

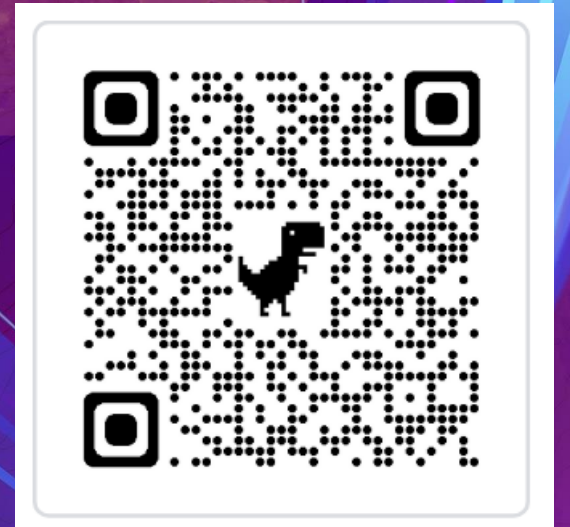
Building Geoprocessing Tools in ArcGIS

Dave Wynne

Sean Lim

<https://github.com/dWynne1/uc2023-tools>

2023 ESRI USER CONFERENCE



Building Geoprocessing Tools in ArcGIS



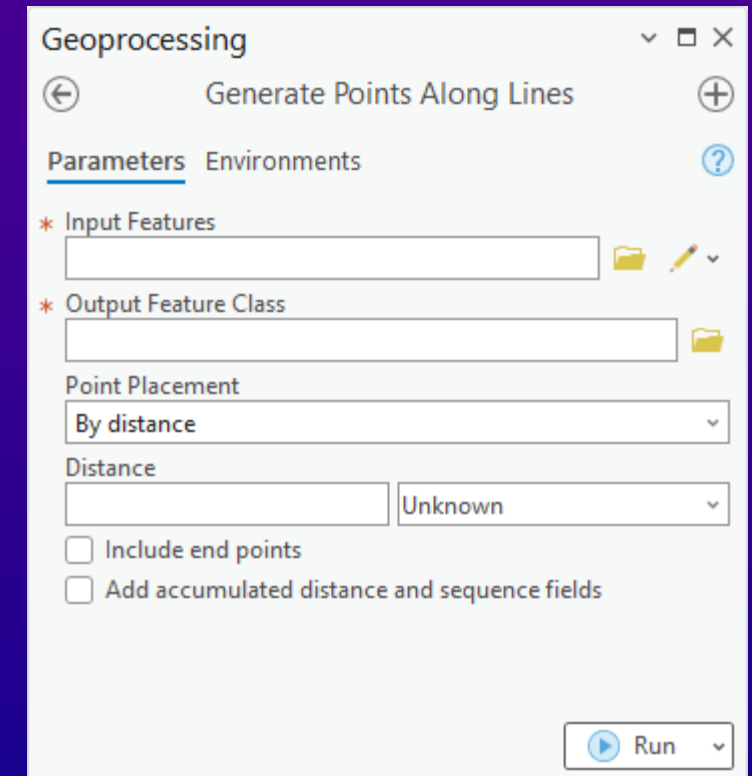
Event: 2023 Esri User Conference

Join us as we step through the process of creating polished, well-designed and useful geoprocessing tools with Python. This session will highlight the important decision in making fully functional geoprocessing tools. Script tools, Python toolboxes and the new ArcGIS toolbox (.atbx) format will be discussed.

Related geoprocessing concepts | Toolbox types | Parameters
Communication | Validation | Tips | And more ...

Why we build tools

- Extend Pro
- Organize functionality
- Your tool becomes part of the geoprocessing framework
- Use your tools in multiple ways
 - Geoprocessing pane
 - Python
 - ModelBuilder
 - Potentially as a geoprocessing service

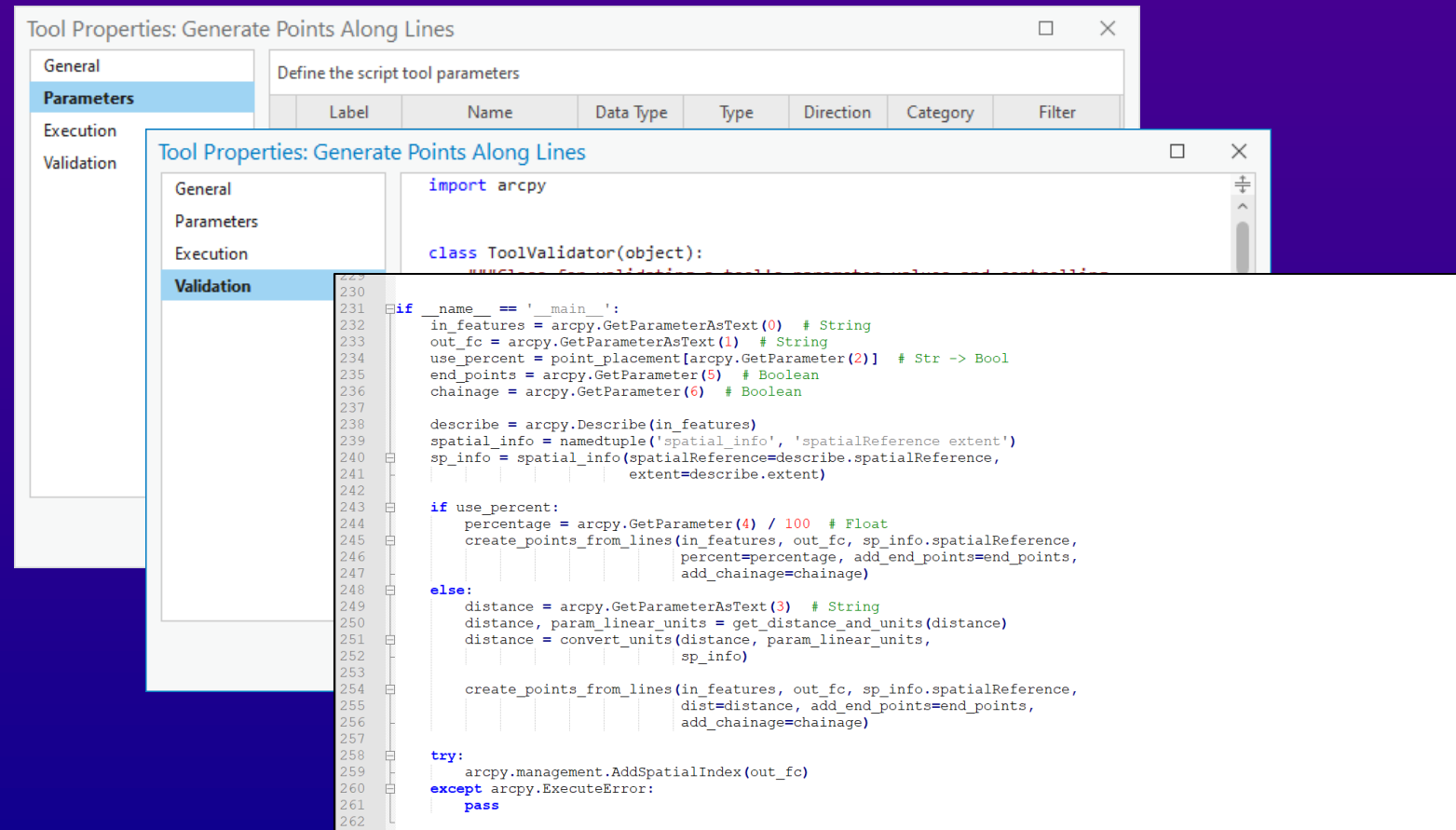


What makes a good geoprocessing tool?

- Performs an essential and elemental operation
- Simple
- Can be combined with other tools for larger processes
- Looks and behaves like other tools
 - For example:
 - A tool always has an output
 - Required parameters precede optional parameters

Tool structure

- Parameters
- Validation code
- Execution code



Toolboxes

- Tools are organized in a toolbox
- We can build Python-based tools in two ways:
 1. ArcGIS toolbox (.atbx) or legacy toolbox (.tbx)
 - Parameters defined through the Pro UI
 - Validation is Python
 - Execution is Python

Toolboxes

- Tools are organized in a toolbox
- We can build Python-based tools in two ways:

2. Python toolbox (.pyt)

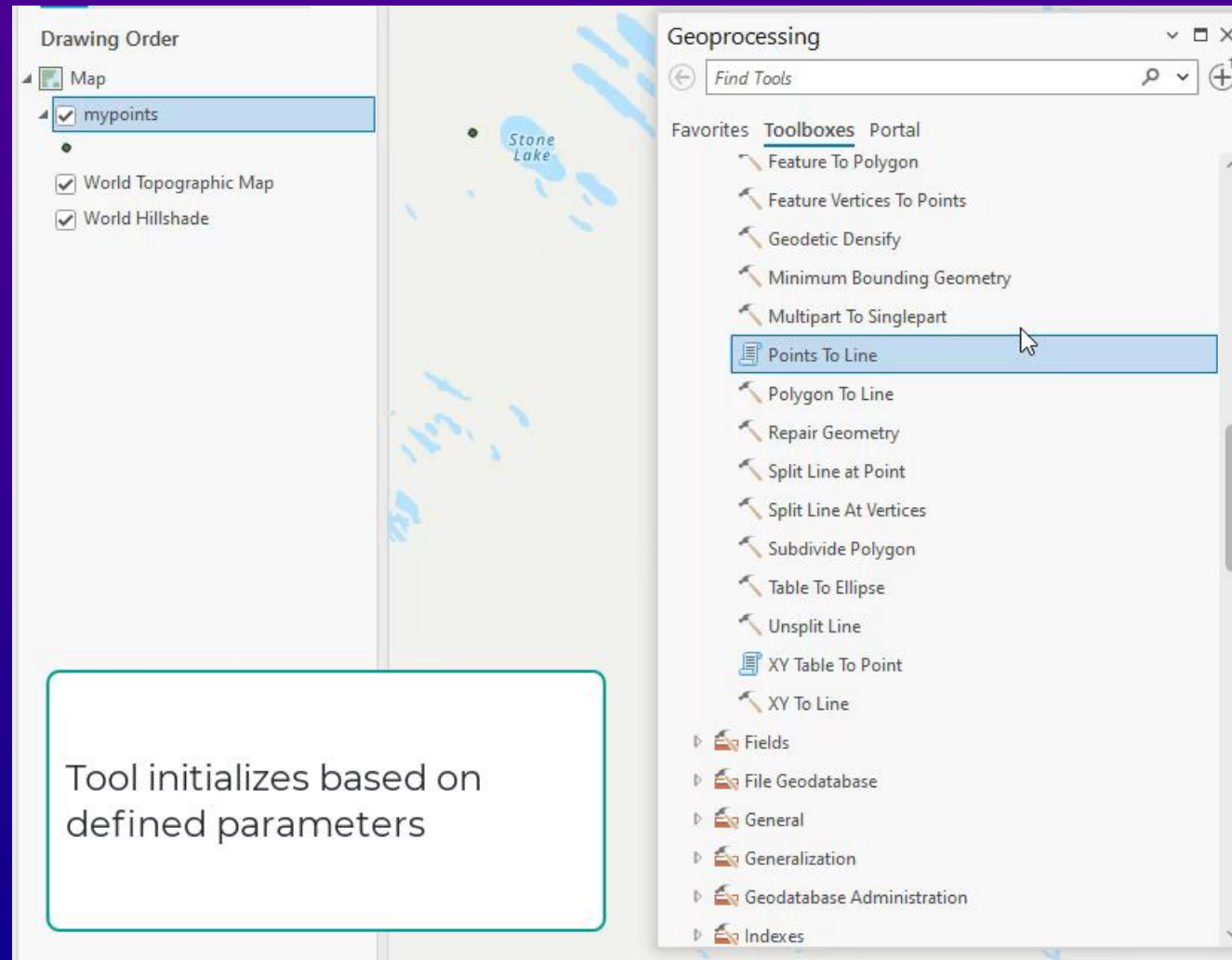
- Parameters are Python
- Validation is Python
- Execution is Python

```
1 import arcpy
2
3
4 class Toolbox(object):
5     def __init__(self):
6         self.label = "Sinuosity toolbox"
7         self.alias = "sinuosity"
8
9         # List of tool classes associated with this toolbox
10        self.tools = [CalculateSinuosity]
11
12
13 class CalculateSinuosity(object):
14     def __init__(self):
15         self.label = "Calculate Sinuosity"
16         self.description = "Sinuosity measures the amount that a river " + \
17                             "meanders within its valley, calculated by " + \
18                             "dividing total stream length by valley length."
19
20     def getParameterInfo(self):
21         #Define parameter definitions
22
23         # Input Features parameter
24         in_features = arcpy.Parameter(
25             displayName="Input Features",
26             name="in_features",
27             datatype="GPFeatureLayer",
28             parameterType="Required",
29             direction="Input")
30
31         in_features.filter.list = ["Polyline"]
32
33         # Sinuosity Field parameter
34         sinuosity_field = arcpy.Parameter(
35             displayName="Sinuosity Field",
36             name="sinuosity_field",
37             datatype="Field",
38             parameterType="Optional",
39             direction="Input")
40
41
```

ArcGIS toolbox format (.atbx)

- Introduced at Pro 2.9
- Similar to the traditional toolbox (.tbx) format
- JSON-based with an open specification
- Stores tools, scripts, and models
- Better cross-release compatibility and persistence

How a tool works



Parameters

- Parameters are how you interact with a tool
- Parameters provide simple rules
 - Does an input exist?
 - Is the input the right type?
 - Is this value an expected keyword?

The screenshot shows the 'Generate Points Along Lines' tool interface. It has a title bar with standard window controls. Below the title bar, there are navigation icons (back, forward) and a search icon. The main area is divided into two tabs: 'Parameters' (selected) and 'Environments'. Under the 'Parameters' tab, there are three main sections: 'Input Features' with a text box and a folder icon; 'Output Feature Class' with a text box and a folder icon; and 'Point Placement' with a dropdown menu set to 'By distance'. Below this is a 'Distance' section with a text box and a dropdown menu set to 'Unknown'. At the bottom, there are two checkboxes: 'Include end points' and 'Add accumulated distance and sequence fields'. A 'Run' button with a play icon is at the bottom right.

Define the script tool parameters											
	Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default	Environment	Symbology
0	Input Feat...	Input_Features	Feature La...	Required	Input		Feature Type				
1	Output Fe...	Output_Feature_Class	Feature Cl...	Required	Output			Input_Feat...			
2	Point Plac...	Point_Placement	String	Required	Input		Value List		DISTANCE		

General

Parameters

Execution

Validation

Define the script tool parameters

	Label	Name	Data Type	Type	Direction	Category	
0	Input Feat...	Input_Features	Feature La...	Required	Input		Fe
1	Output Fe...	Output_Feature_Class	Feature Cl...	Required	Output		
2	Point Plac...	Point_Placement	String	Required	Input		Va
3	Distance	Distance	Linear Unit	Optional	Input		
4	Percentage	Percentage	Double	Optional	Input		Ra
5	Include en...	Include_End_Points	Boolean	Optional	Input		Bo
6	Add accu...	Add_Chainage_Fields	Boolean	Optional	Input		Bo

<

OK

Parameters

Sean Lim

Accessing parameters in the code

- To access parameter values from arcpy, use:
 - GetParameterAsText – values returned as a string
 - GetParameter – values returned as a dynamic type*
- *GetParameter is best for Boolean and numeric types
- For derived parameters, returns values back to the tool using:
 - SetParameterAsText or SetParameter

Communication within the tool (messages)

- Relay information using arcpy message functions
 - AddMessage
 - AddWarning
 - AddError
 - AddIDMessage – use Esri standard ID codes
- Note: Error messages are just messages, they will not end the script
 - Best to exit your code soon after, such as Python's `sys.exit()`


```

"""
from math import radians, sin, cos, asin, sqrt
radius_of_earth_km = 6371
lat1, lng1, lat2, lng2 = list(map(radians, list(point1 + point2)))
d = sin((lat2 - lat1) / 2) ** 2 + cos(lat1) * cos(lat2) * sin((lng2 - lng1) / 2) ** 2
return 2 * radius_of_earth_km * asin(sqrt(d))

__name__ == '__main__':
in_features = arcpy.GetParameterAsText(0) # String
out_fc = arcpy.GetParameterAsText(1) # String
use_percent = point_placement[arcpy.GetParameter(2)] # Str -> Bool
end_points = arcpy.GetParameter(5) # Boolean
chainage = arcpy.GetParameter(6) # Boolean

describe = arcpy.Describe(in_features)
spatial_info = namedtuple('spatial_info', 'spatialReference extent')
sp_info = spatial_info(spatialReference=describe.spatialReference,
                       extent=describe.extent)

if use_percent:
    percentage = arcpy.GetParameter(4) / 100 # Float
    create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                             percent=percentage, add_end_points=end_points,
                             add_chainage=chainage)
else:
    distance = arcpy.GetParameterAsText(3) # String
    distance, param_linear_units = get_distance_and_units(distance)
    distance = convert_units(distance, param_linear_units,
                              sp_info)

    create_points_from_lines(in_features, out_fc, sp_info.spatialReference,
                             dist=distance, add_end_points=end_points,
                             add_chainage=chainage)

try:
    arcpy.management.AddSpatialIndex(out_fc)
except arcpy.ExecuteError:
    pass

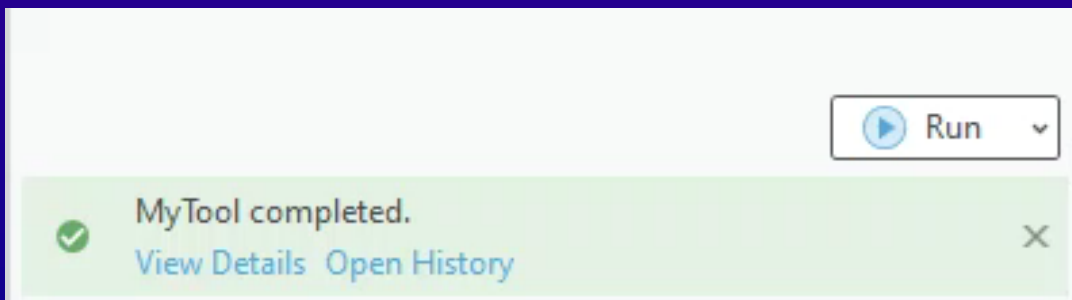
```

Tool source code

Sean Lim

Communication within the tool (progressor)

- Relay simple information to the Geoprocessing pane
- Can provide messages and step increments
 - SetProgressor
 - SetProgressorPosition
 - SetProgressorLabel
 - ResetProgressor



```
5 feature_count = int(arcpy.management.GetCount(in_features)[0])
6
7 # Set up the progressor to update every 5% of the features
8 if feature_count > 20:
9
10     arcpy.SetProgressor(
11         type="STEP",
12         message="Processing features ... ",
13         min_range=0,
14         max_range=100,
15         step_value=5)
16
17     step = feature_count // 20
18
19 for i in range(1, feature_count + 1):
20
21     # Your data processing goes here
22
23
24 if feature_count > 20:
25     if i % step == 0:
26         # Update the progressor message
27         arcpy.SetProgressorLabel(
28             "Processing feature {0}...".format(i))
29
30         # Update the progressor position
31         arcpy.SetProgressorPosition()
```

Parameter validation

- Parameters provide some simple 'free' validation
- Refine your tool's behavior with additional validation
 - Parameter interaction
 - Calculate defaults
 - Enable or disable parameters
 - Set parameter errors and messages
 - Define characteristics of your output (for ModelBuilder)
- Validation runs every time a parameter is modified

```
class ToolValidator:
    """
    Class to add custom behavior and properties to
    the tool and tool parameters.
    """

    ...

    def updateParameters(self):
        """Modify parameter values and properties."""

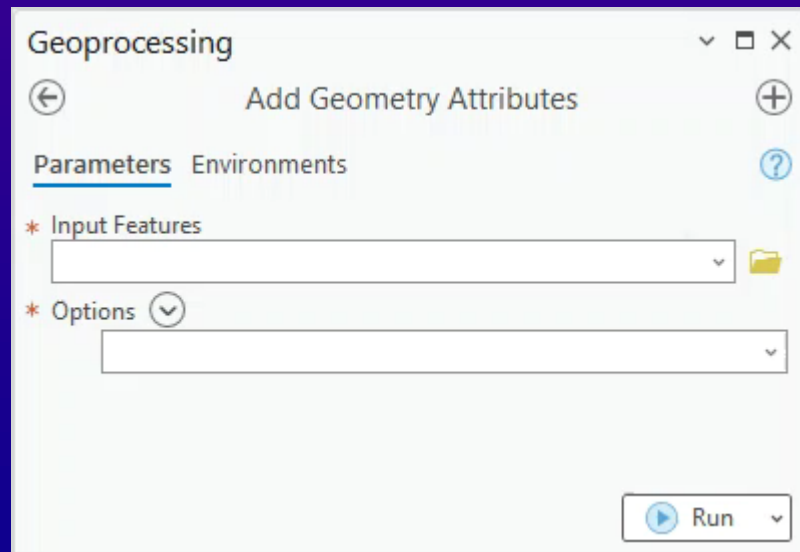
        return

    def updateMessages(self):
        """Customize messages for the parameters."""

        return
```

Validation – updateParameters

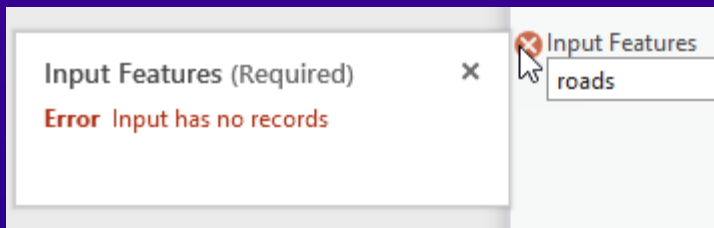
- updateParameters allows you to change certain parameter characteristics
 - Values
 - Filters
 - Enabled
 - Etc.



```
def updateParameters(self):  
    """Modify parameter values and properties."""  
  
    in_features = self.params[0].value  
    if in_features:  
        shape_type = arcpy.Describe(in_features).shapeType  
        if shape_type == 'Polygon':  
            self.params[1].filter.list = ['AREA', 'LENGTH', 'CENTROID']  
  
        elif shape_type == 'Polyline':  
            self.params[1].filter.list = ['LENGTH', 'CENTROID']  
  
        else:  
            self.params[1].filter.list = ['CENTROID']  
    else:  
        self.params[1].filter.list = ['AREA', 'LENGTH', 'CENTROID']  
  
    return
```

Validation – updateMessages

- updateMessages allows you provide warnings or errors before running the tool
- Provides information in Geoprocessing pane in real time

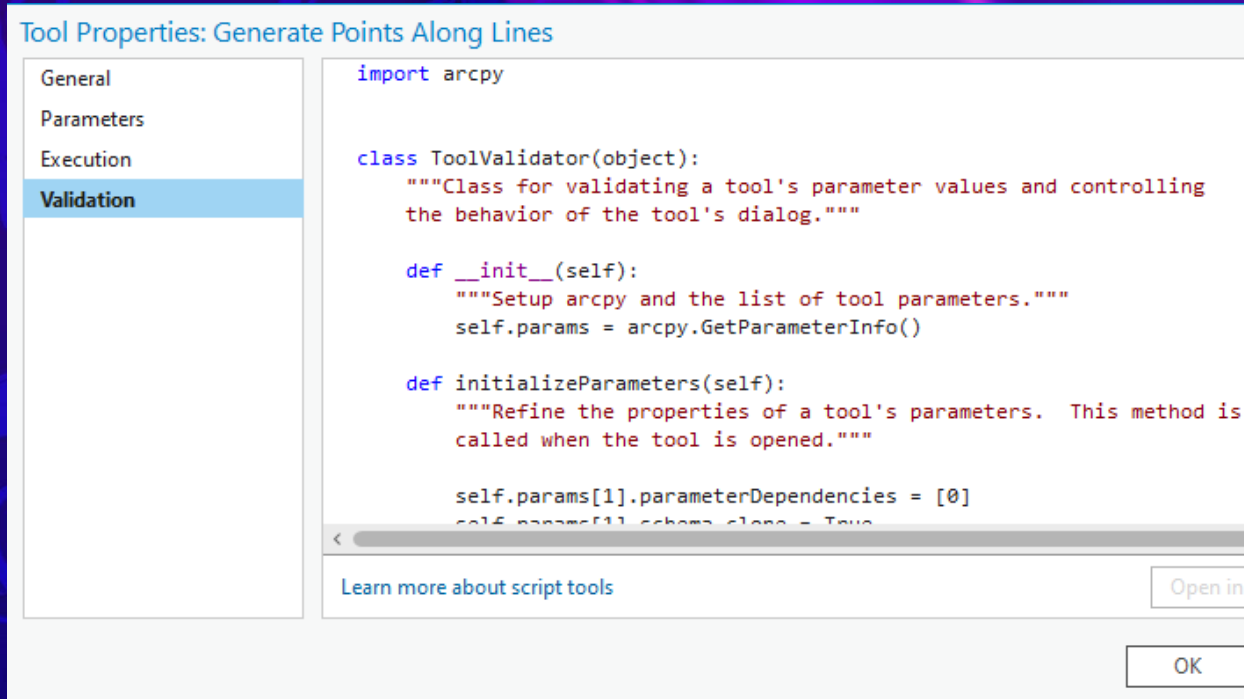


```
def updateMessages(self):  
    """Customize messages for the parameters."""  
  
    in_features = self.params[0].value  
  
    if in_features:  
        selection = arcpy.Describe(in_features).FIDSet  
  
        if not selection:  
            self.params[0].setErrorMessage('Input has no selection')  
  
    return
```

- Note: only use message methods in updateMessages

Validation (and metadata)

Sean Lim



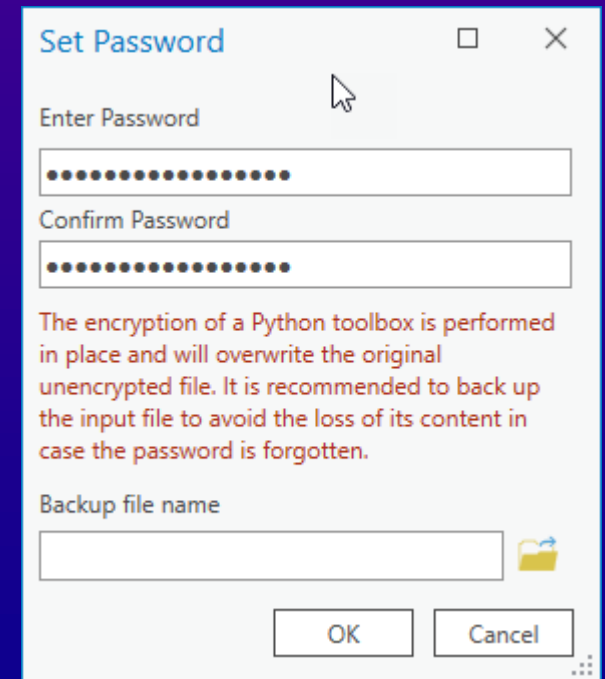
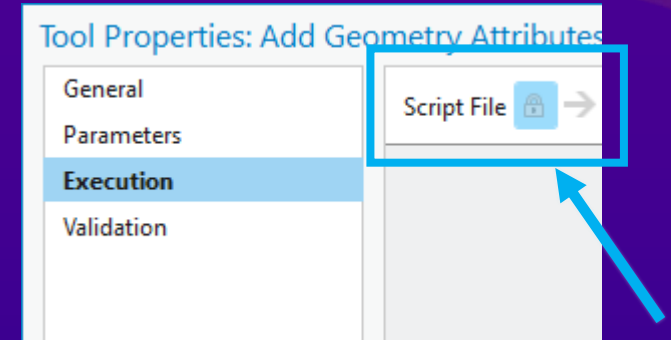
Symbology

- Use a layer file to set a parameter's symbology property
- Or, use the postExecute validation method (*new at 3.0*)
 - Is called when a tool completes
 - Use the arcpy.mp module

```
def postExecute(self):  
    """This method takes place after outputs are processed and added to the display."""  
  
    try:  
        project = arcpy.mp.ArcGISProject('CURRENT')  
        active_map = project.activeMap  
  
        if active_map:  
            out_layer = active_map.listLayers(os.path.basename(self.params[0].valueAsText))[0]  
  
            symbology = out_layer.symbology  
            symbology.updateRenderer('SimpleRenderer')  
            symbology.renderer.symbol.applySymbolFromGallery('Airport')  
            symbology.renderer.symbol.size = 12  
            out_layer.symbology = symbology  
  
    except Exception:  
        pass  
  
    return
```

Embedding and encryption

- You can embed code in .atbx and .tbx toolboxes
 - One less file to manage
- Once embedded the code can be encrypted
- Python toolboxes also support encryption
 - The entire .pyt file is encrypted



Work with geoprocessing samples

ArcGIS Pro 3.1 | [Other versions](#) ▾ | [Help archive](#)

A geoprocessing sample is a convenient method to share geoprocessing tools and workflows. A geoprocessing sample is a .zip file containing toolboxes, scripts, models, data, and supporting files that run geoprocessing tools in ArcGIS. A geoprocessing sample can be a local .zip file on your desktop or on a network folder. Or, a geoprocessing sample can be shared and hosted in ArcGIS Online or your ArcGIS Enterprise portal.

Contents in ArcGIS Online or your ArcGIS Enterprise portal.

You and others can view and use the tools and data inside the geoprocessing sample by adding it to your ArcGIS Pro project.

Geoprocessing samples

Sean Lim

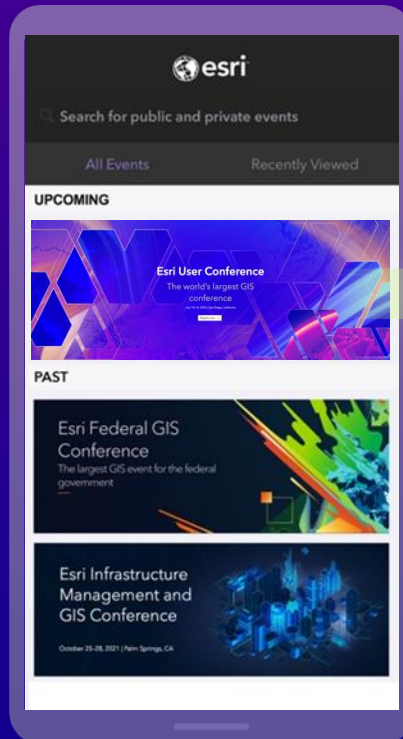
<https://pro.arcgis.com/en/pro-app/latest/help/analysis/geoprocessing/share-analysis/geoprocessing-samples.htm>

Come say hi

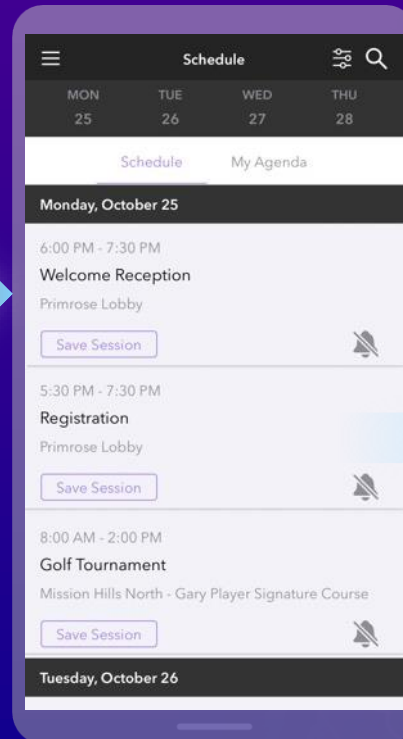


Please Share Your Feedback in the App

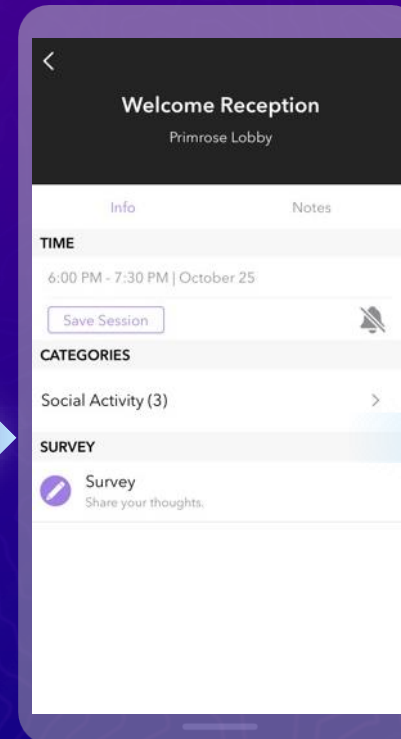
Download the Esri Events app and find your event



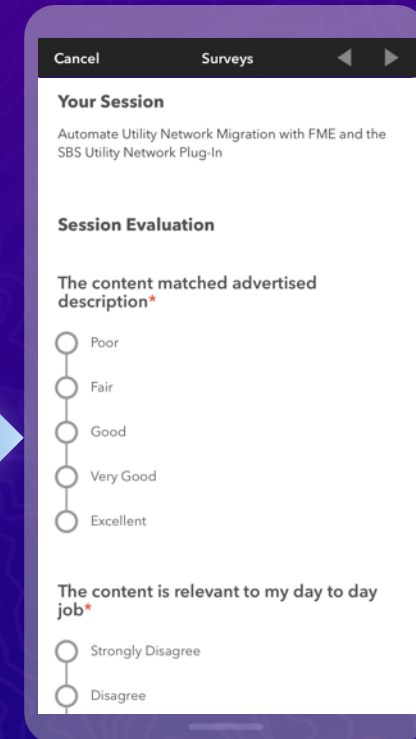
Select the session you attended



Scroll down to "Survey"



Log in to access the survey





esri[®]

**THE
SCIENCE
OF
WHERE[®]**