PYTHON:
BUILDING GEOPROCESSING TOOLS

David Wynne, Andrew Ortego

Python: Building Geoprocessing Tools

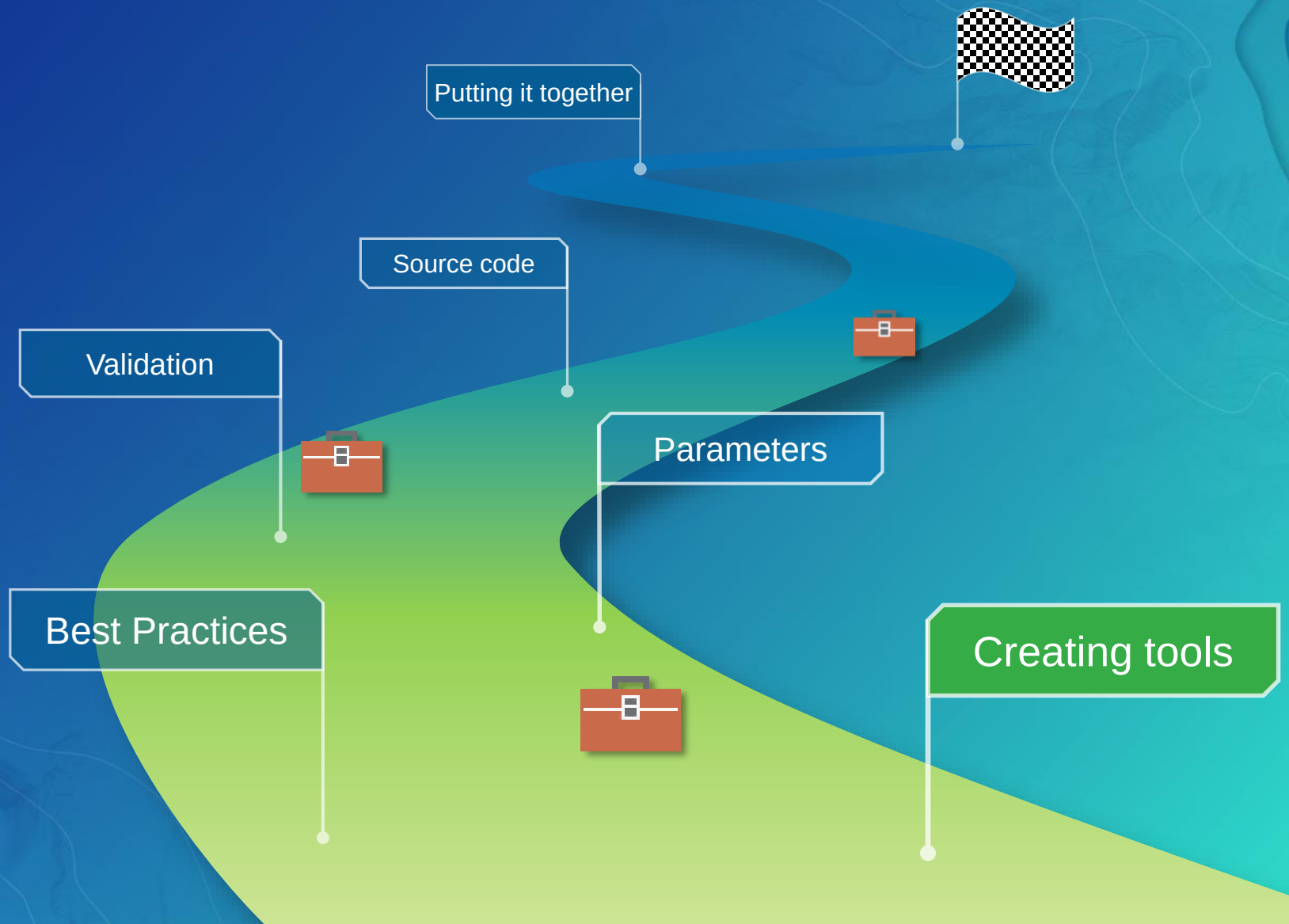
Being able to build a geoprocessing tool from Python is a fundamental building block for adding your own custom functionality into ArcGIS. Join us as we step through the process of taking your Python code and turning it into fully functional geoprocessing tools. Both script tools and Python toolboxes will be explored.

Categories - - Technical Workshops, Performing Analysis

★	Tuesday, July 11	Python: Building Geoprocessing Tools 1:30pm - 2:45pm SDCC - Ballroom 06 E
☆	Friday, July 14	Python: Building Geoprocessing Tools 9:00am - 10:15am SDCC - Room 01 A

http://esriurl.com/CreatingTools | http://esriurl.com/CreatingToolsPro

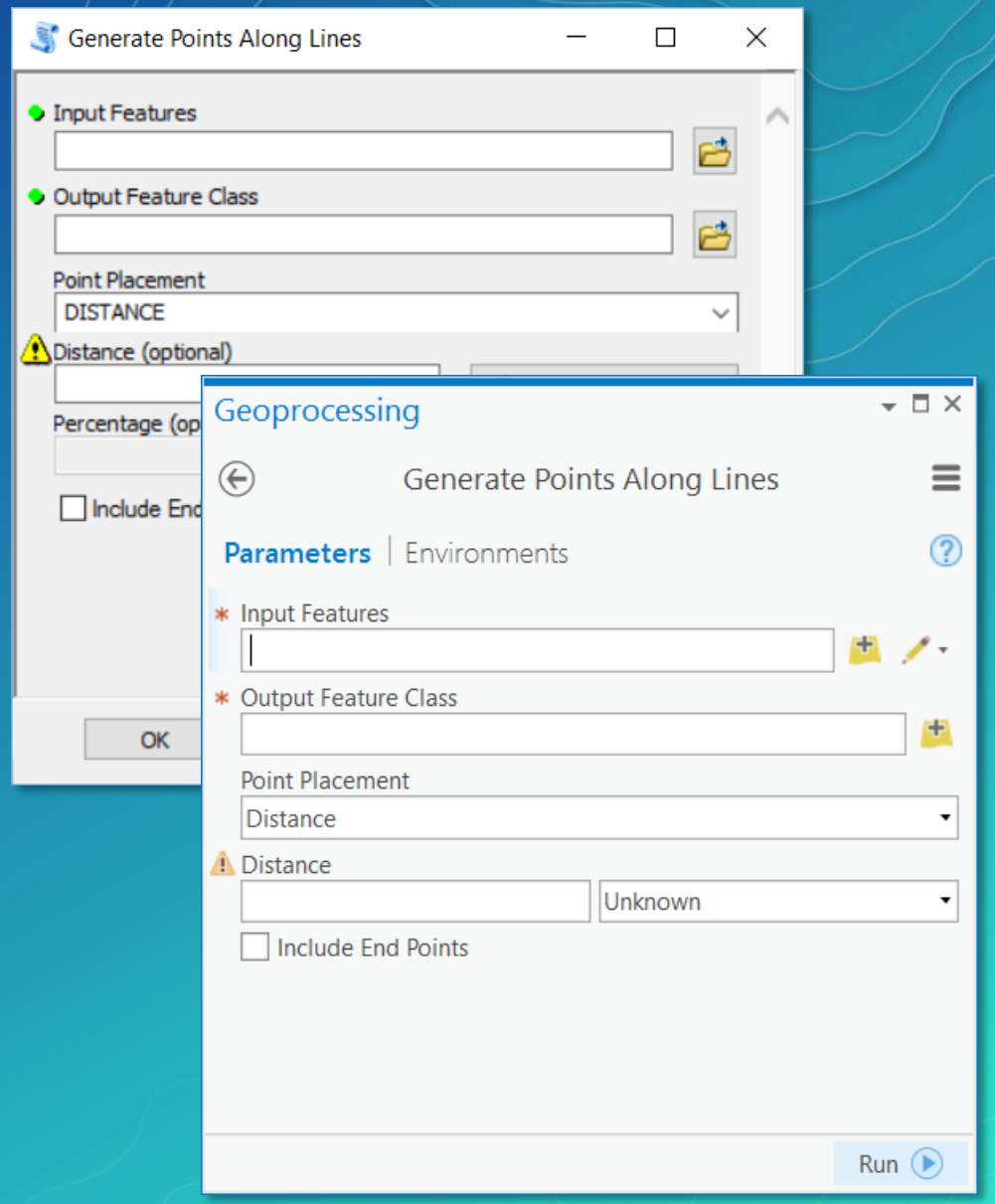
Today


Putting it together
Source code
Validation
Parameters
Best Practices
Creating tools

Why we build geoprocessing tools

- Your work becomes part of the geoprocessing framework

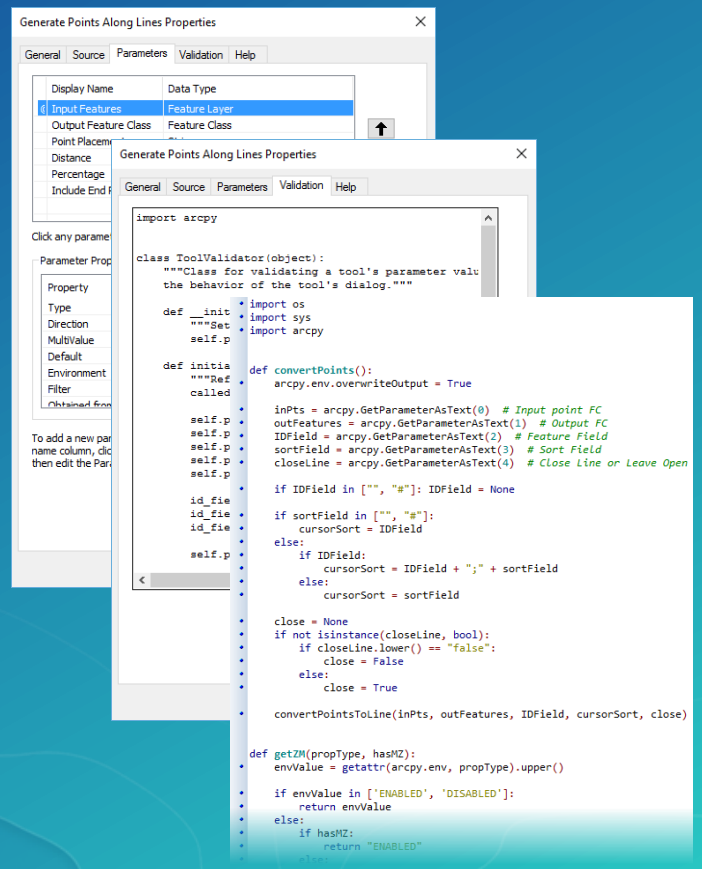
- Easy to access and run from within ArcGIS
- Familiar look and feel
- Make a mistake?
  - Re-run from the previous result
- Run from anywhere you can run a tool
  - Run from Python, ModelBuilder, a service
- Supported in multiple products
- No UI programming



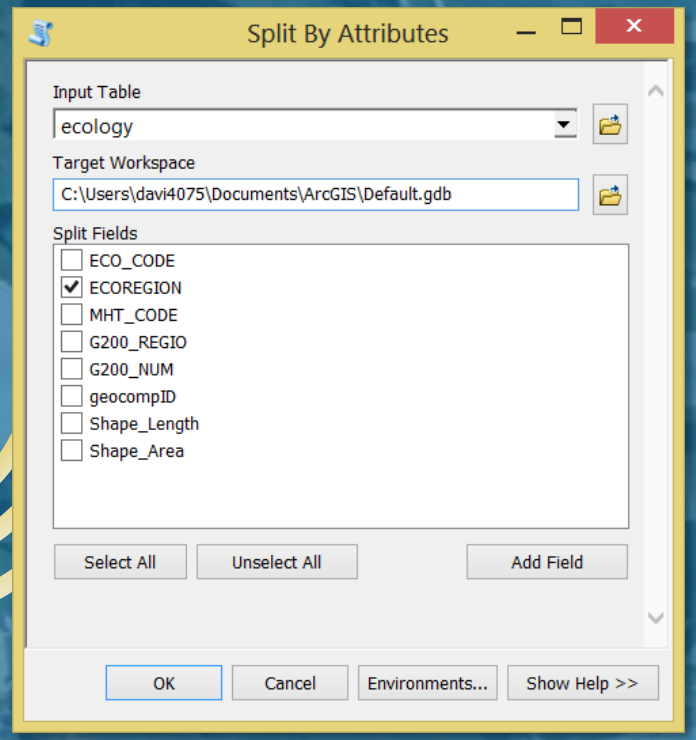
Tool Recipe

- A geoprocessing tool is made from 3 main ingredients

1. Parameters
2. Parameter validation
3. Source code to be executed



The geoprocessing tool




```
'''
Python code

  Tool-specific behaviors

  1. Receive arguments
  2. Tool messages
  3. Progressor
  4. Cancellation behaviors
  5. Send arguments

'''

import arcpy

# More code
```

Validation

1. Update parameters
2. *Internal validation*
3. Update messages

Geoprocessing Tool Commandments

Thou shall …

i. Have unique parameter names within the tool
ii. Keep the cost of validation to a minimum
iii. Always have an output, even if it must be derived
iv. Populate all output data elements within validation
v. Not test the validity of any derived value within validation
vi. Have a coded value domain for every Boolean
vii. Test the function from a script, a model, a dialog, and the command line

A couple more…

- Default values should not raise errors
- Parameter ordering patterns:
  1. Input datasets
  2. Output datasets
  3. Other

  Required

  4. Input datasets
  5. Other
  6. Output datasets
  7. Derived outputs

  Optional

Turning Python code into geoprocessing tools

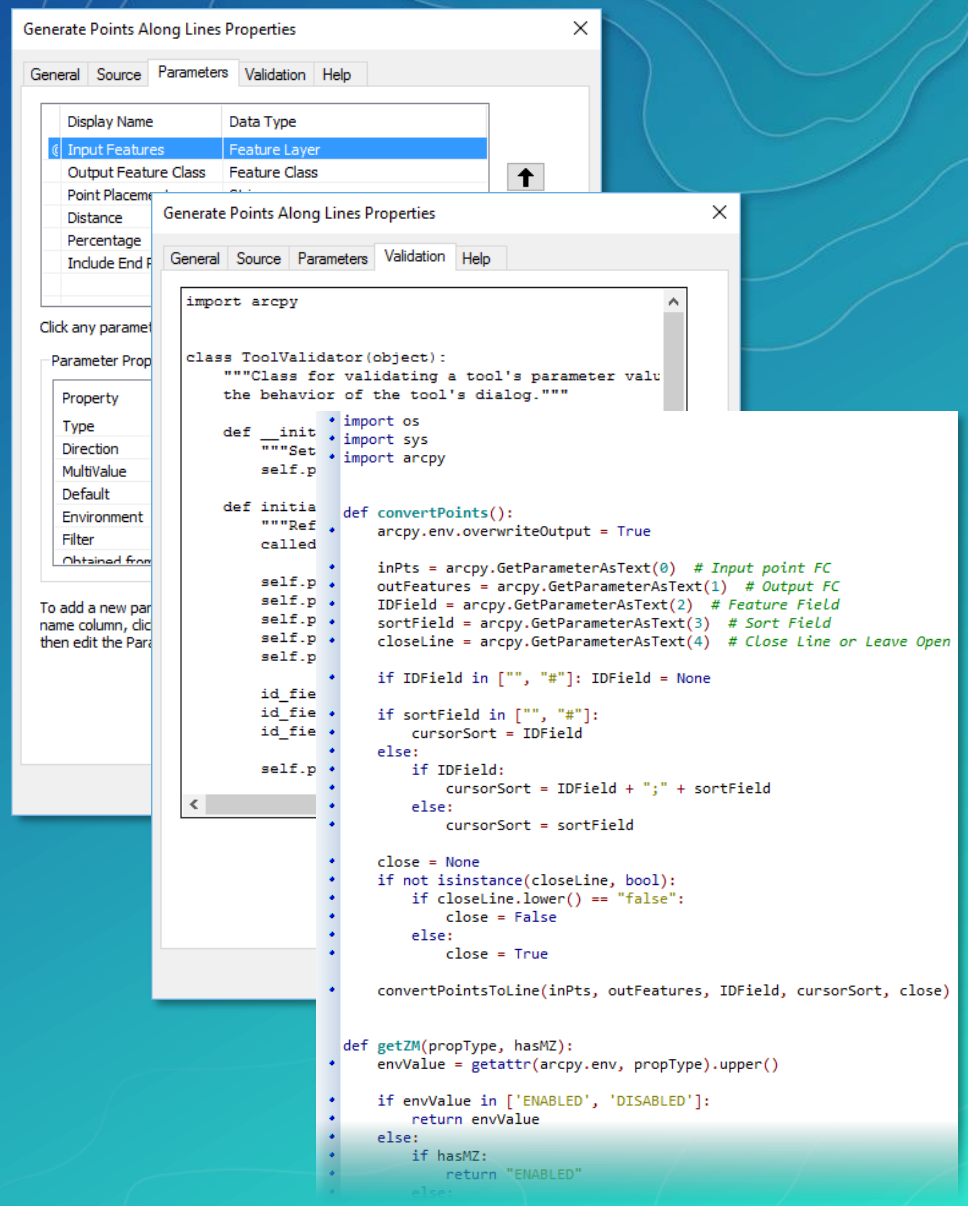
Demo

Script tools vs Python toolboxes

- **Using Python, we can build tools in two ways:**

Script tools

- Source is Python
- Parameters through wizard
- Validation is Python (stored in toolbox)



Script tools vs Python toolboxes

- **Using Python, we can build tools in two ways:**

Python toolboxes

- Source is Python
- Parameters are Python
- Validation is Python

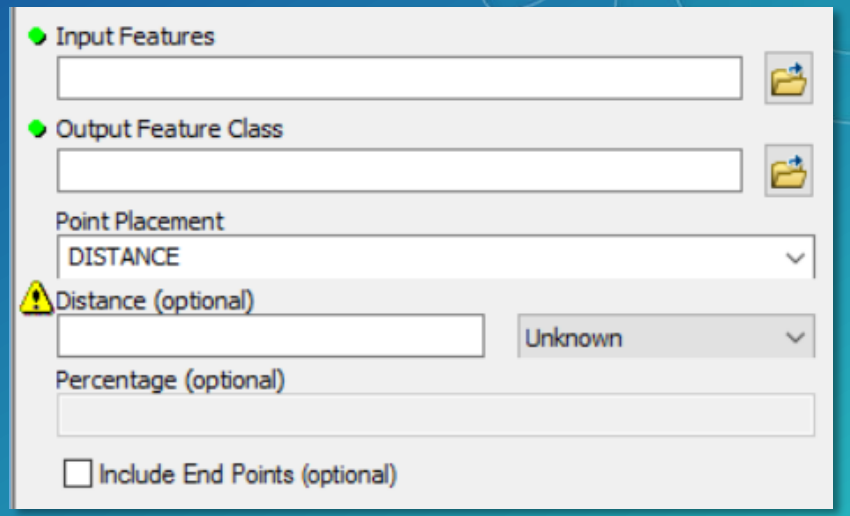
- **Which do I use?**
  - **“A tool is a tool”**

```python
import arcpy


class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name of the
        .pyt file)."""

        self.label =  "Sinuosity toolbox"
        self.alias  = "sinuosity"

        # List of tool classes associated with this toolbox
        self.tools = [CalculateSinuosity]

class CalculateSinuosity(object):
    def __init__(self):
        self.label       = "Calculate Sinuosity"
        self.description = "Sinuosity measures the amount that a river " + \
                           "meanders within its valley, calculated by " + \
                           "dividing total stream length by valley length."

    def getParameterInfo(self):
        """Define the tool (tool name is the name of the class)."""

        in_features = arcpy.Parameter(
            displayName="Input Features",
            name="in_features",
            datatype="GPFeatureLayer",
            parameterType="Required",
            direction="Input")

        in_features.filter.list = ["Polyline"]

        sinuosity_field = arcpy.Parameter(
            displayName="Sinuosity Field",
            name="sinuosity_field",
            datatype="Field",
            parameterType="Optional",
            direction="Input")

        sinuosity_field.value = "sinuosity"

        out_features = arcpy.Parameter(
            displayName="Output Features",
            name="out_features",
            datatype="GPFeatureLayer",
            parameterType="Derived",
            direction="Output")

        out_features.parameterDependencies = [in_features.name]
        out_features.schema.clone = True

        parameters = [in_features, sinuosity_field, out_features]

        return parameters
```

Parameters

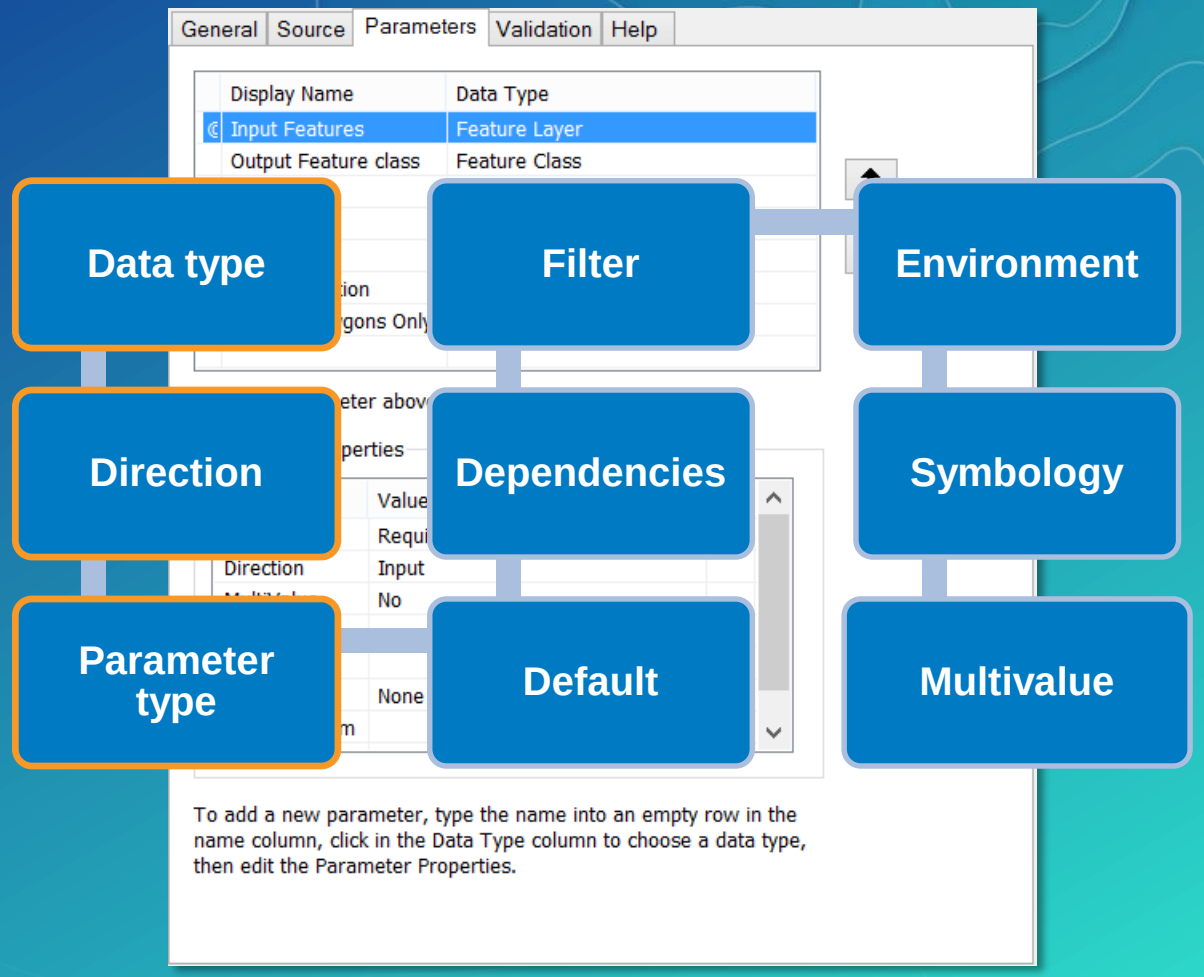
- Parameters are how you interact with a tool

- Simple rules to guide behaviors
  - Does an input exist?
  - Is the input the right type?
  - What are valid fields for this data?
  - Is this value an expected keyword?



Parameter properties

1. Data type
   - Feature Layer, Raster Layer, Table View, …
   - String, Boolean, Long, Float, …
2. Direction
   - Input, Output
3. Parameter type
   - Required, Optional, Derived



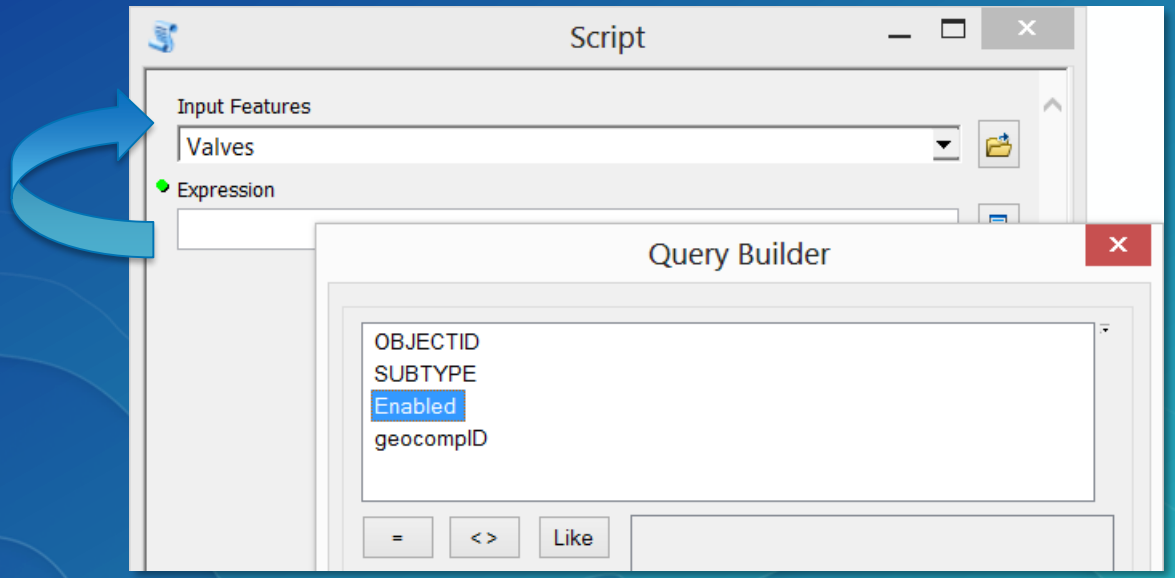
Filters

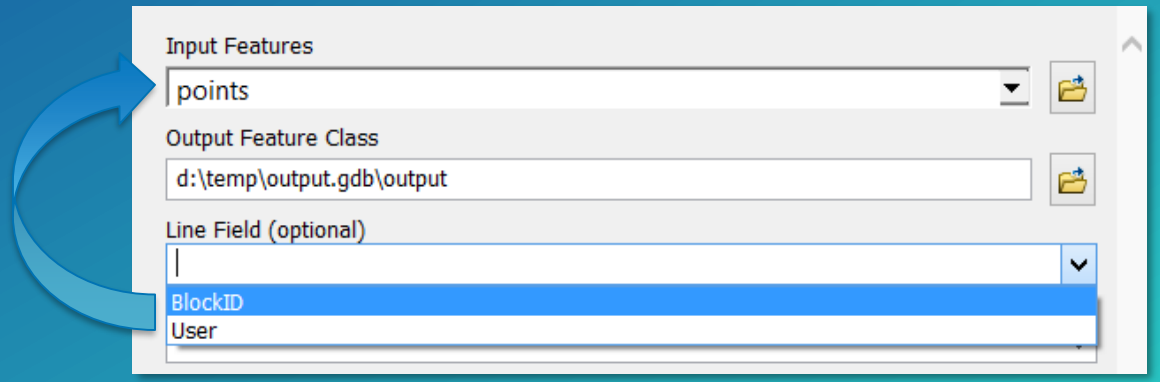
- **Filters are used to limit choices to acceptable values**
- **For example:**
  - **A number between 1 and 10**
  - **A string parameter with keywords**
  - **A file parameter that will only accept files with a .txt extension**

Filter	Values	Relevant data types
Value List	String (**keywords**) or **numeric** values	String, Long, Double, Boolean
Range	Between a **minimum** and **maximum** value	Long, Double
Feature Class	Allowable feature class types	Feature Layer, Feature Class
File	File extensions	File
Field	Supported field types	Field
Workspace	Supported workspaces	Workspace

Parameter Dependencies / ‘obtained from’

- Some data types are dependent on other parameters
  - Field, SQL Expression, Field Info provide little value on their own
- Set a dependency to a different data type





Parameters - Symbology

- Control the appearance of output by setting symbology
- Set using a layer file either as :

1. Parameter property

Parameter Properties

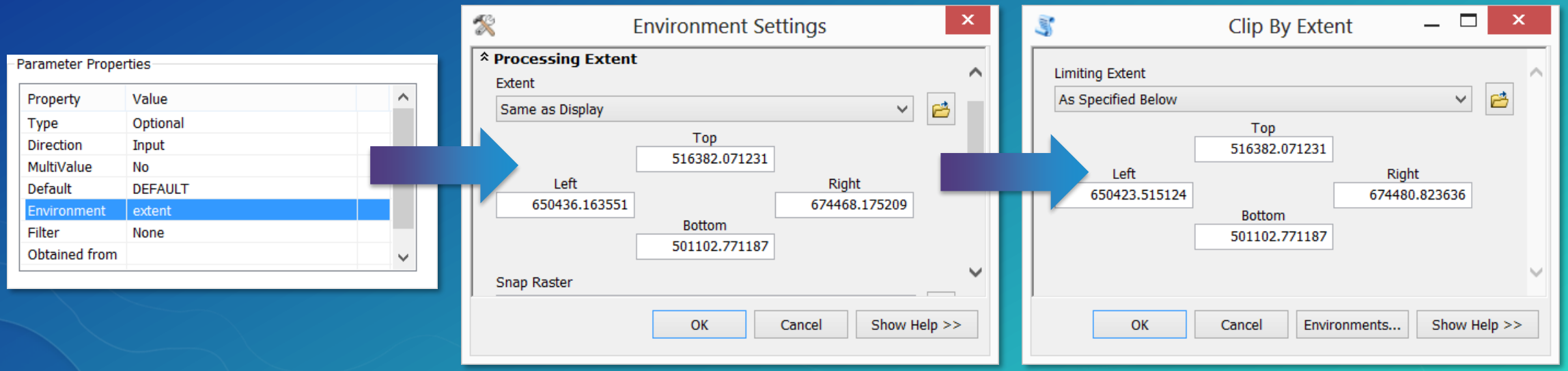
Property	Value
Direction	Output
MultiValue	No
Default	
Environment	
Filter	None
Obtained from	
Symbology	mo2\featureclasses_classname.lyr ...

2. Or in your source code
   - *Important if you want to vary the symbology based on logical criteria*

```
# Set the symbology
params = arcpy.GetParameterInfo()
if len(params) > 0:
    params[1].symbology = os.path.join(layerfile_folder,
                                       'poly_symbology.lyr')
```

Parameters - Defaults

- Set a parameter default by value or by environment
  - Some data types match closely with geoprocessing environments
  - Can configure a parameter to use an environment by default



Parameter-izing a tool

Demo

Parameters – Validation

- As parameters are entered and modified, validation responds and interacts
  - Everything that happens before pushing OK

- Tools validate parameters in two ways

1. Basic or ‘free’ validation, such as:
   - Does the input (or output) exist?
   - Is it the right data type?
   - Does this value match its filter?
   - Have all required parameters been supplied?

2. Additional rules and behavior you add

Validation

- Provides more control
  - Parameter interaction
  - Calculate defaults
  - Enable or disable parameters
- Setting parameter errors and messages
- Defining output characteristics
  - *Chain tools in ModelBuilder*

Validation

- Validation is about responding to changes in:

- `value` / `valueAsText`
  - Does a parameter have a value?
  - What is the value?
  - Properties of the data (`arcpy.Describe`)
- `altered`
  - Has the parameter been altered?
- `hasBeenValidated`
  - Has internal validation checked the parameter?

```python
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."""

    if self.params[0].value:
        if not self.params[2].altered:
            extent = arcpy.Describe(self.params[0].value).extent
            if extent.width > extent.height:
                self.params[2].value = extend.width / 100.0
            else:
                self.params[2].value = extent.height / 100.0

    return
```

```python
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."""

    if self.params[0].value:
        p = feedparser.parse(self.params[0].valueAsText)

        if p['bozo'] == 0:  # Successful read
            entry = p.entries[0]
            field_names = entry.keys()
            field_names.remove('georss_point')
            field_names.remove('georss_elev')
            self.params[2].filter.list = field_names
```

A quick note about Python toolboxes and script tools

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

    # Distance should never be negative
    if parameters[2].value <= 0.0:
        parameters[2].setErrorMessage(
            'Distance value cannot be a negative n

    # If using percentages, distance must be less
    elif parameters[3].value:
       if parameters[2].value > 1.0:
            parameters[2].setErrorMessage(
                'Percentages must be between 0.0 a
    return
```

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

    # Distance should never be negative
    if self.params[2].value <= 0.0:
        self.params[2].setErrorMessage(
            'Distance value cannot be a negative number')

    # If using percentages, distance must be less than 1.0
    elif self.params[3].value:
       if self.params[2].value > 1.0:
            self.params[2].setErrorMessage(
                'Percentages must be between 0.0 and 1.0')

    return
```

Validation: ModelBuilder

Your tool → ???

- Describe outputs for chaining in ModelBuilder
- By updating `schema` of outputs, subsequent tools can see pending changes prior to execution

```
self.params[1].parameterDependencies = [0]
self.params[1].schema.clone = True
self.params[1].schema.geometryTypeRule = 'AsSpecified'
self.params[1].schema.geometryType = 'Point'
self.params[1].schema.fieldsRule = 'FirstDependencyFIDs'

id_field = arcpy.Field()
id_field.name = 'ORIG_FID'
id_field.type = 'Integer'

self.params[1].schema.additionalFields = [id_field]
```

Validation: Messages

- Add more stringent errors based on your own criteria
- Evaluate system messages, and relax or change

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

    # Distance should never be negative
    if self.params[2].value <= 0.0:
        self.params[2].setErrorMessage(
            'Distance value cannot be a negative number')

    # If using percentages, distance must be less than 1.0
    elif self.params[3].value:
       if self.params[2].value > 1.0:
            self.params[2].setErrorMessage(
                'Percentages must be between 0.0 and 1.0')
```

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

    has_error = self.params[0].hasError()
    message = self.params[0].message

    # Watch for "ERROR 000800: The value is not a member of ..."
    if has_error and message.find('000800') > -1:
        self.params[0].setWarningMessage(message)
```

Working with validation

Demo

Making your Python code work as a tool

1. Parameters are received
   - Script tool – `GetParameter` or `GetParameterAsText`
   - Python toolbox – `parameter.value` or `.valueAsText`

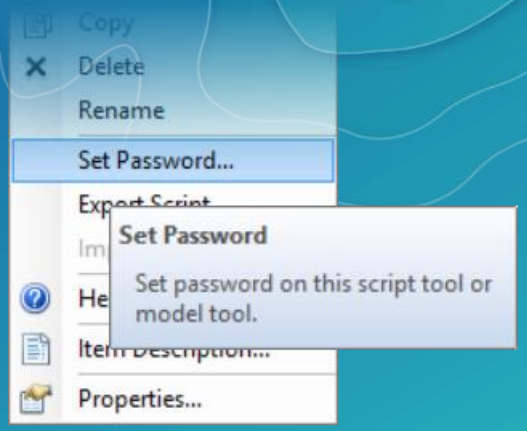
2. arcpy internals
   - Messages – `arcpy.AddMessage`, `AddWarning`, `AddError`
   - Progressor
   - Optionally, control actions after a cancellation

3. Derived values, if any, are pushed back to the tool

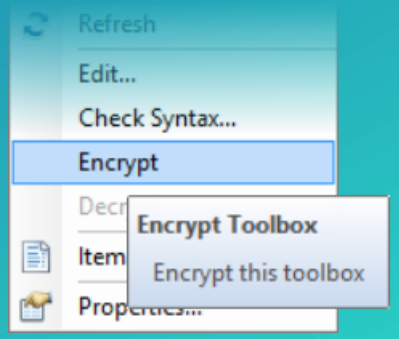
```
'''
Python code

  Tool-specific behaviors

  1. Receive arguments
  2. Tool messages
  3. Progressor
  4. Cancellation behaviors
  5. Send arguments

'''

import arcpy

# More code
```

Keeping your work private

- Script tools have supported encryption for many releases
  - Embed, then set a password



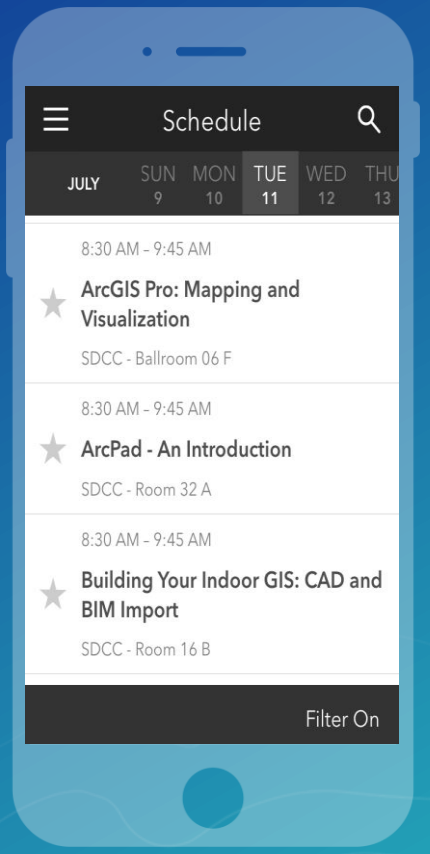
- Python toolboxes support encryption at 10.5, Pro 1.3
  - Encrypt the toolbox in one step
  - Python toolboxes are encrypted in place



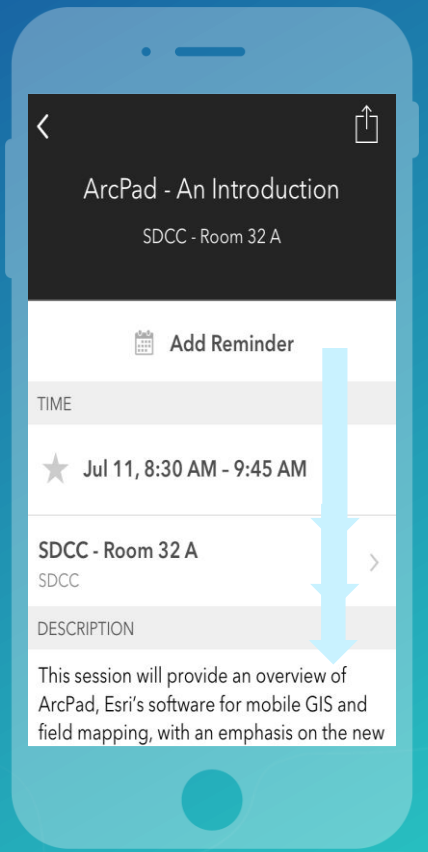
Please Take Our Survey on the **Esri Events App!**

Download the Esri Events app and find your event

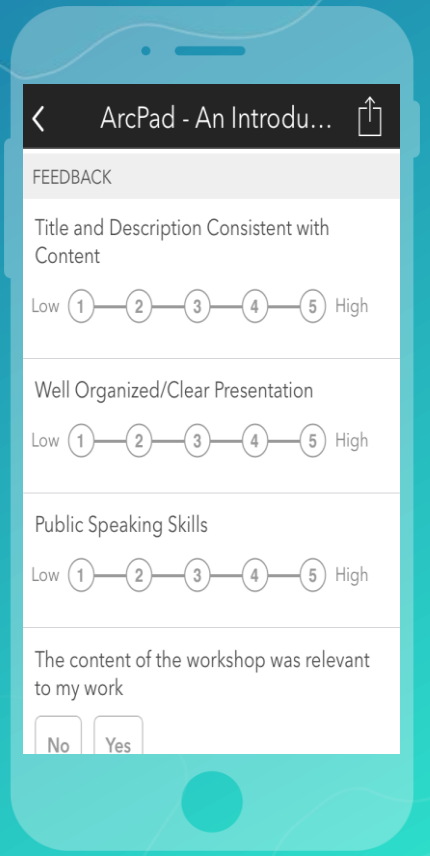
Select the session you attended



Scroll down to find the survey



Complete Answers and Select "Submit"



esri

THE SCIENCE OF WHERE