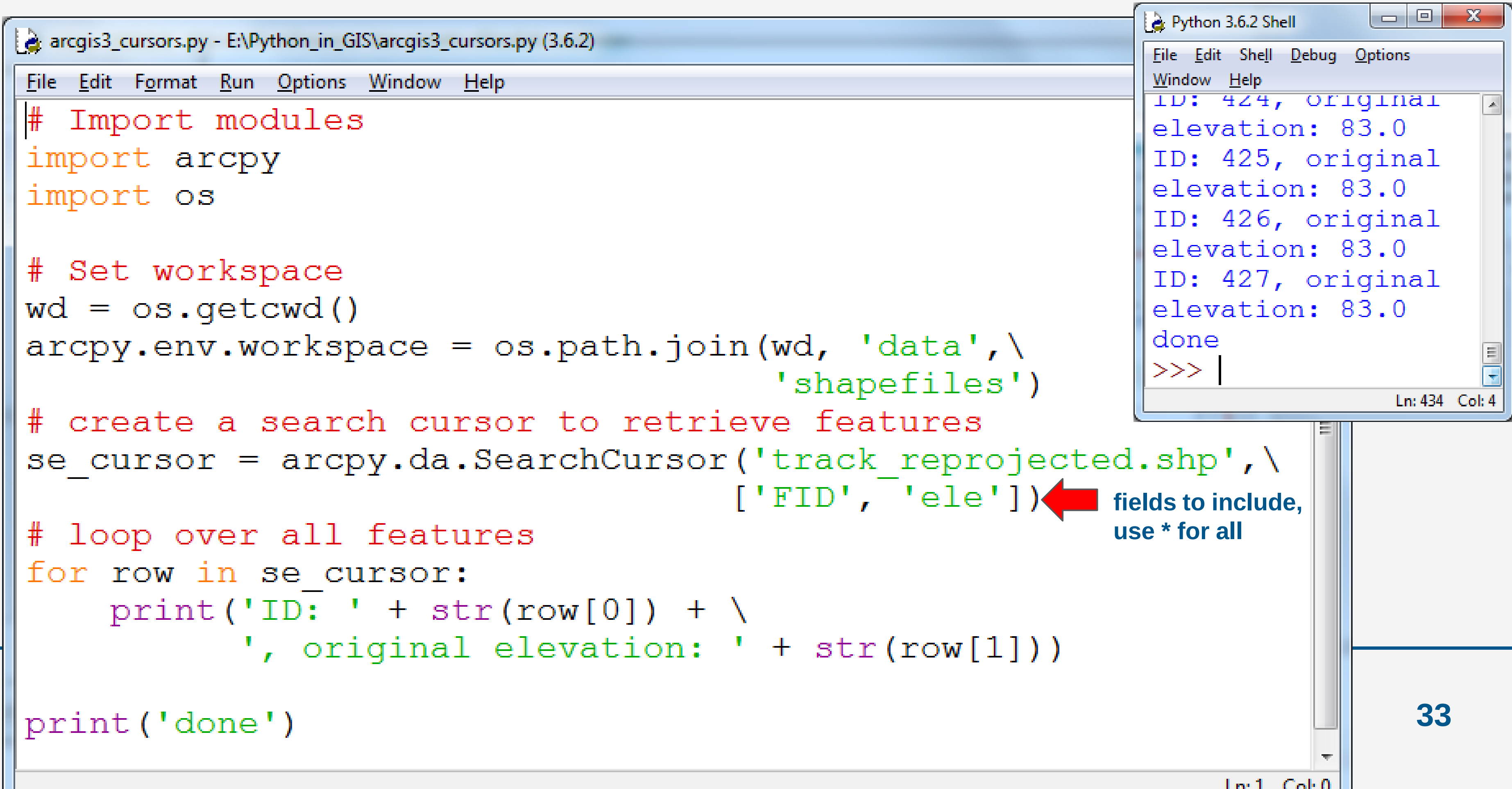
Cursors can only be navigated in a forward direction; they do not support backing up and retrieving rows that have already been accessed.

If a script needs to make multiple passes over the data, you may do that by:

- Using the cursor's `reset()` method`;`
- Deleting the cursor object and creating a new one; or
- Using the `with` statement to automatically close → as for `open()` for files

Search cursors

Syntax: `SearchCursor(dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})`



arcgis3_cursors.py - E:\Python_in_GIS\arcgis3_cursors.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 1 Col: 0
Python 3.6.2 Shell
File Edit Shell Debug Options
Window Help
Ln: 434 Col: 4

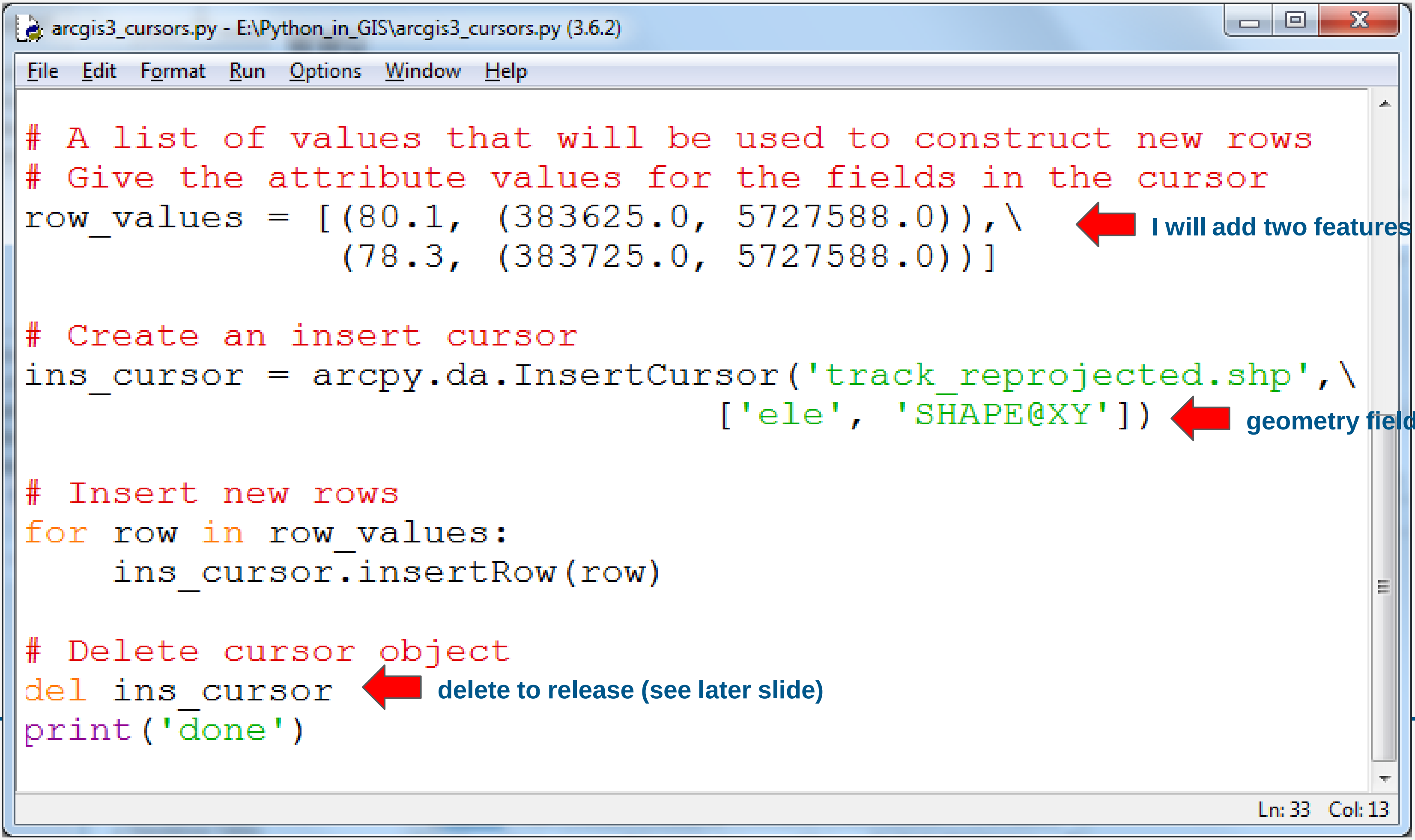
```
# Import modules
import arcpy
import os

# Set workspace
wd = os.getcwd()
arcpy.env.workspace = os.path.join(wd, 'data',\
                                   'shapefiles')
# create a search cursor to retrieve features
se_cursor = arcpy.da.SearchCursor('track_reprojected.shp',\
                                 ['FID', 'ele'])
# loop over all features
for row in se_cursor:
    print('ID: ' + str(row[0]) + \
          ', original elevation: ' + str(row[1]))

print('done')
```

```
ID: 424, original
elevation: 83.0
ID: 425, original
elevation: 83.0
ID: 426, original
elevation: 83.0
ID: 427, original
elevation: 83.0
done
>>> 
```

fields to include, use * for all

Insert cursors (1)

Syntax: `InsertCursor (dataset, {spatial_reference})`



arcgis3_cursors.py - E:\Python_in_GIS\arcgis3_cursors.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 33 Col: 13

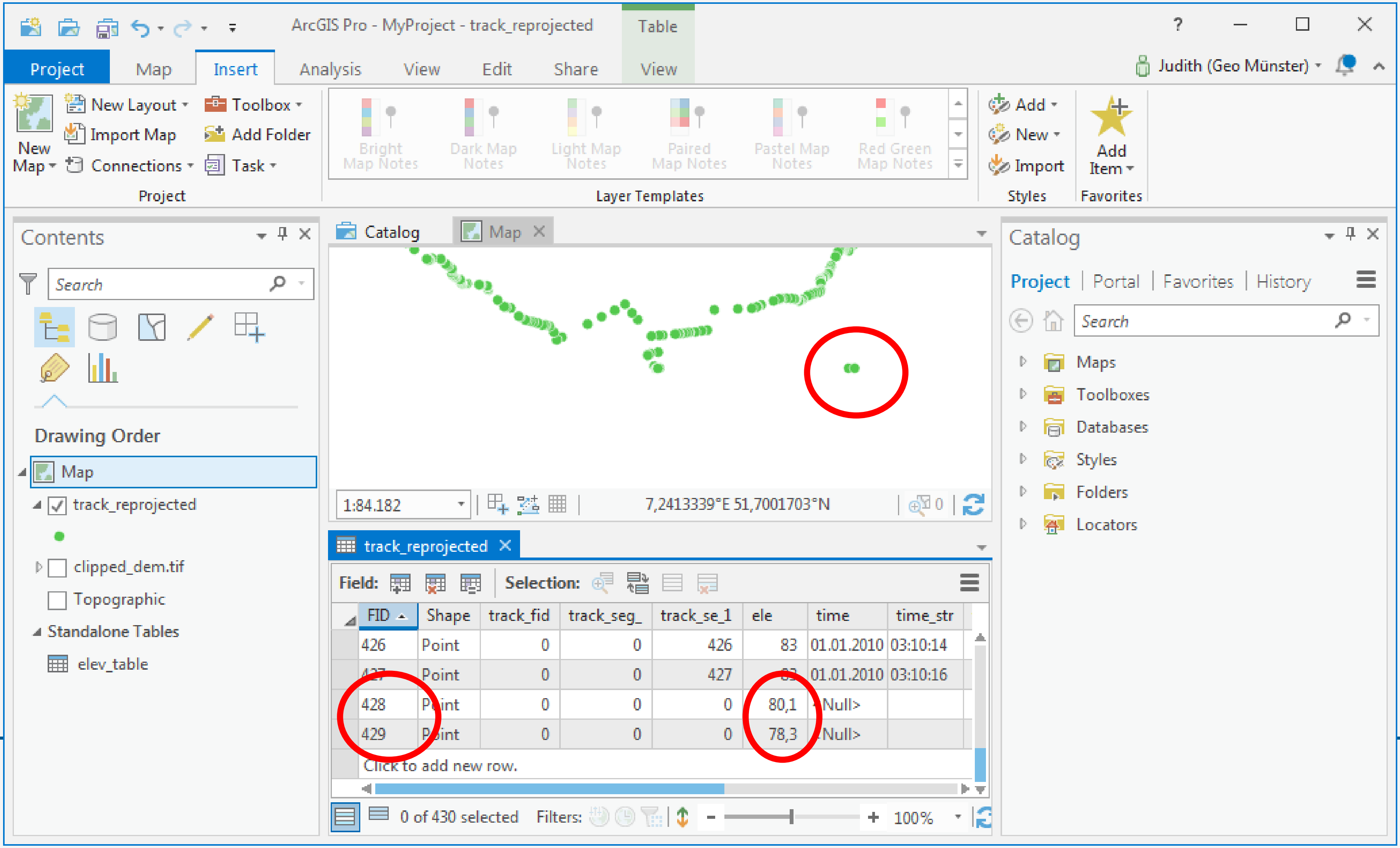
```
# A list of values that will be used to construct new rows
# Give the attribute values for the fields in the cursor
row_values = [(80.1, (383625.0, 5727588.0)),\
              (78.3, (383725.0, 5727588.0))]

# Create an insert cursor
ins_cursor = arcpy.da.InsertCursor('track_reprojected.shp',\
                                   ['ele', 'SHAPE@XY'])

# Insert new rows
for row in row_values:
    ins_cursor.insertRow(row)

# Delete cursor object
del ins_cursor
print('done')
```

I will add two features

geometry field

delete to release (see later slide)

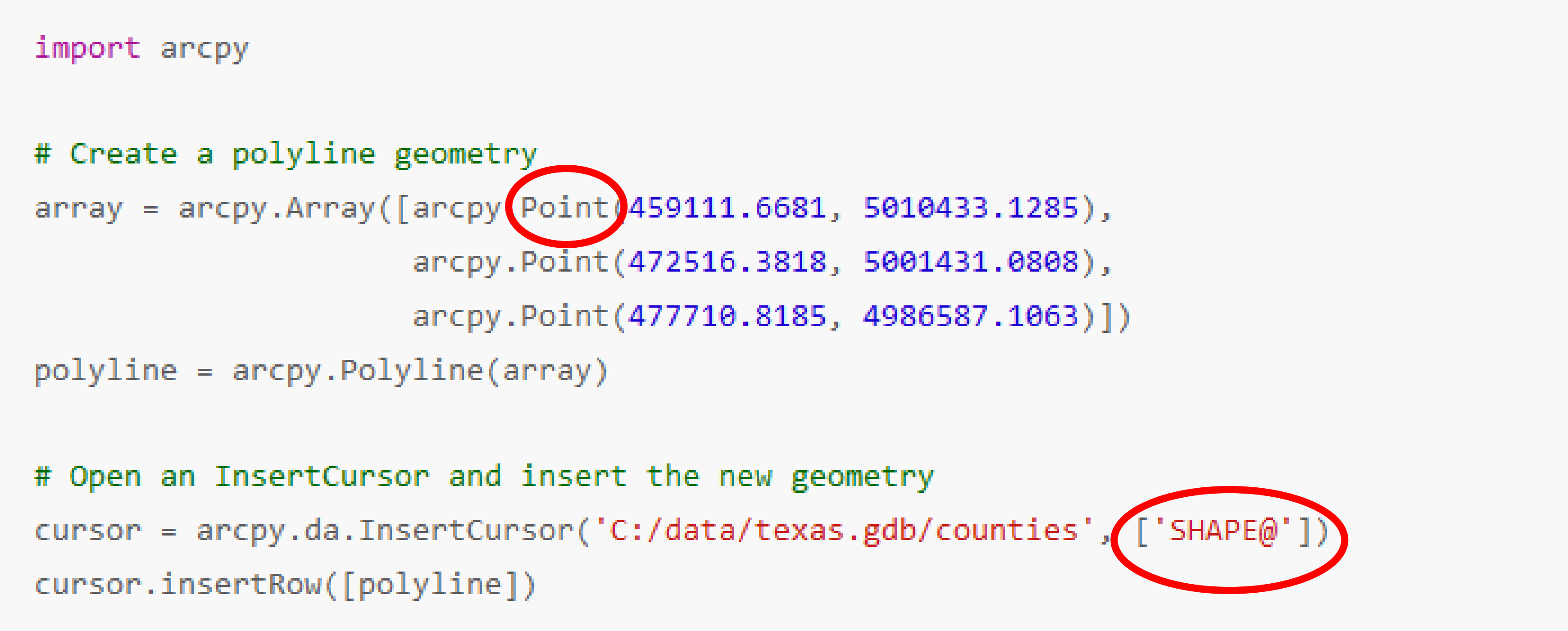
Insert cursors (2)



FID	Shape	track_fid	track_seg_	track_se_1	ele	time	time_str
426	Point	0	0	426	83	01.01.2010	03:10:14
427	Point	0	0	427	83	01.01.2010	03:10:16
428	Point	0	0	0	80,1	<Null>	
429	Point	0	0	0	78,3	<Null>	

Insert cursors (3)

When working with line or polygon datasets, you need to create a geometry object first, as in this example from the arcpy documentation:



```
import arcpy

# Create a polyline geometry
array = arcpy.Array([arcpy.Point(459111.6681, 5010433.1285),
                     arcpy.Point(472516.3818, 5001431.0808),
                     arcpy.Point(477710.8185, 4986587.1063)])
polyline = arcpy.Polyline(array)

# Open an InsertCursor and insert the new geometry
cursor = arcpy.da.InsertCursor('C:/data/texas.gdb/counties', ['SHAPE@'])
cursor.insertRow([polyline])
```

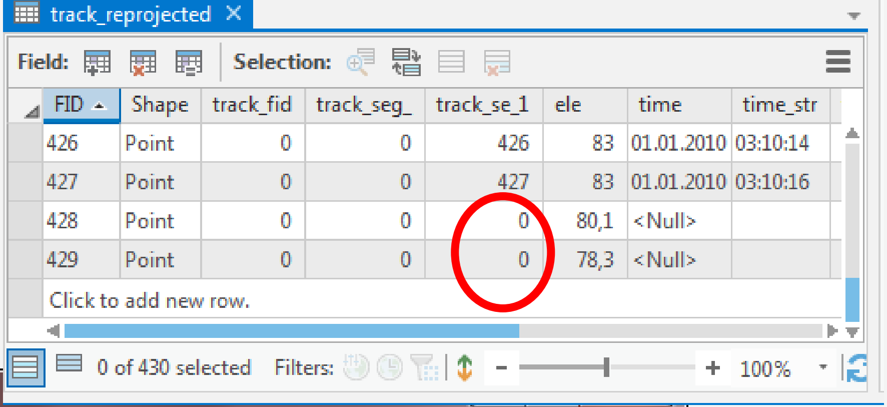
Update cursors (1)

Syntax: `UpdateCursor(dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})`

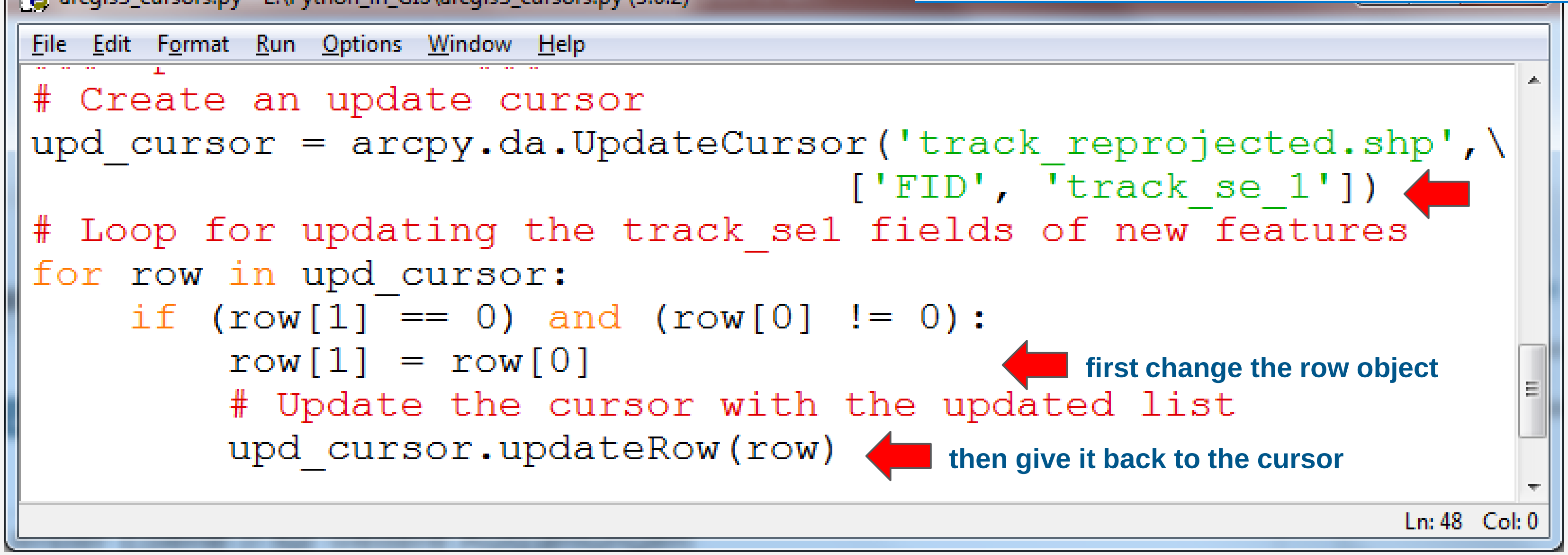
Methods of the UpdateCursor:

Method	Explanation
deleteRow ()	Deletes the current row.
reset ()	Resets the cursor back to the first row.
updateRow (row)	Updates the current row in the table.

Update cursors (2)



FID	Shape	track_fid	track_seg_	track_se_1	ele	time	time_str
426	Point	0	0	426	83	01.01.2010	03:10:14
427	Point	0	0	427	83	01.01.2010	03:10:16
428	Point	0	0	0	80,1	<Null>	
429	Point	0	0	0	78,3	<Null>	



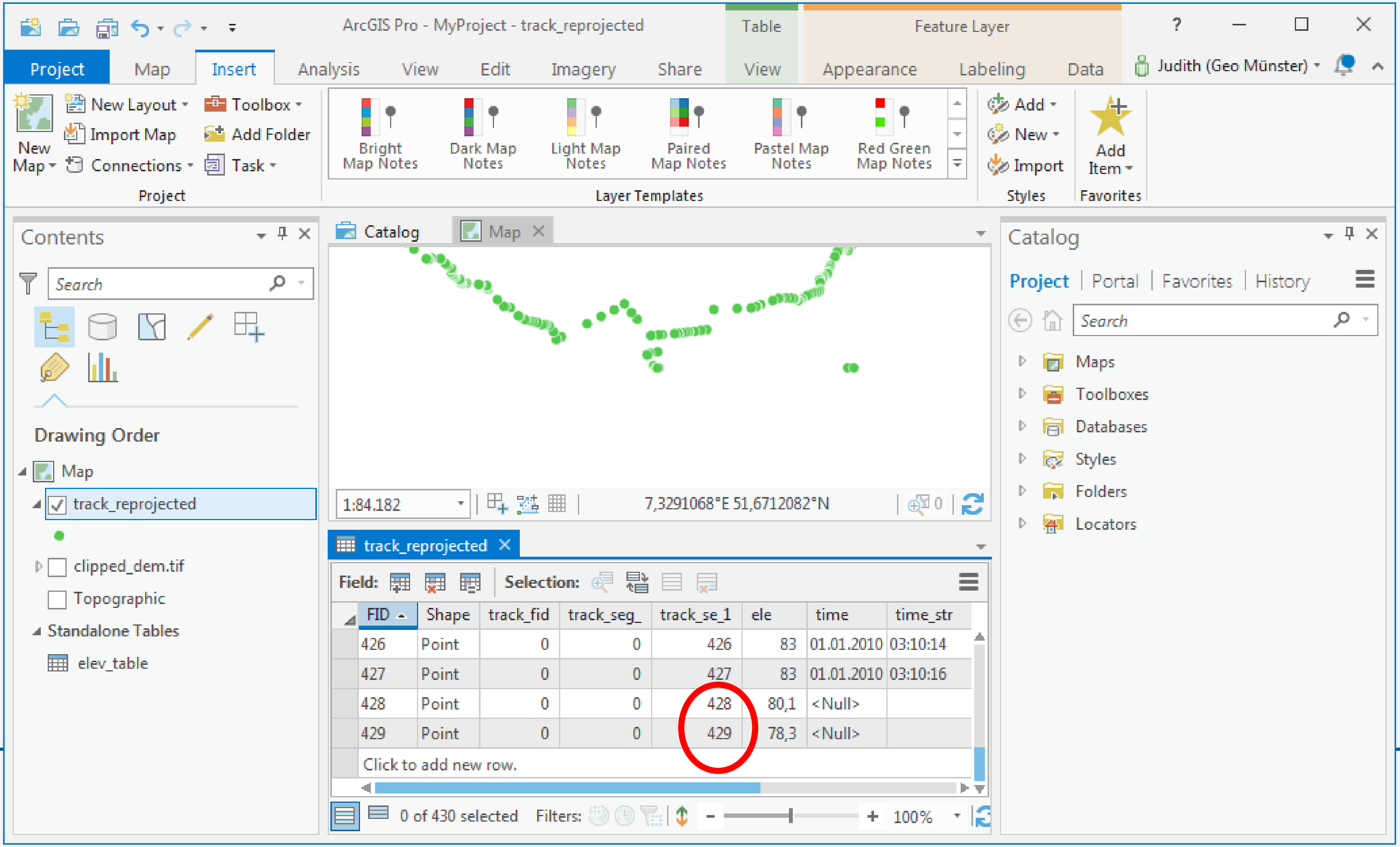
arcgis3_cursors.py - E:\Python_in_GIS\arcgis3_cursors.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 48 Col: 0

```
# Create an update cursor
upd_cursor = arcpy.da.UpdateCursor('track_reprojected.shp',\
                                 ['FID', 'track_se_1'])
# Loop for updating the track_se1 fields of new features
for row in upd_cursor:
    if (row[1] == 0) and (row[0] != 0):
        row[1] = row[0]
        # Update the cursor with the updated list
        upd_cursor.updateRow(row)
```

first change the row object

then give it back to the cursor

Update cursors (3)

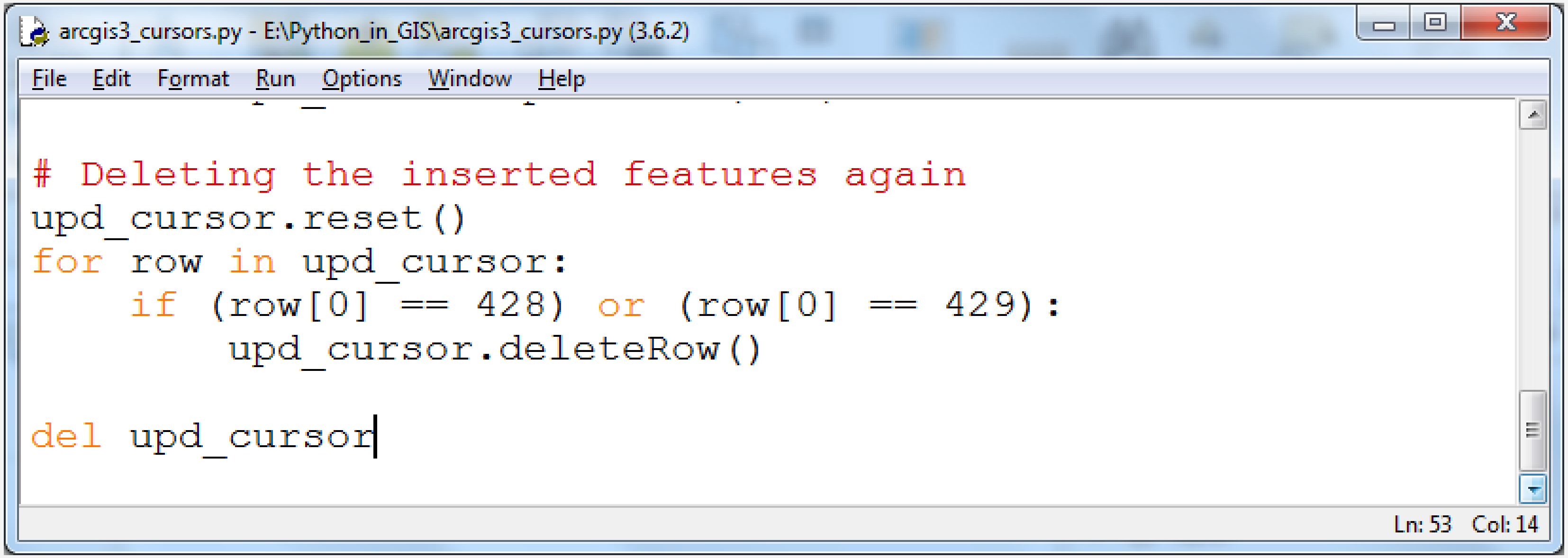


FID	Shape	track_fid	track_seg_	track_se_1	ele	time	time_str
426	Point	0	0	426	83	01.01.2010	03:10:14
427	Point	0	0	427	83	01.01.2010	03:10:16
428	Point	0	0	428	80,1	<Null>	
429	Point	0	0	429	78,3	<Null>	

Click to add new row.

0 of 430 selected

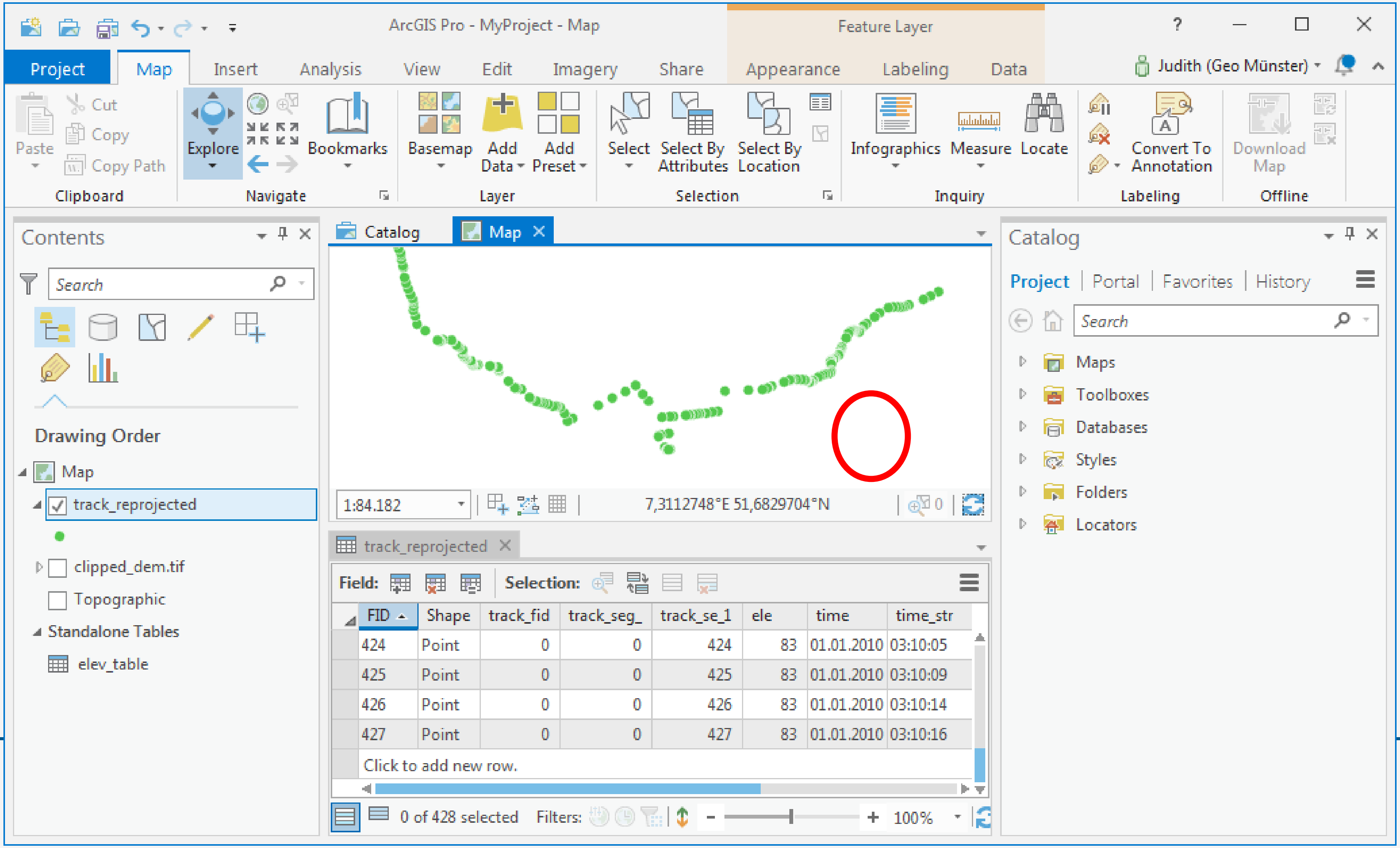
Update cursors (4)



arcgis3_cursors.py - E:\Python_in_GIS\arcgis3_cursors.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 53 Col: 14

```
# Deleting the inserted features again
upd_cursor.reset()
for row in upd_cursor:
    if (row[0] == 428) or (row[0] == 429):
        upd_cursor.deleteRow()

del upd_cursor
```

Update cursors (5)



FID	Shape	track_fid	track_seg_	track_se_1	ele	time	time_str
424	Point	0	0	424	83	01.01.2010	03:10:05
425	Point	0	0	425	83	01.01.2010	03:10:09
426	Point	0	0	426	83	01.01.2010	03:10:14
427	Point	0	0	427	83	01.01.2010	03:10:16

Click to add new row.

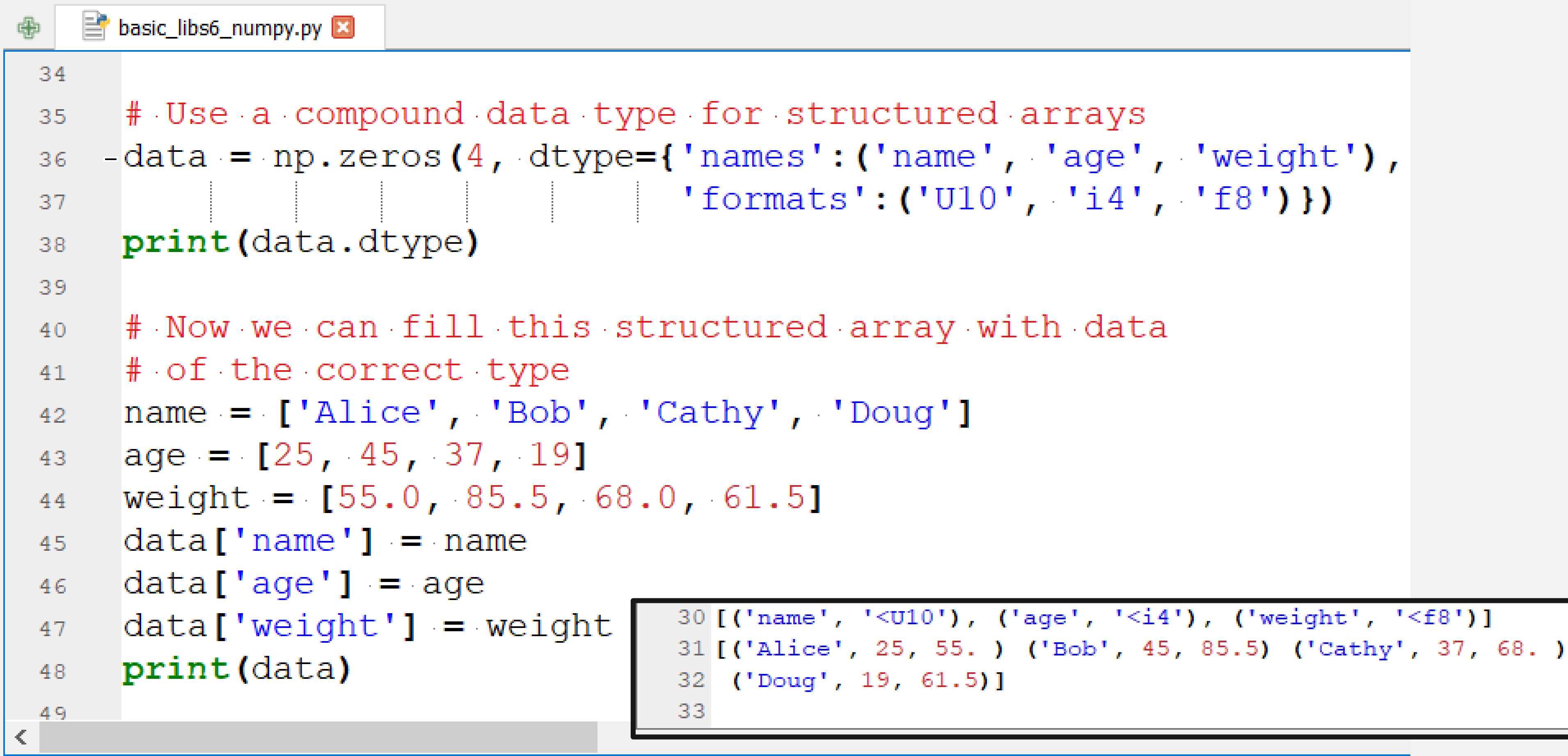
0 of 428 selected

Combining arcpy and numpy

Remember: Structured arrays

Structured arrays can have a separate data type per column, i.e. attribute

And you can refer to attributes either by index or by name



```python

# Use a compound data type for structured arrays
data = np.zeros(4, dtype={'names':('name', 'age', 'weight'),
                          'formats':('U10', 'i4', 'f8')})
print(data.dtype)

# Now we can fill this structured array with data
# of the correct type
name = ['Alice', 'Bob', 'Cathy', 'Doug']
age = [25, 45, 37, 19]
weight = [55.0, 85.5, 68.0, 61.5]
data['name'] = name
data['age'] = age
data['weight'] = weight
print(data)

```

```
[('name', '<U10'), ('age', '<i4'), ('weight', '<f8')]
[('Alice', 25, 55. ) ('Bob', 45, 85.5) ('Cathy', 37, 68. )
 ('Doug', 19, 61.5)]

```

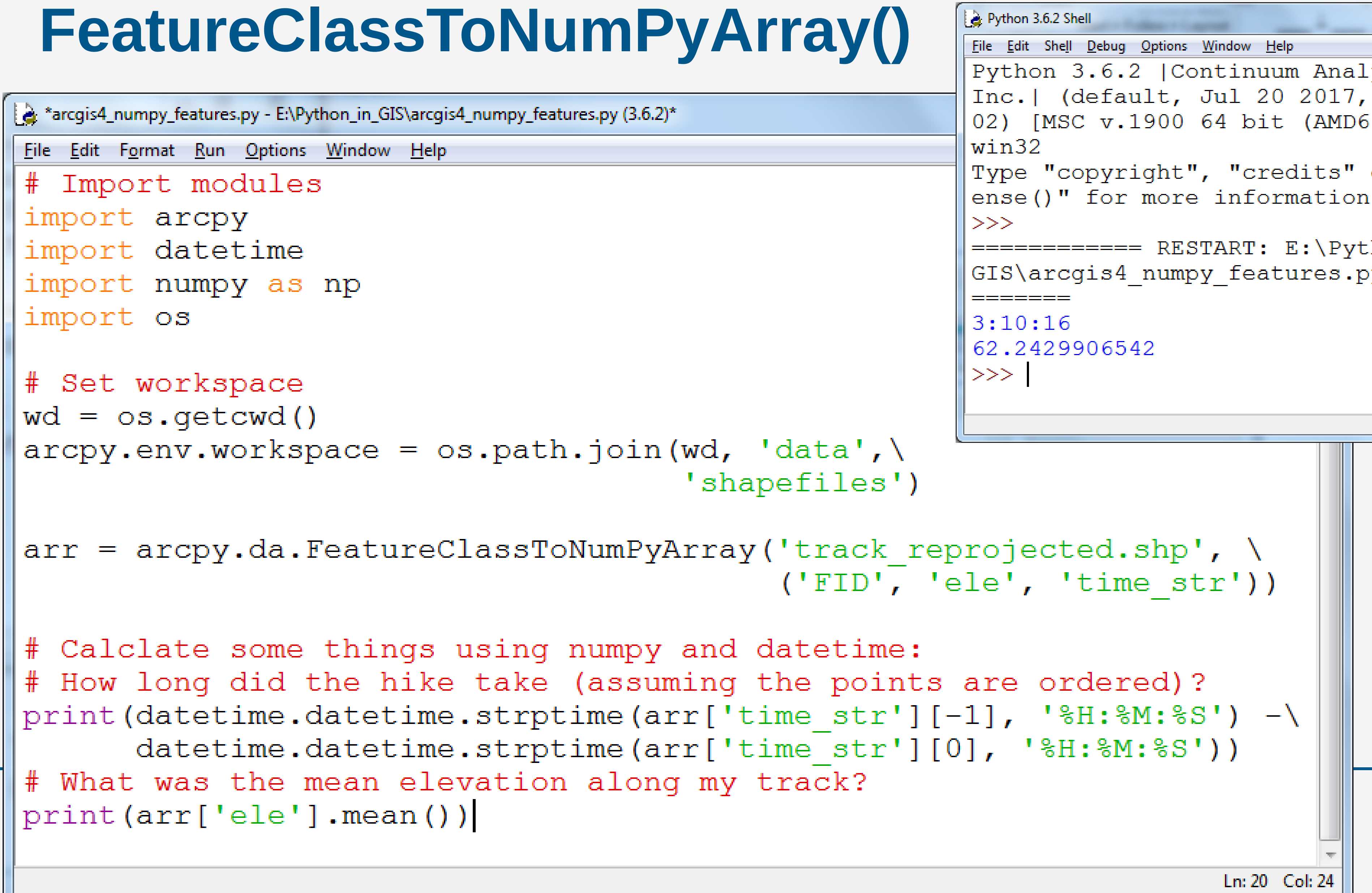
Using numpy for feature data in ArcGIS

Table and feature classes can be converted to and from numpy arrays using functions in the data access (arcpy.da) module.

Functions	Explanation
ExtendTable()	Join the contents of a numpy array to a table based on a common field.
FeatureClassToNumPyArray()	Convert a feature class to a numpy array.
NumPyArrayToFeatureClass()	Convert a numpy array to a feature class.
NumPyArrayToTable()	Convert a numpy array to a table.
TableToNumPyArray()	Convert a table to numpy array.

To convert numpy arrays to tables and feature classes, the arrays must be structured arrays.

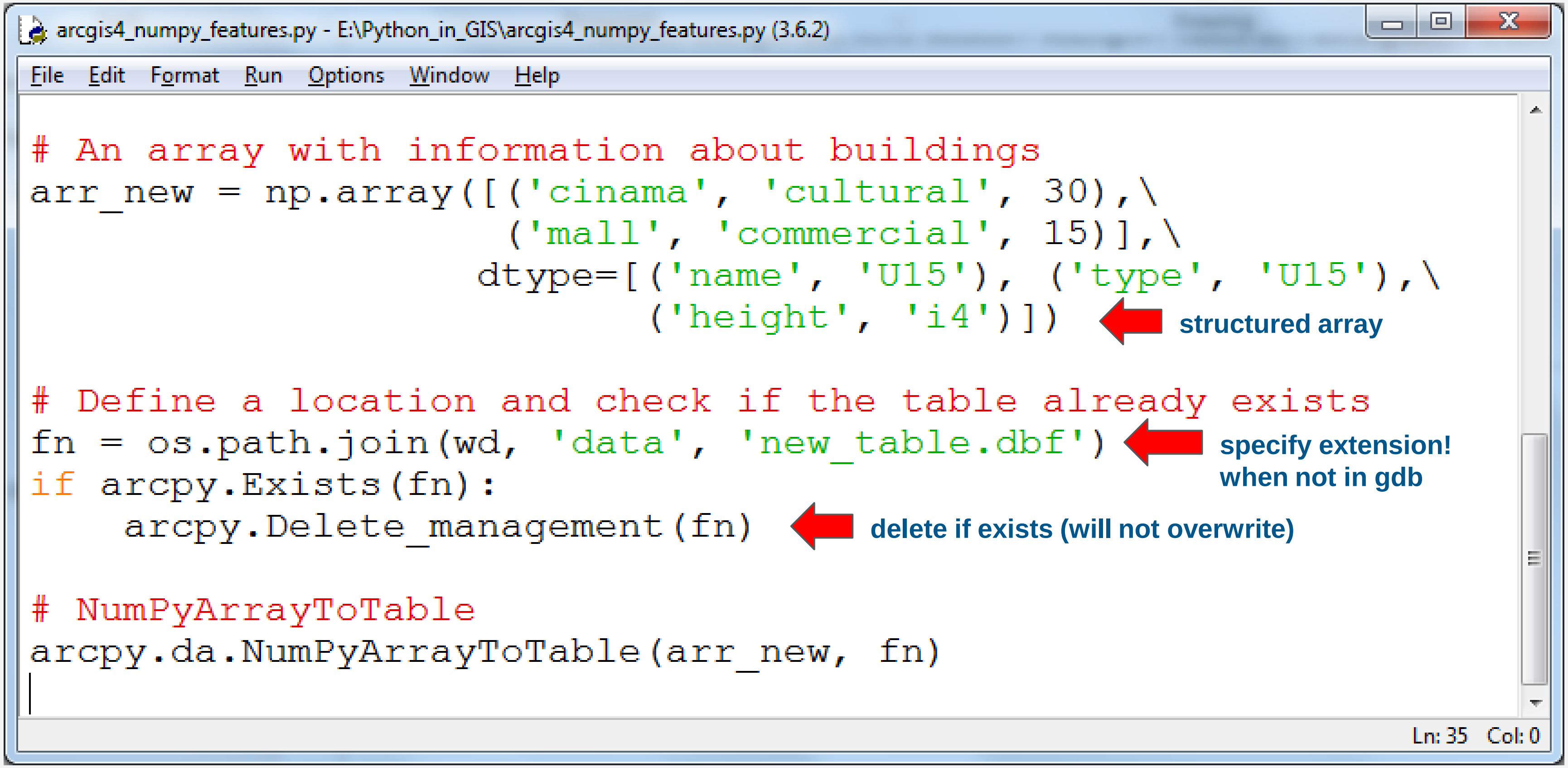
FeatureClassToNumPyArray()


arcgis4_numpy_features.py - E:\Python_in_GIS\arcgis4_numpy_features.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 20 Col: 24
Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

```python
# Import modules
import arcpy
import datetime
import numpy as np
import os

# Set workspace
wd = os.getcwd()
arcpy.env.workspace = os.path.join(wd, 'data',\
                                   'shapefiles')

arr = arcpy.da.FeatureClassToNumPyArray('track_reprojected.shp', \
                                        ('FID', 'ele', 'time_str'))

# Calclate some things using numpy and datetime:
# How long did the hike take (assuming the points are ordered)?
print(datetime.datetime.strptime(arr['time_str'][-1], '%H:%M:%S') -\
      datetime.datetime.strptime(arr['time_str'][0], '%H:%M:%S'))
# What was the mean elevation along my track?
print(arr['ele'].mean())
```

```
Python 3.6.2 |Continuum Anal
Inc.| (default, Jul 20 2017,
02) [MSC v.1900 64 bit (AMD6
win32
Type "copyright", "credits" 
ense()" for more information
>>> 
============ RESTART: E:\Pyt
GIS\arcgis4_numpy_features.p
=======
3:10:16
62.2429906542
>>> 
```

NumPyArrayToTable()



arcgis4_numpy_features.py - E:\Python_in_GIS\arcgis4_numpy_features.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 35 Col: 0

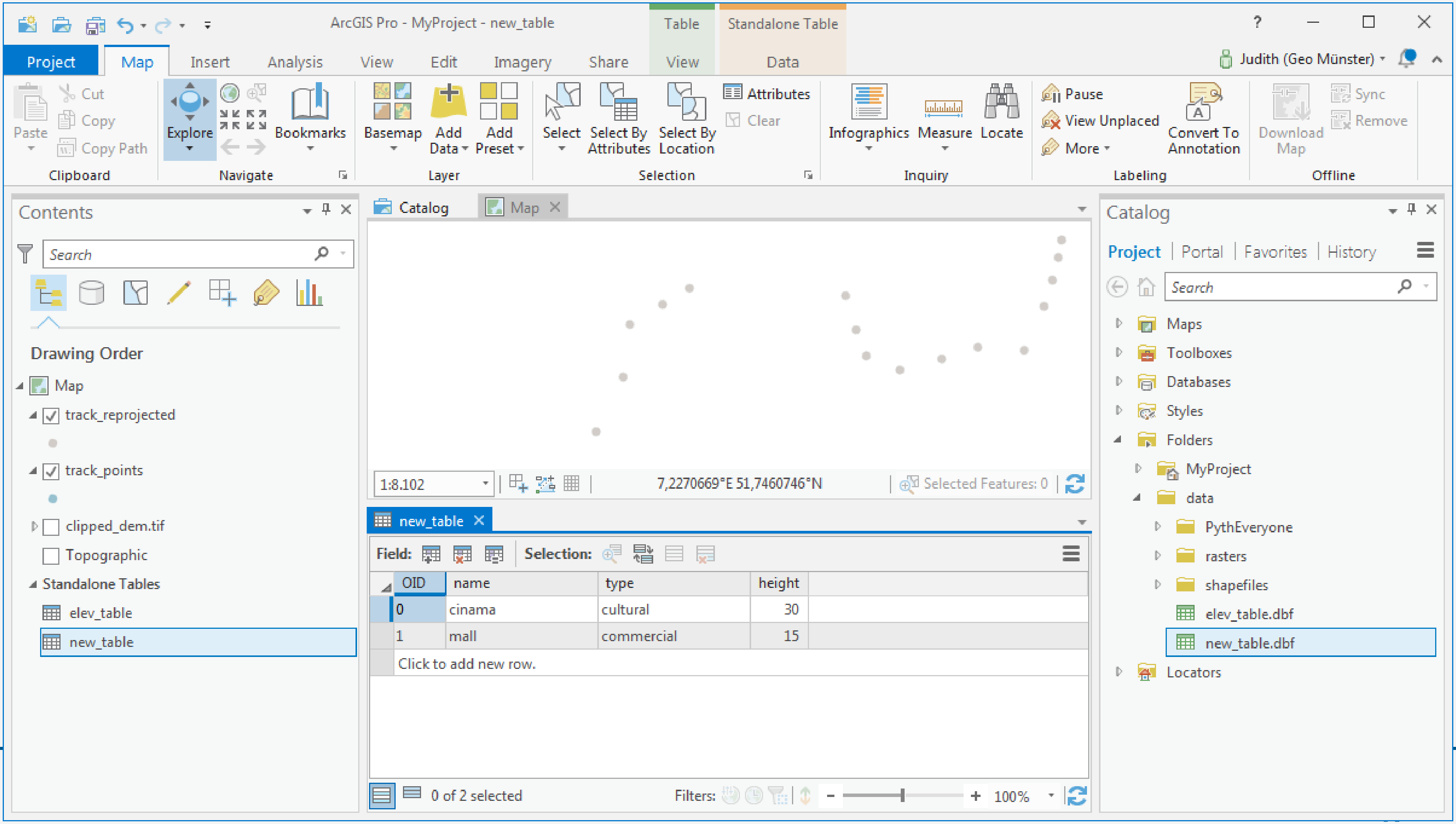
```
# An array with information about buildings
arr_new = np.array([('cinama', 'cultural', 30),\
                    ('mall', 'commercial', 15)],\
                  dtype=[('name', 'U15'), ('type', 'U15'),\
                         ('height', 'i4')])

# Define a location and check if the table already exists
fn = os.path.join(wd, 'data', 'new_table.dbf')
if arcpy.Exists(fn):
    arcpy.Delete_management(fn)

# NumPyArrayToTable
arcpy.da.NumPyArrayToTable(arr_new, fn)
```

structured array

specify extension!
when not in gdb

delete if exists (will not overwrite)

Result



OID	name	type	height
0	cinama	cultural	30
1	mall	commercial	15

Click to add new row.

0 of 2 selected

Using numpy for raster data

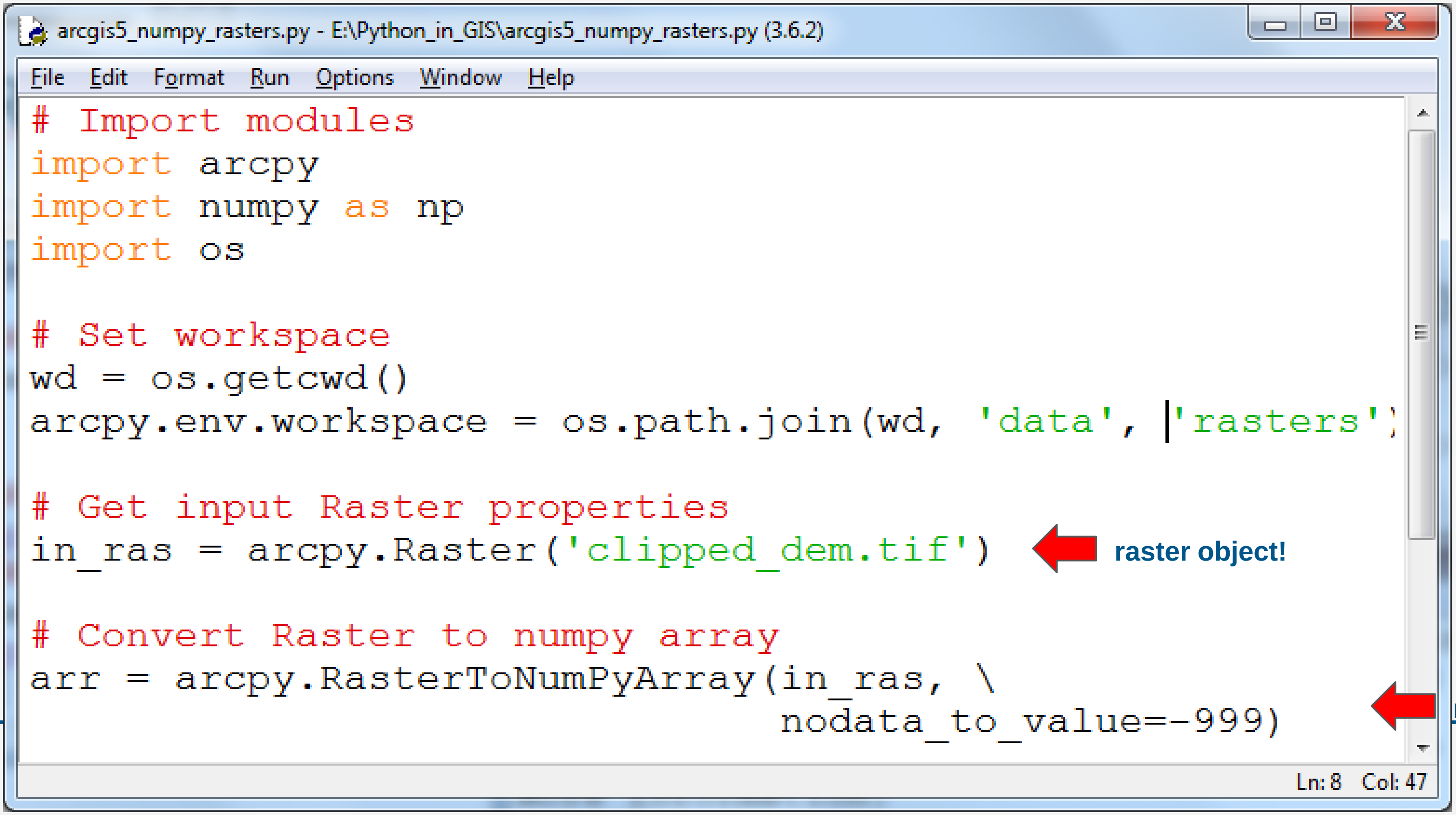
Rasters can be converted to and from numpy arrays. You may want to convert an ArcGIS raster to a NumPy array to:

- Implement an existing numpy function on the array (e.g. run filters).
- Develop a custom function by accessing the individual cells within the NumPy array (e.g. implement a neighborhood notation).

Functions	Explanation
RasterToNumPyArray()	Convert a raster to a NumPy array.
NumPyArrayToRaster()	Convert a NumPy array to a raster.

RasterToNumPyArray() (1)

Syntax: `RasterToNumPyArray (in_raster, {lower_left_corner}, {ncols}, {nrows}, {nodata_to_value})`

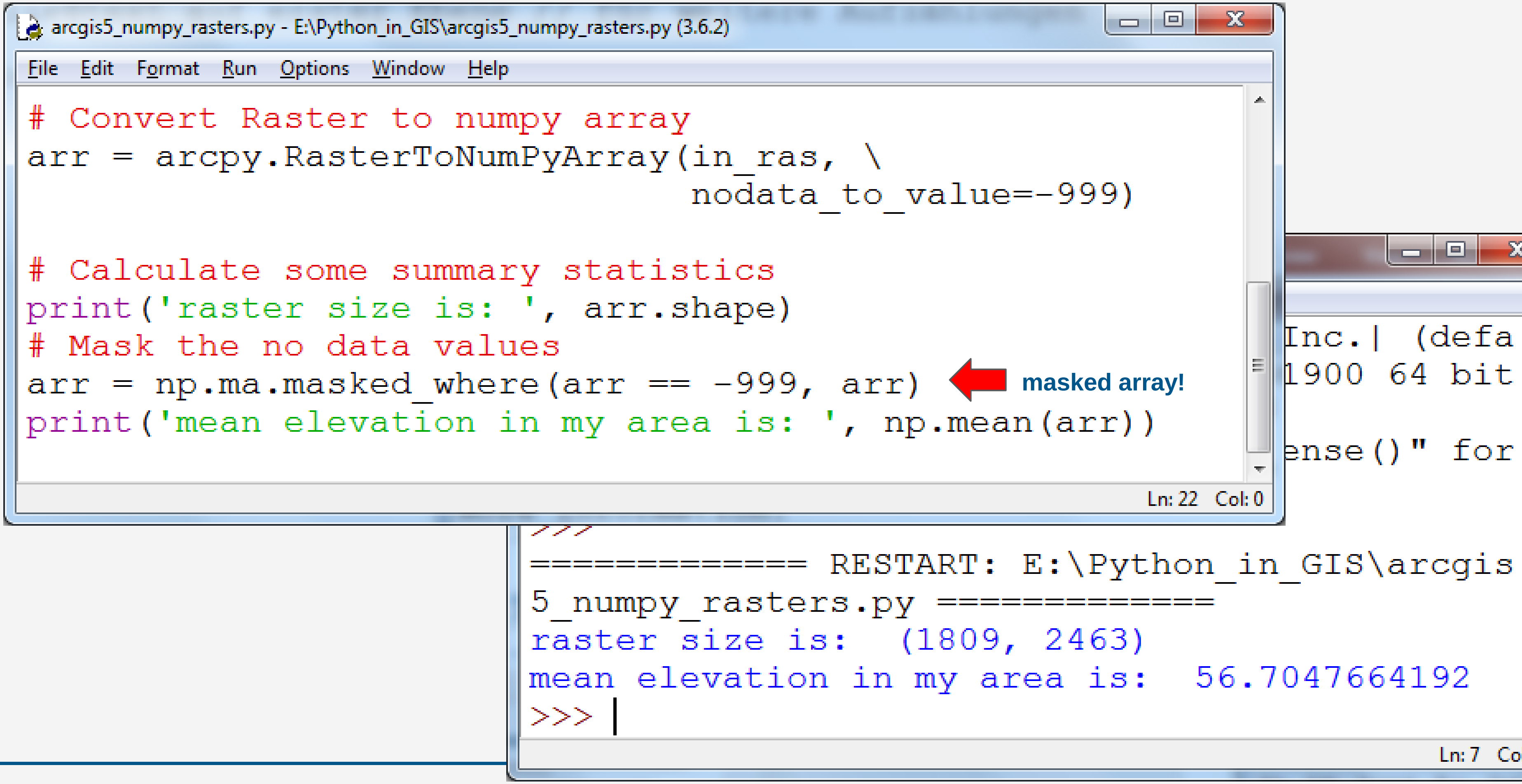

arcgis5_numpy_rasters.py - E:\Python_in_GIS\arcgis5_numpy_rasters.py (3.6.2)
File
Edit
Format
Run
Options
Window
Help
Ln: 8
Col: 47

```
# Import modules
import arcpy
import numpy as np
import os

# Set workspace
wd = os.getcwd()
arcpy.env.workspace = os.path.join(wd, 'data', 'rasters')

# Get input Raster properties
in_ras = arcpy.Raster('clipped_dem.tif')

# Convert Raster to numpy array
arr = arcpy.RasterToNumPyArray(in_ras, \
                               nodata_to_value=-999)
```

raster object!

missing values

RasterToNumPyArray() (2)


arcgis5_numpy_rasters.py - E:\Python_in_GIS\arcgis5_numpy_rasters.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 22 Col: 0
Ln: 7 Co

```
# Convert Raster to numpy array
arr = arcpy.RasterToNumPyArray(in_ras, \
                               nodata_to_value=-999)

# Calculate some summary statistics
print('raster size is: ', arr.shape)
# Mask the no data values
arr = np.ma.masked_where(arr == -999, arr)
print('mean elevation in my area is: ', np.mean(arr))
```

masked array!

```
Inc.| (defa
1900 64 bit
ense()" for
>>>
============== RESTART: E:\Python_in_GIS\arcgis
5_numpy_rasters.py ==============
raster size is:  (1809, 2463)
mean elevation in my area is:  56.7047664192
>>> 
```

Exercise #2

Make an arcpy tool script to calculate the mean difference between the elevation from the GPS track and the one you have obtained from the DEM (now a field in the attribute table) and the standard deviation of the differences.

FID	...	ele	...	ele_raster
0		86		90
1		81		85
2		79		80
3		79		80
4		78		80
5		80		78

Average difference over all features

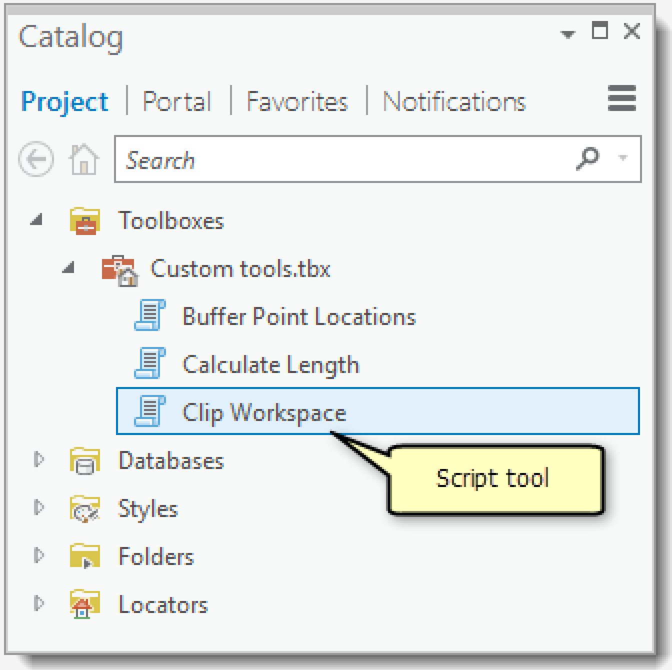
Script tools

What is a script tool?

It is a Python-based tool inside a toolbox.

A script tool is like any other tool; it can:

- be opened and executed from the tool dialog box,
- be used in models and the Python window,
- write messages to the Geoprocessing history,
- and be called from scripts.



Source: https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/a-quick-tour-of-creating-tools-in-python.htm

How to create a script tool?

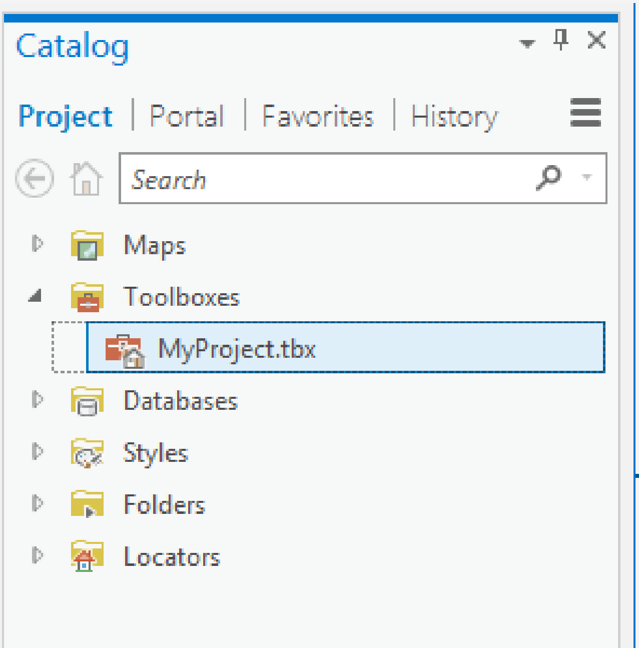
You need at least four things:

1. A custom toolbox
2. Create a script tool in this toolbox
3. Create / adapt a script to become the source code of this script tool
4. A definition of the parameters in your script
5. (Optional: Validation)

Step 1: Create a custom toolbox

A custom toolbox can be created by:

1. right-clicking Toolboxes in the Catalog pane,
2. New Toolbox,
3. In the Select Toolbox dialog box, browse to the folder or geodatabase in which you want to create the new toolbox, then enter a new name,
4. and Save.



Step 2: Create a script tool (1)

To create a script tool in the new toolbox:

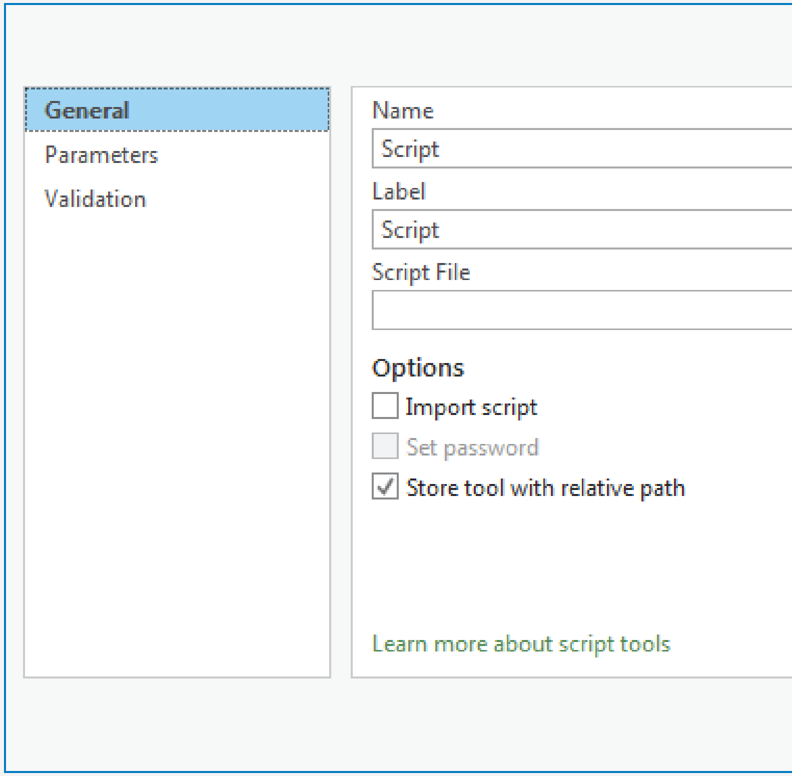
- right-click your toolbox,
- New > Script.
- This opens a dialog taking you through the process of creating a script tool.

You can make changes to a script tool you’ve created before by right-clicking the tool and choosing Properties.

Step 2: Create a script tool (2)

Within this process of creating the script tool, you need to fill in the following:

- Name: used when the tool is run from Python.
  - So, no space in this name!
- Label: display name in the Geoprocessing pane
- Script file: Location of the script (see next slides)
- Optionally: you can import (embed) the script
  - This stores your script with the toolbox
- Optionally: If embedded, you can set a password
  - This restrict access to the source code



Name
Script

Label
Script

Script File

Options

□ Import script

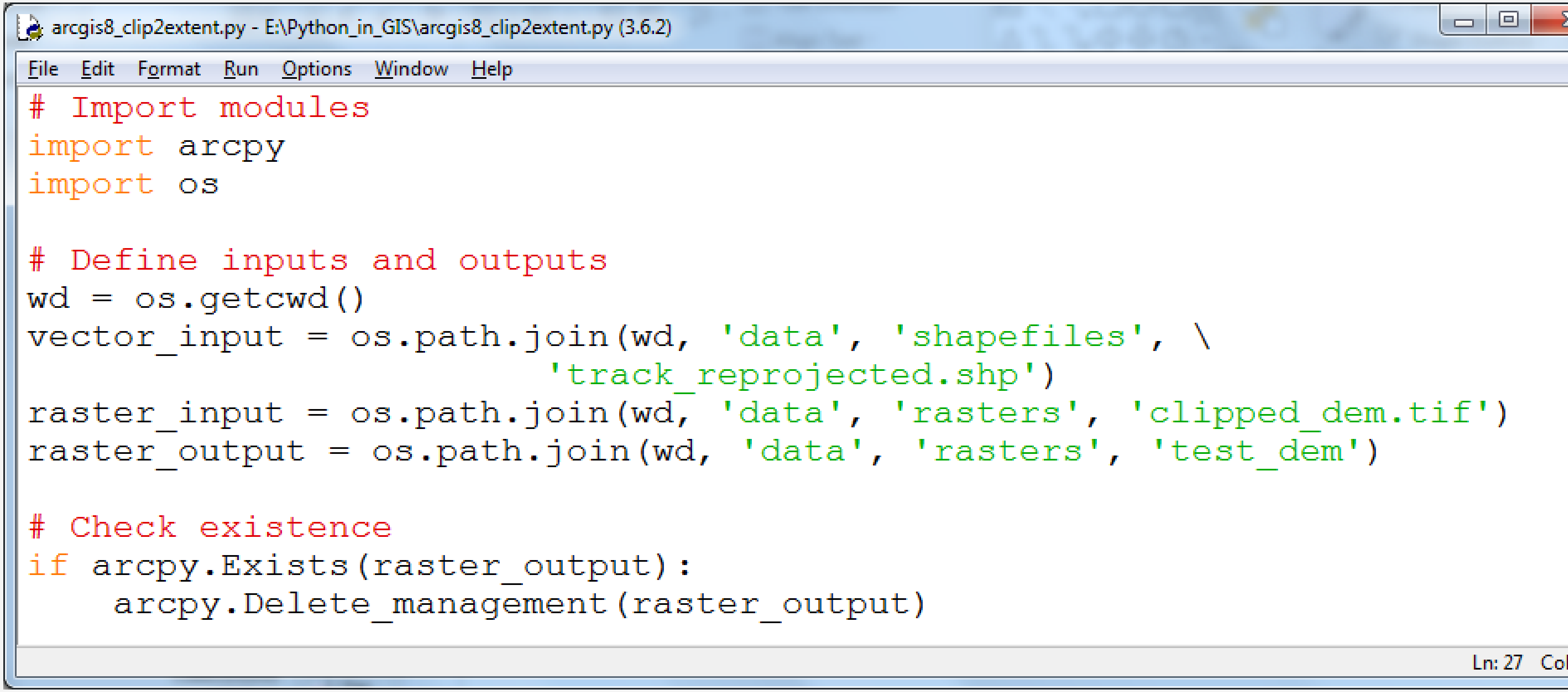
□ Set password

☑ Store tool with relative path

Step 3: Create / adapt a script to become the source code of this script tool

See example on the following slides.

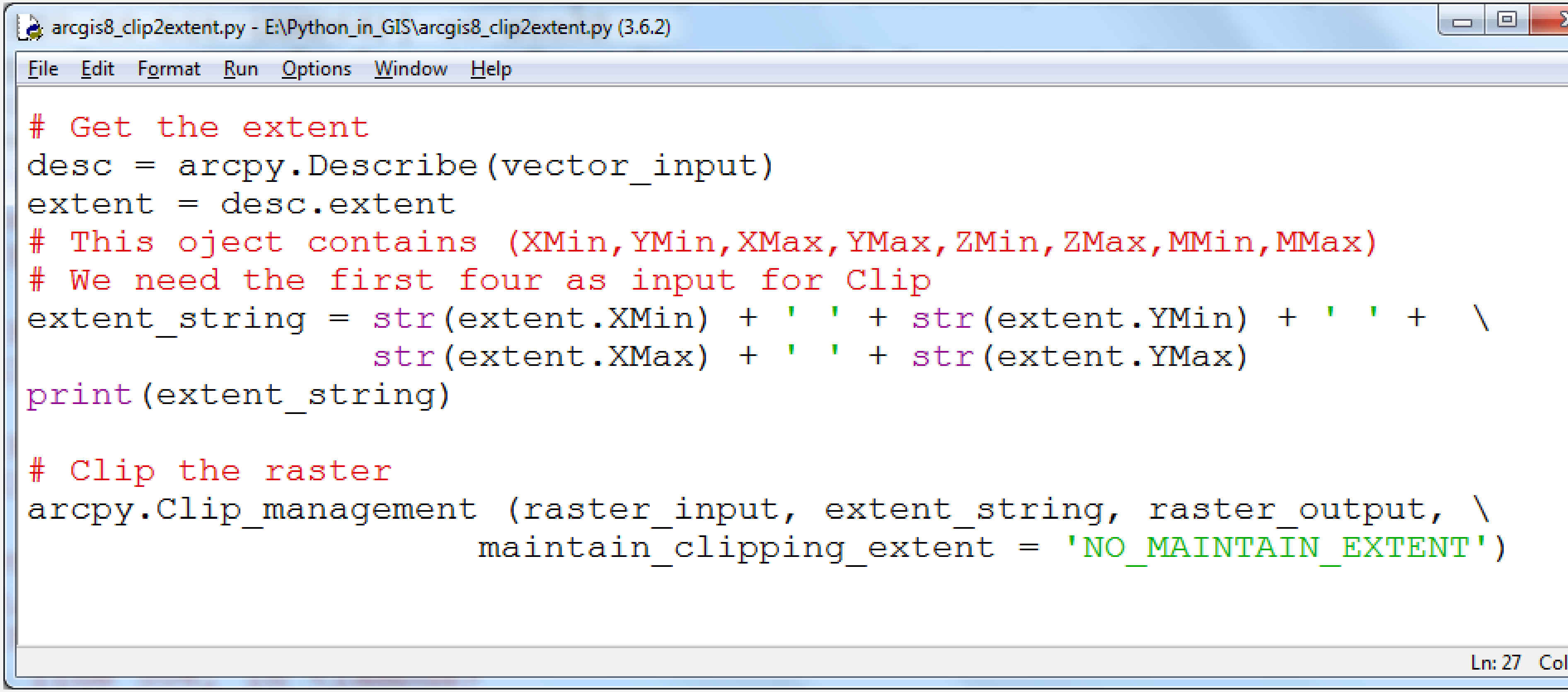
Consider making a tool of this script (1)



arcgis8_clip2extent.py - E:\Python_in_GIS\arcgis8_clip2extent.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 27 Col

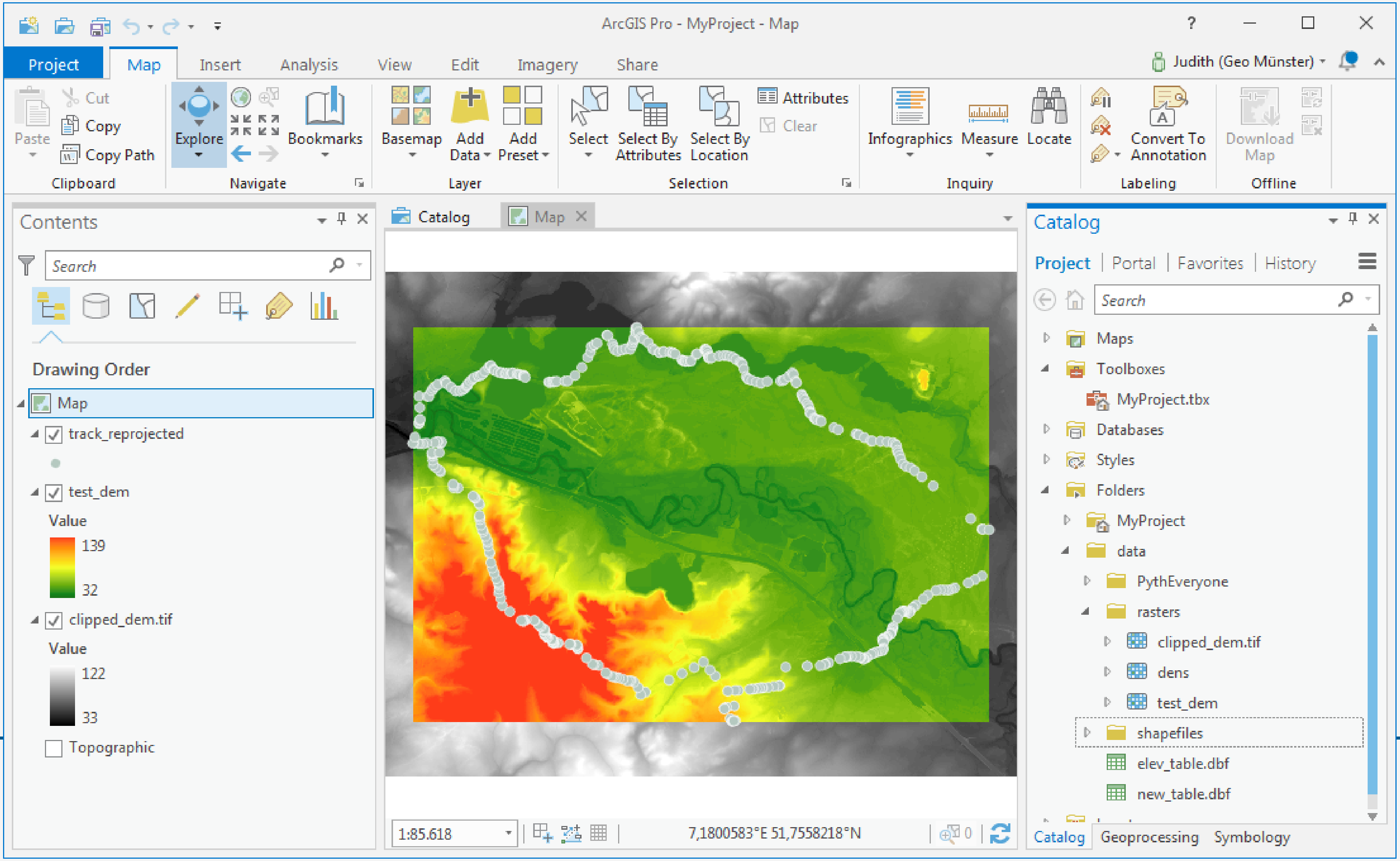
```
# Import modules
import arcpy
import os

# Define inputs and outputs
wd = os.getcwd()
vector_input = os.path.join(wd, 'data', 'shapefiles', \
                        'track_reprojected.shp')
raster_input = os.path.join(wd, 'data', 'rasters', 'clipped_dem.tif')
raster_output = os.path.join(wd, 'data', 'rasters', 'test_dem')

# Check existence
if arcpy.Exists(raster_output):
    arcpy.Delete_management(raster_output)
```

Consider making a tool of this script (2)


arcgis8_clip2extent.py - E:\Python_in_GIS\arcgis8_clip2extent.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 27 Col

```python
# Get the extent
desc = arcpy.Describe(vector_input)
extent = desc.extent
# This oject contains (XMin,YMin,XMax,YMax,ZMin,ZMax,MMin,MMax)
# We need the first four as input for Clip
extent_string = str(extent.XMin) + ' ' + str(extent.YMin) + ' ' +  \
                str(extent.XMax) + ' ' + str(extent.YMax)
print(extent_string)

# Clip the raster
arcpy.Clip_management (raster_input, extent_string, raster_output, \
                     maintain_clipping_extent = 'NO_MAINTAIN_EXTENT')
```

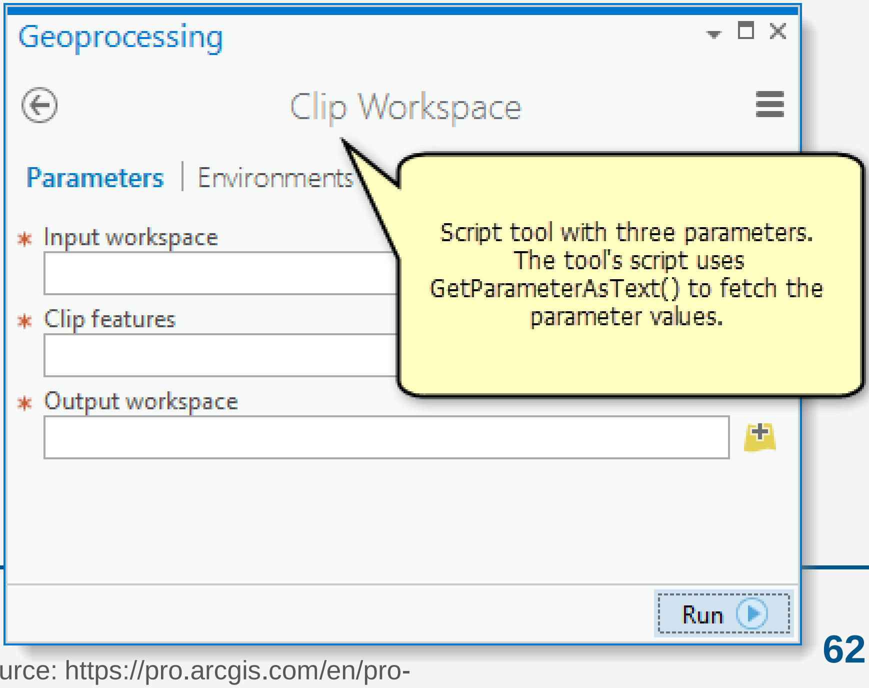
Result


ArcGIS Pro - MyProject - Map
Project
Map
Insert
Analysis
View
Edit
Imagery
Share
Judith (Geo Münster)
Paste
Cut
Copy
Copy Path
Clipboard
Explore
Bookmarks
Navigate
Basemap
Add Data
Add Preset
Layer
Select
Select By Attributes
Select By Location
Attributes
Clear
Selection
Infographics
Measure
Locate
Inquiry
Convert To Annotation
Labeling
Download Map
Offline
Contents
Search
Drawing Order
Map
track_reprojected
test_dem
Value
139
32
clipped_dem.tif
Value
122
33
Topographic
Catalog
Map
1:85.618
7,1800583°E 51,7558218°N
0
Catalog
Project
Portal
Favorites
History
Search
Maps
Toolboxes
MyProject.tbx
Databases
Styles
Folders
MyProject
data
PythEveryone
rasters
clipped_dem.tif
dens
test_dem
shapefiles
elev_table.dbf
new_table.dbf
Catalog
Geoprocessing
Symbology

Think about input and output parameters

- Almost all tools have parameters.
- You set their values on the tool dialog box or within a script.
- When the tool is executed, the parameter values are sent to your tool's source code.
- Your script reads these values and proceeds with its work.

What are the parameters in our script?



Script tool with three parameters. The tool's script uses GetParameterAsText() to fetch the parameter values.

source: https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/understanding-script-tool-parameters.htm

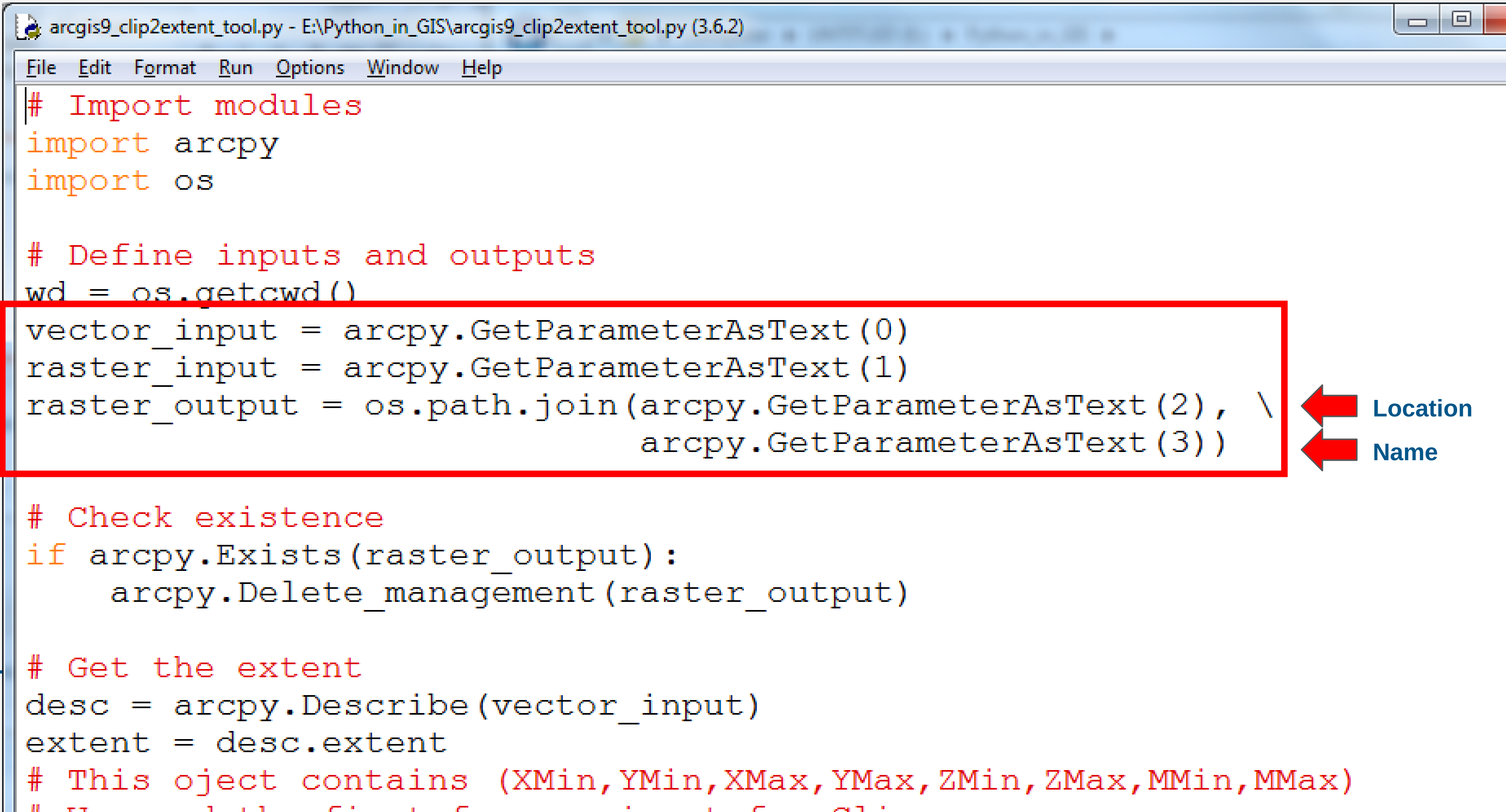
Adapt script to accomodate parameters (1)

To access the parameters in the tool interface, use one of:

- GetParameter(): Gets script tool parameter (by index) as object.
- GetParameterAsText(): Gets the specified parameter as a string.

Adapt script to accomodate parameters (2)

So our example script becomes:



arcgis9_clip2extent_tool.py - E:\Python_in_GIS\arcgis9_clip2extent_tool.py (3.6.2)
File Edit Format Run Options Window Help

```
# Import modules
import arcpy
import os

# Define inputs and outputs
wd = os.getcwd()
vector_input = arcpy.GetParameterAsText(0)
raster_input = arcpy.GetParameterAsText(1)
raster_output = os.path.join(arcpy.GetParameterAsText(2), \
                             arcpy.GetParameterAsText(3))

# Check existence
if arcpy.Exists(raster_output):
    arcpy.Delete_management(raster_output)

# Get the extent
desc = arcpy.Describe(vector_input)
extent = desc.extent
# This oject contains (XMin,YMin,XMax,YMax,ZMin,ZMax,MMin,MMax)
```

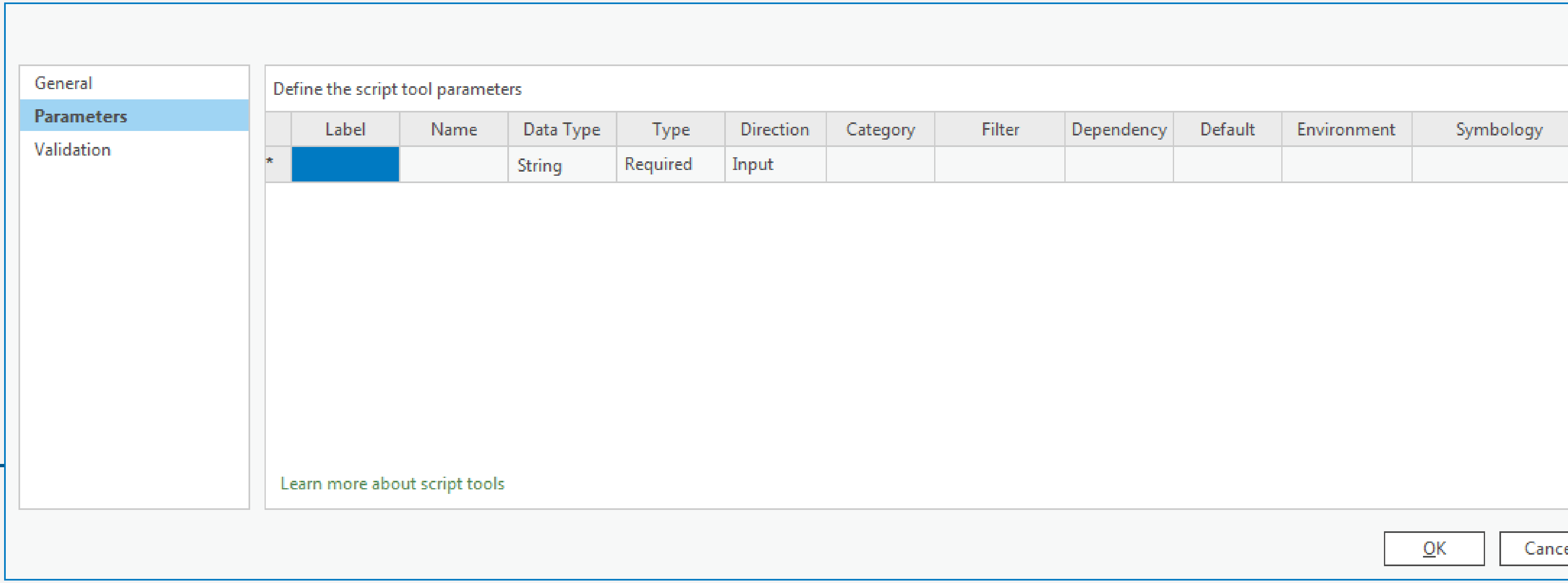
Location

Name

Step 4: Define the parameters in the tool (1)

Per parameter, you need to fill in at least the first five columns:

- Label and Name, as before,
- Data type, can be tricky what to choose
- Required/opional and Input/Output (for use in other scripts or ModelBuilder)

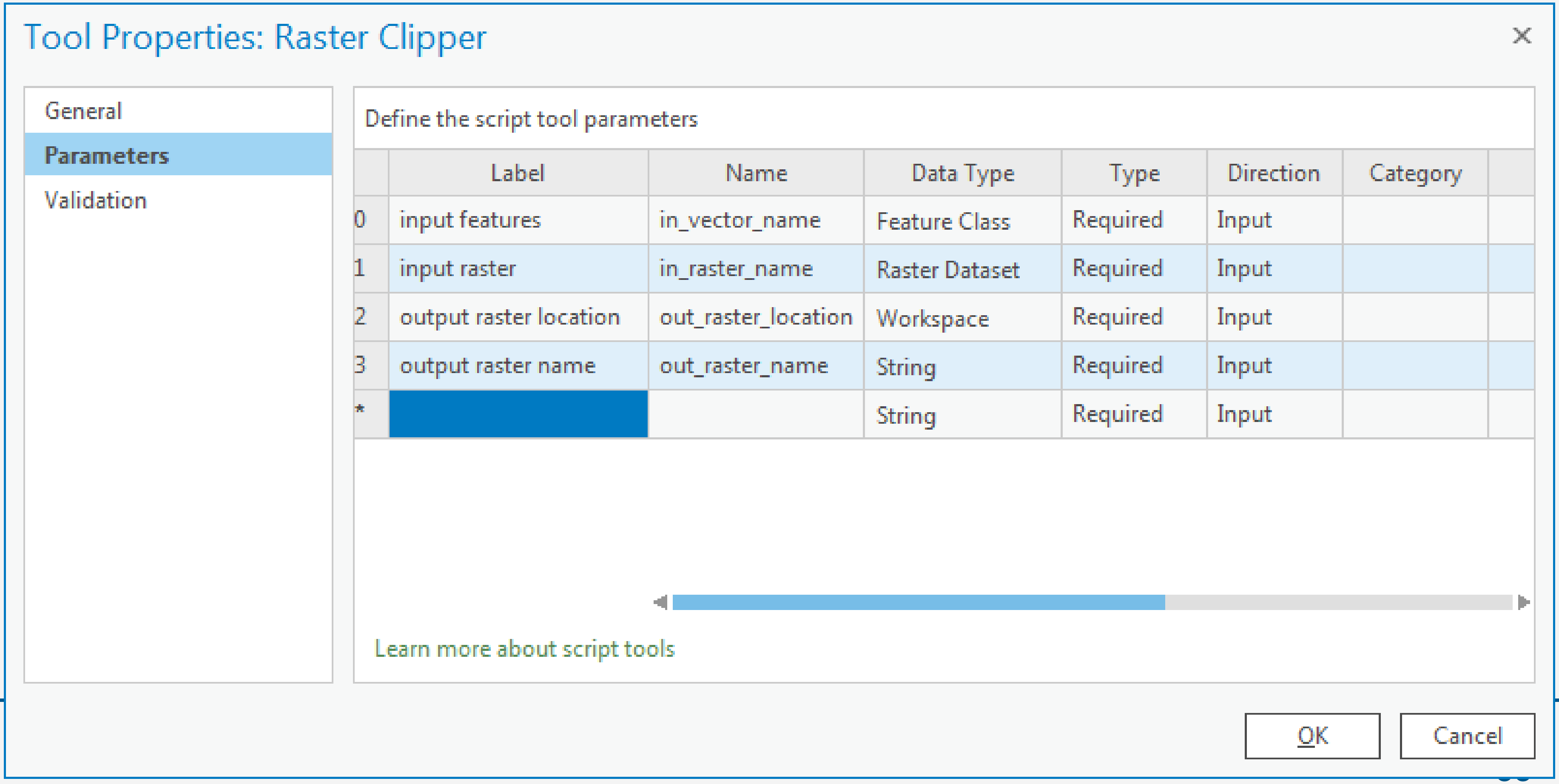


Define the script tool parameters

	Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default	Environment	Symbology
*			String	Required	Input						

Step 4: Define the parameters in the tool (2)

So for our example:



Tool Properties: Raster Clipper

Define the script tool parameters

	Label	Name	Data Type	Type	Direction	Category	
0	input features	in_vector_name	Feature Class	Required	Input		
1	input raster	in_raster_name	Raster Dataset	Required	Input		
2	output raster location	out_raster_location	Workspace	Required	Input		
3	output raster name	out_raster_name	String	Required	Input		
*			String	Required	Input		

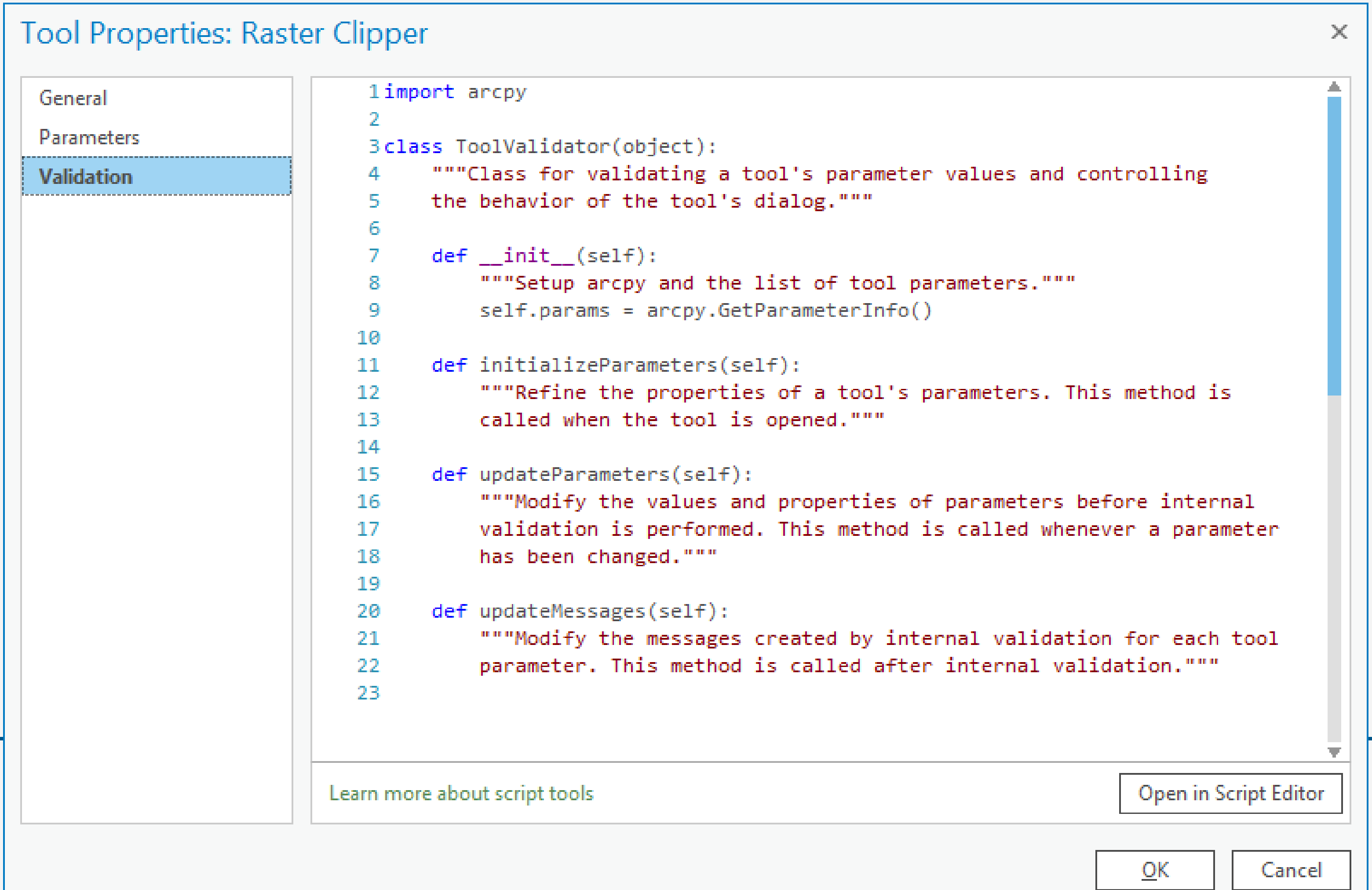
Learn more about script tools

Step 5: (Optional: Validation) (1)

- Validation is everything that happens before a tool's OK button is clicked.
- It allows you to customize how parameters respond and interact to values and each other.
- Validation is performed with a block of Python code.

Step 5: (Optional: Validation) (2)

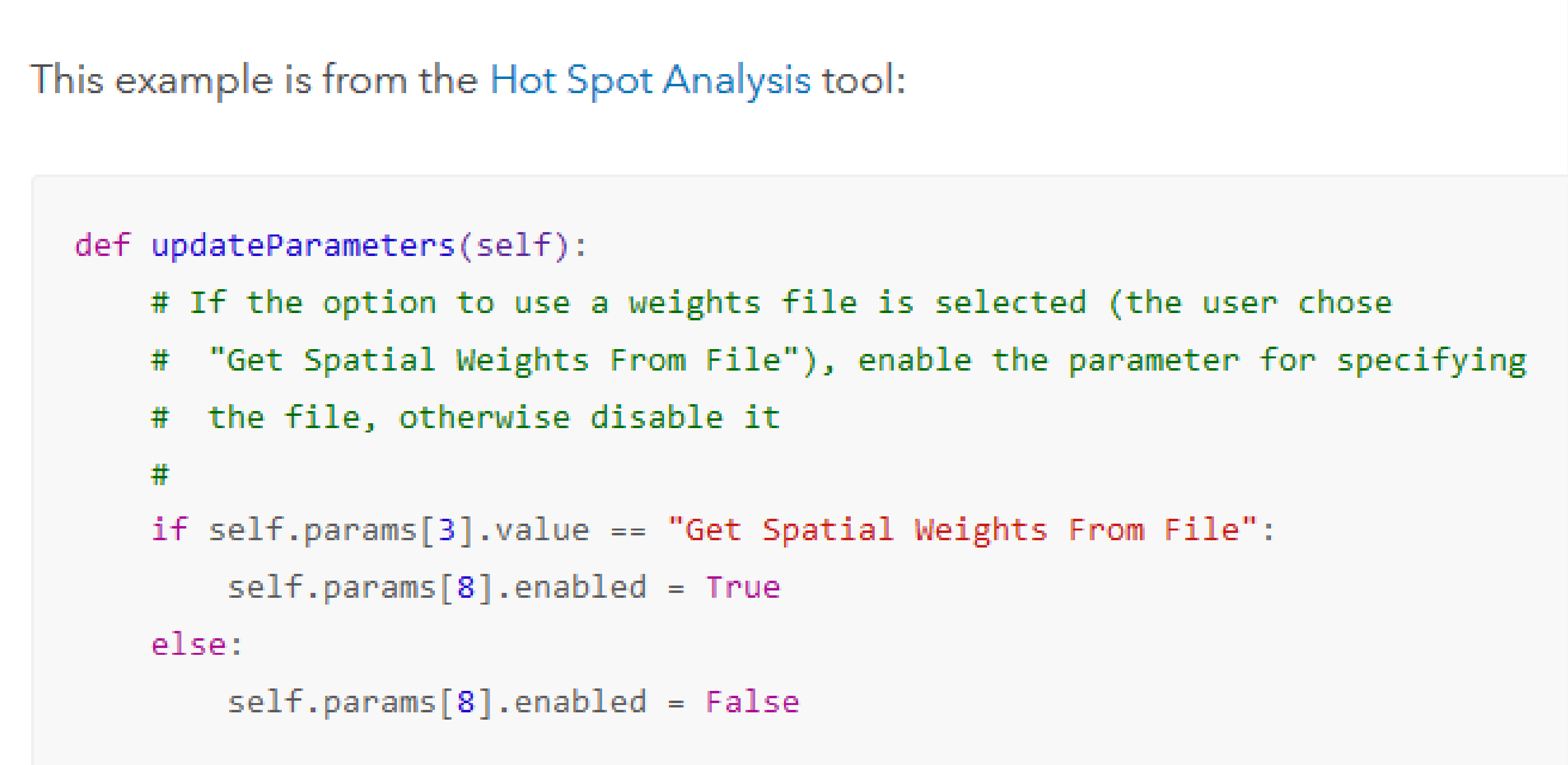
The default looks like this


Tool Properties: Raster Clipper
General
Parameters
Validation
Learn more about script tools
Open in Script Editor
OK
Cancel

```python
import arcpy

class ToolValidator(object):
    """Class for validating a tool's parameter values and controlling
    the behavior of the tool's dialog."""

    def __init__(self):
        """Setup arcpy and the list of tool parameters."""
        self.params = arcpy.GetParameterInfo()

    def initializeParameters(self):
        """Refine the properties of a tool's parameters. This method is
        called when the tool is opened."""

    def updateParameters(self):
        """Modify the values and properties of parameters before internal
        validation is performed. This method is called whenever a parameter
        has been changed."""

    def updateMessages(self):
        """Modify the messages created by internal validation for each tool
        parameter. This method is called after internal validation."""

```

Step 5: (Optional: Validation) (3)

Example from the arcpy documentation:

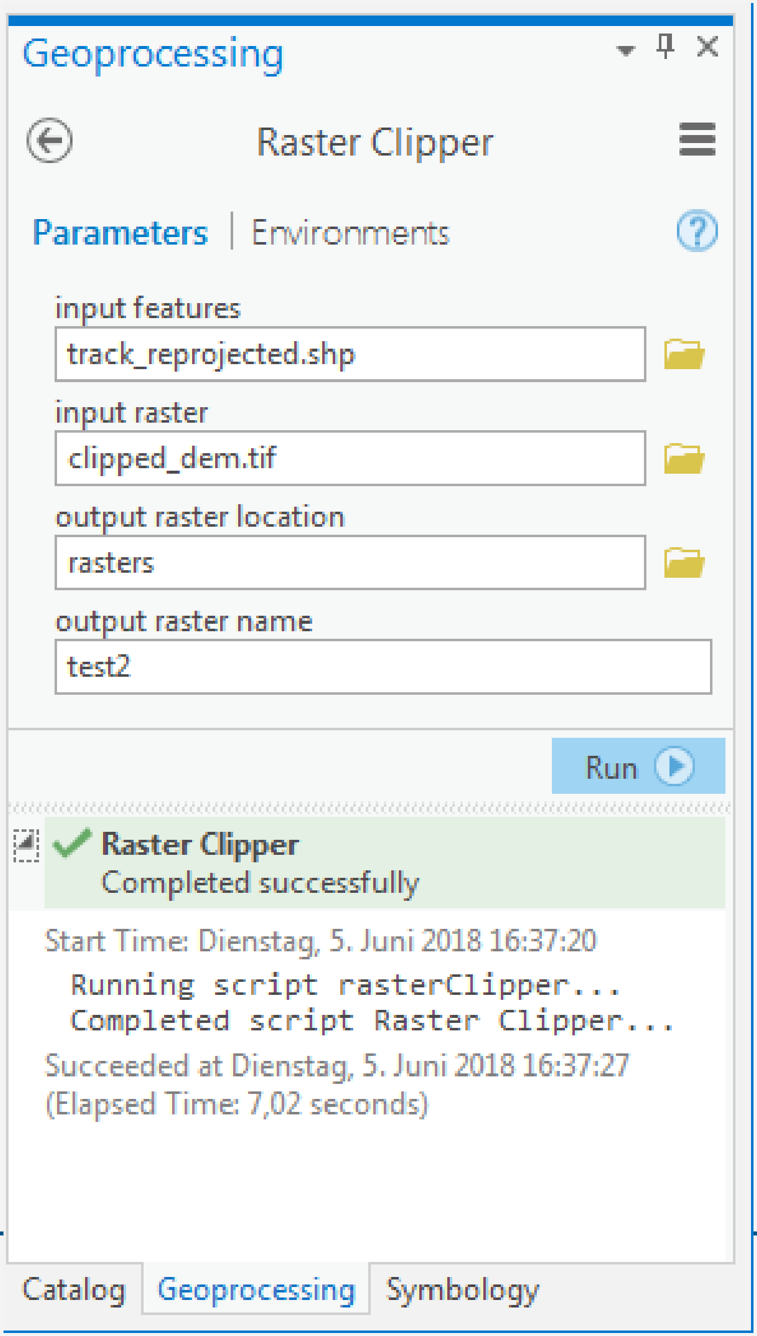


This example is from the Hot Spot Analysis tool:

```
def updateParameters(self):
    # If the option to use a weights file is selected (the user chose
    #  "Get Spatial Weights From File"), enable the parameter for specifying
    #  the file, otherwise disable it
    #
    if self.params[3].value == "Get Spatial Weights From File":
        self.params[8].enabled = True
    else:
        self.params[8].enabled = False
```

Source: https://pro.arcgis.com/en/pro-app/arcpy/geoprocessing_and_python/customizing-script-tool-behavior.htm

Now run the tool in ArcGIS Pro



input features: track_reprojected.shp

input raster: clipped_dem.tif

output raster location: rasters

output raster name: test2

Raster Clipper
Completed successfully

Start Time: Dienstag, 5. Juni 2018 16:37:20

```
  Running script rasterClipper...
  Completed script Raster Clipper...
```

Succeeded at Dienstag, 5. Juni 2018 16:37:27
(Elapsed Time: 7,02 seconds)

Exercise #3 (extra)

Make a script tool that adds a field to your track‘s attribute table with the speed between a point and the previous point.

You may calculate this in a 2d plane (not taking into account the elevation).

The script tool should have as inputs:

- The gps track (feature class)
- The name of the field with the times, that should be used to compute the speeds

Hint: The time field has data type string; use the datetime library to convert it to a format you can calculate with