Building Geoprocessing Tools with Python: *Getting Started*

Dave Wynne

Andrew Ortego

Dave

- Building GP tools since 2003
- Roles (past and present):
  - Python Lead
  - GP Documentation Lead
- Dog guy

Drew

- Works at Esri?
- Better hair
- More of a cat guy

Building Geoprocessing Tools with Python: Getting Started

Join us as we step through the process of creating geoprocessing tools using Python. Using both script tools and Python toolboxes as examples, this workshop will highlight the important decisions in making fully functional geoprocessing tools.

---

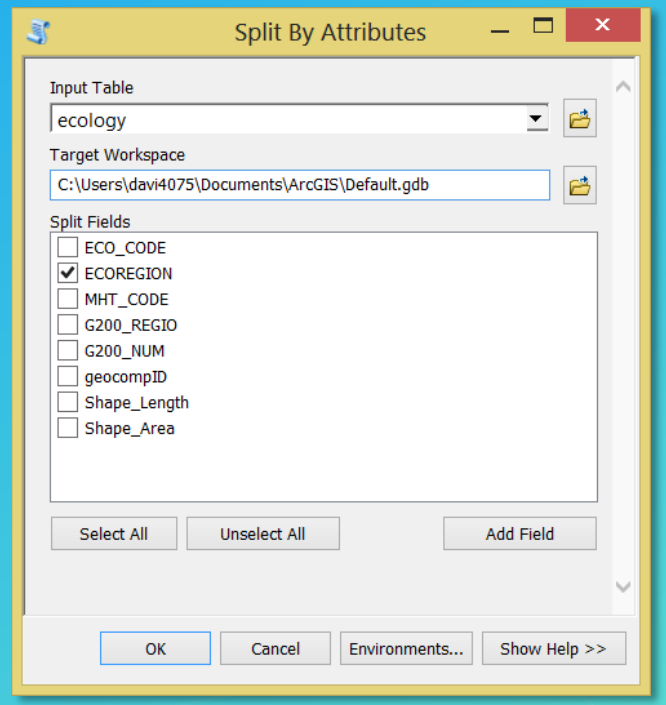
basics | tool mechanics | design | script tools |
Python toolboxes | parameters | validation | migration

Overview

- Creating a tool, allows you to benefit from the geoprocessing framework

- Use your tool as a …
  - … dialog (but no UI programming)
  - … Python function
  - … tool in ModelBuilder
  - … service, package

- Supported in multiple products
- Your tools can have a long history (and future)



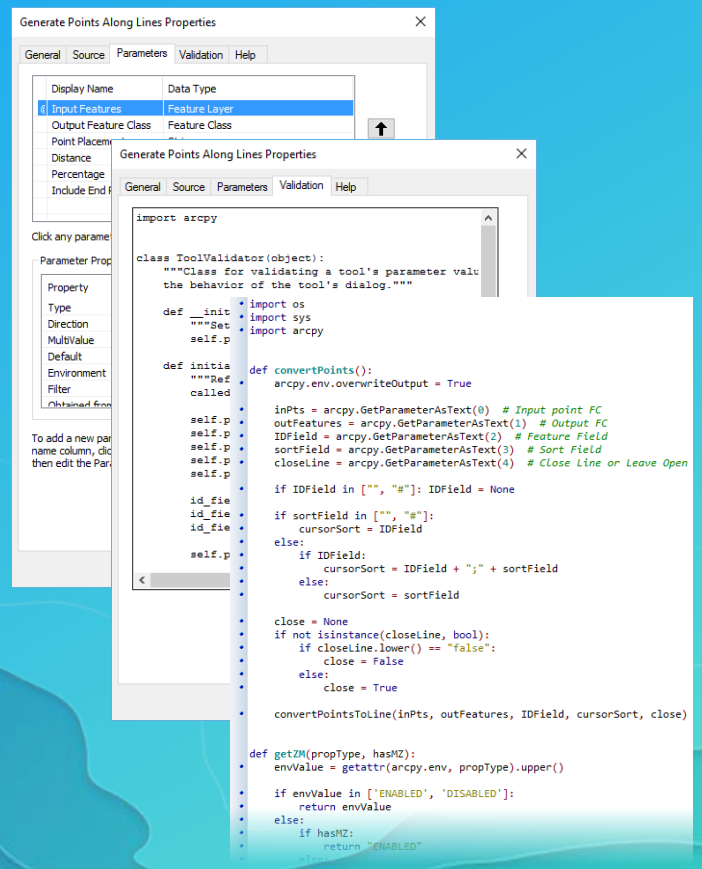
Geoprocessing tools – Fun facts

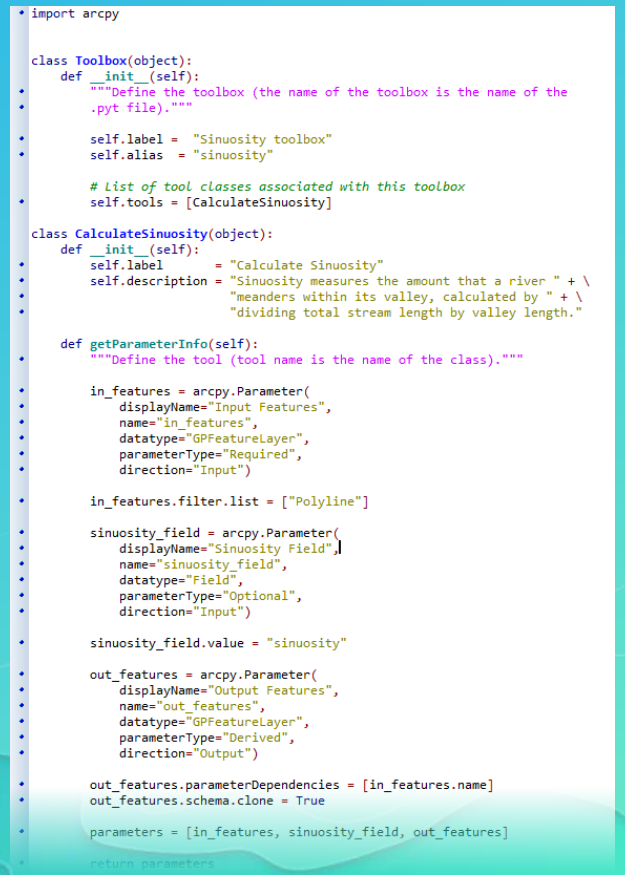
- Established 2004 (ArcGIS 9.0)
- 1200 tools included with Pro at 2.1
  - 10% are Python-based
- Two varieties of Python-based tools
  - Script tools – ArcGIS 9.0
  - Python toolboxes – ArcGIS 10.1

Tool recipe

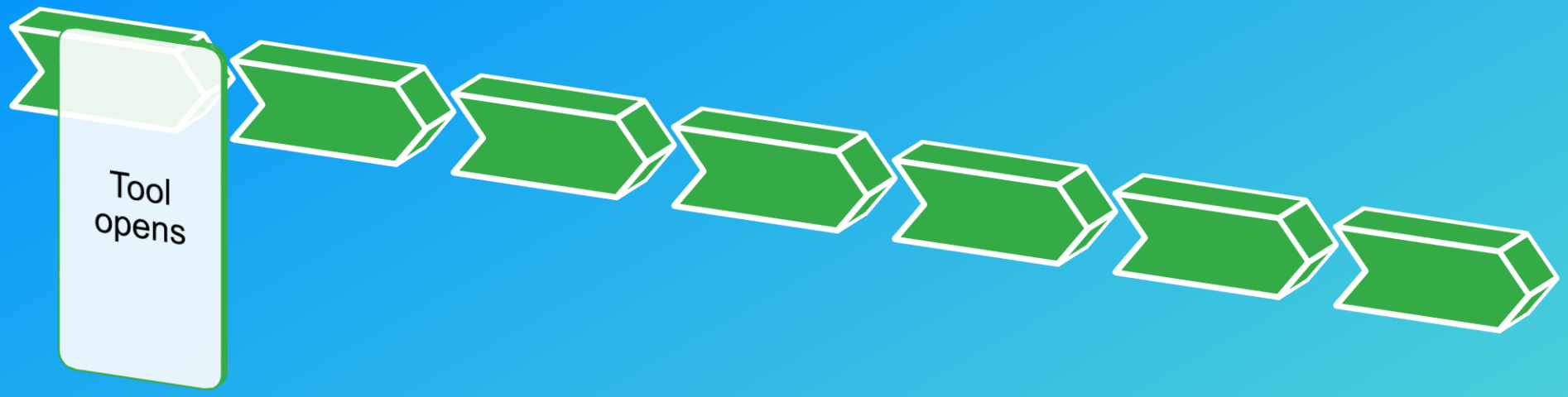
- A geoprocessing tool is made from 3 main ingredients

1. Parameters
2. Validation
3. Source code



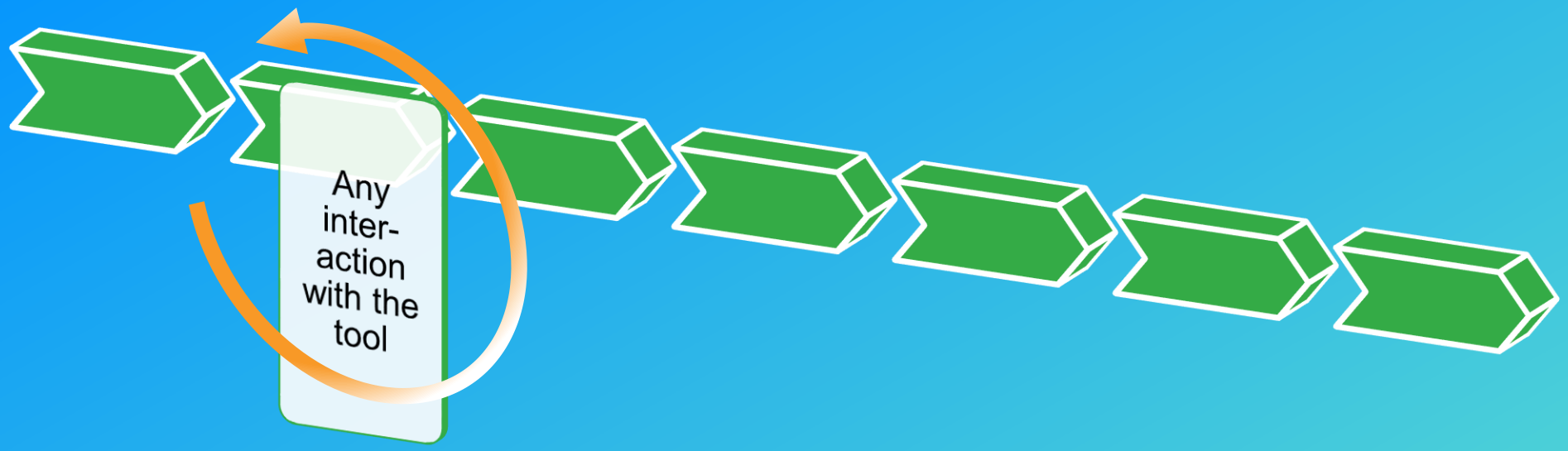


How a tool works



- Tool parameters are initialized based on their definitions

How a tool works

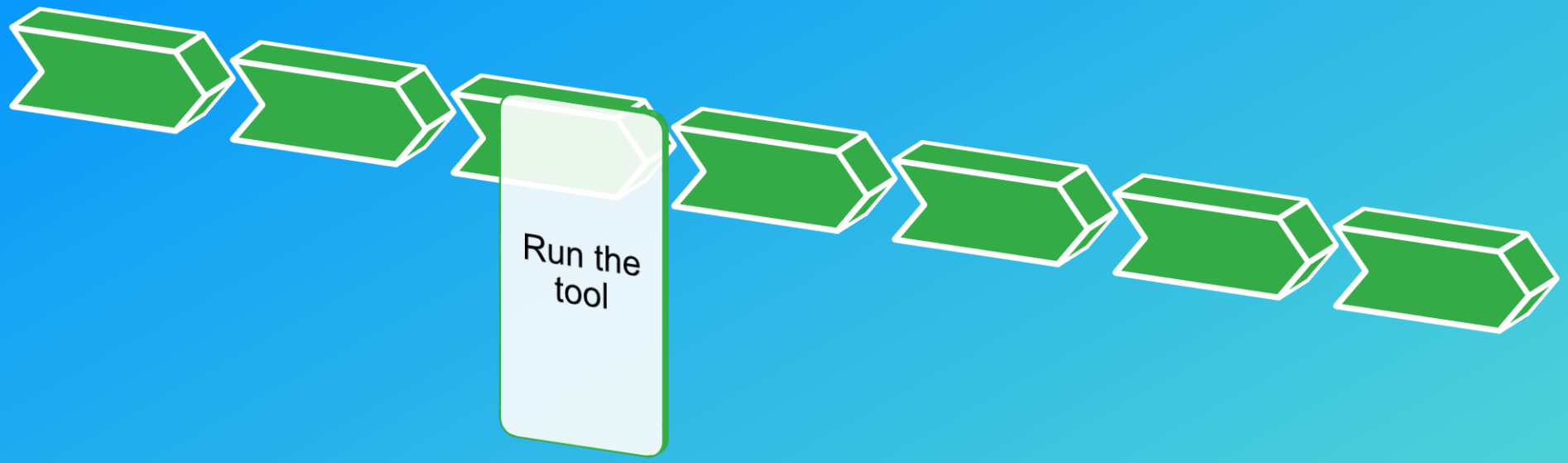


- Following steps occur:
  - updateParameters (written by you)
  - Internal validation is called
  - updateMessages (written by you)

Internal validation

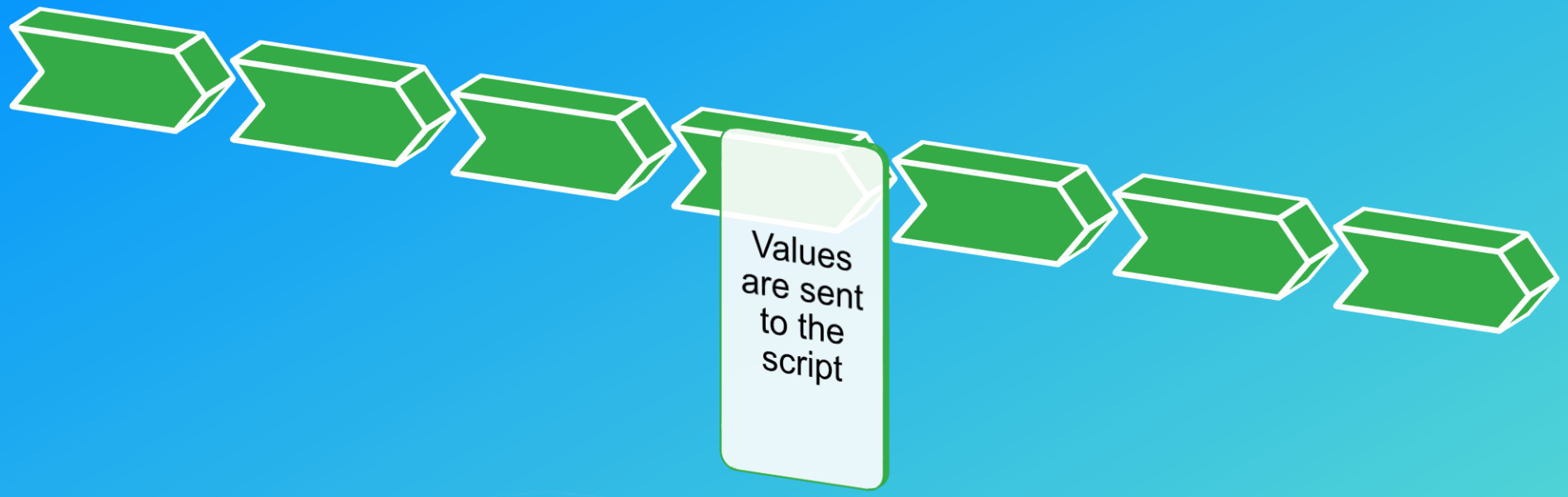
- Internal validation is checks performed by the framework, including:
  - Have all the required parameters been supplied?
  - Are the values of the appropriate data types?
  - Does the input or output exist?
  - Do values match their filter?

How a tool works



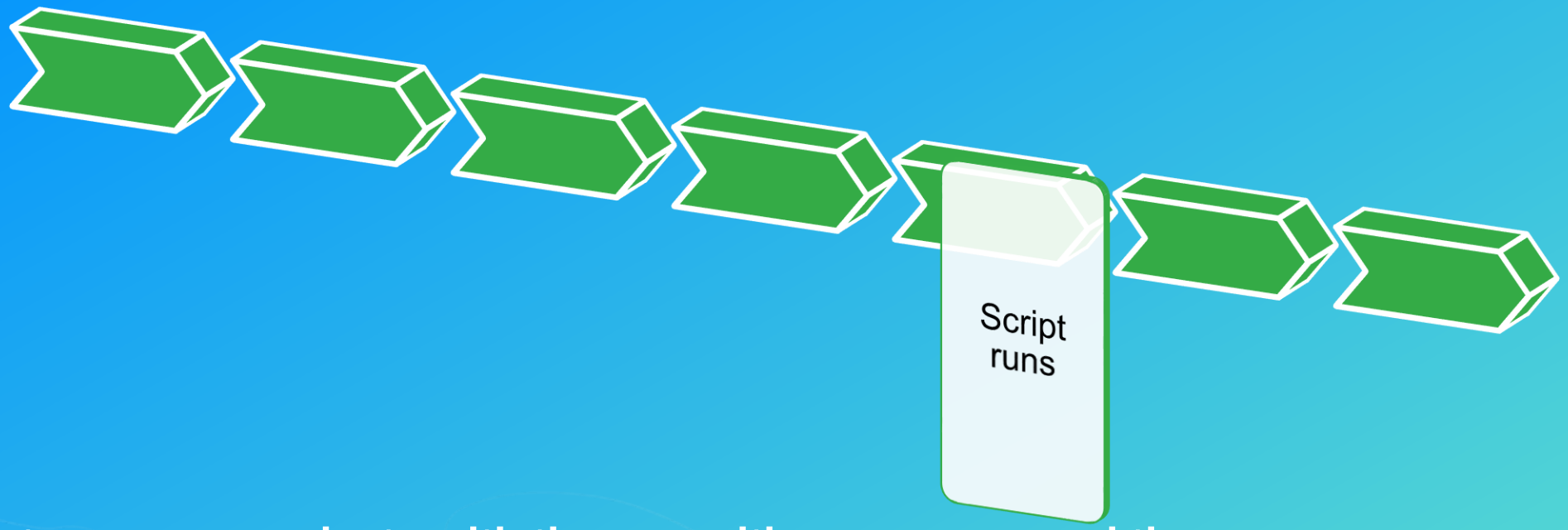
- The OK or Run buttons are pushed
  - Arguments are sent
  - .py is called

How a tool works



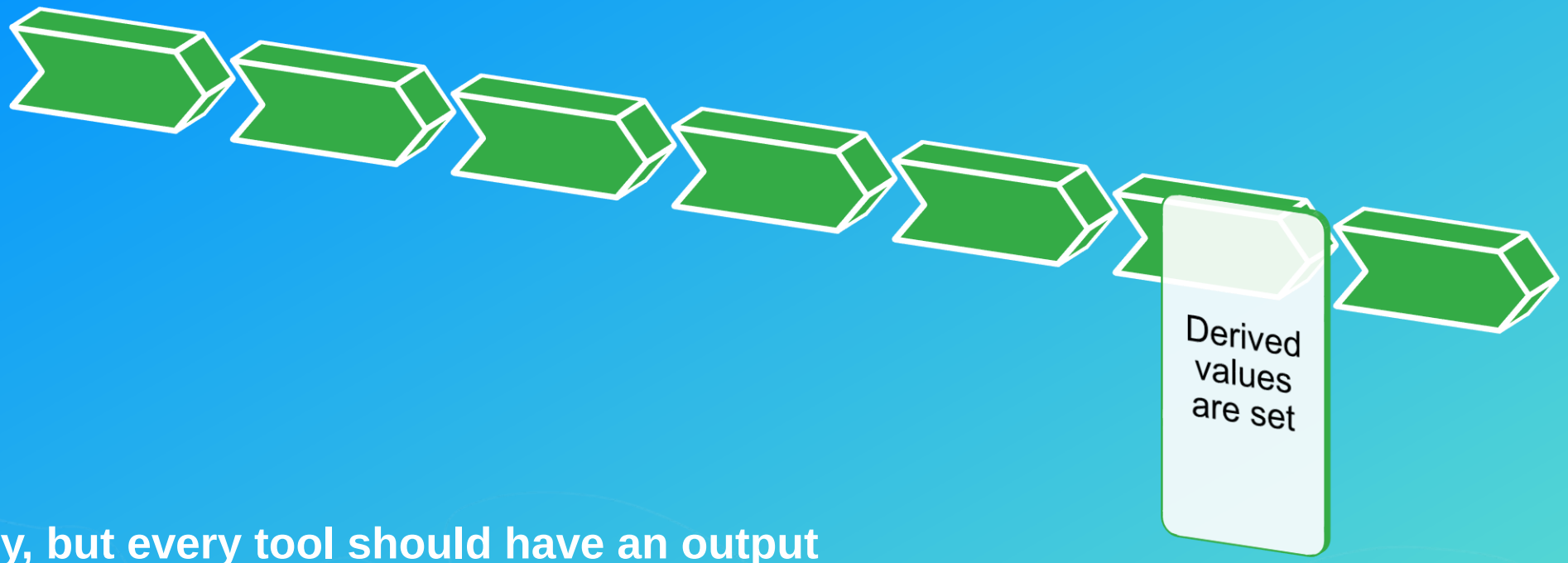
- Script receives arguments with GetParameter or GetParameterAsText

How a tool works



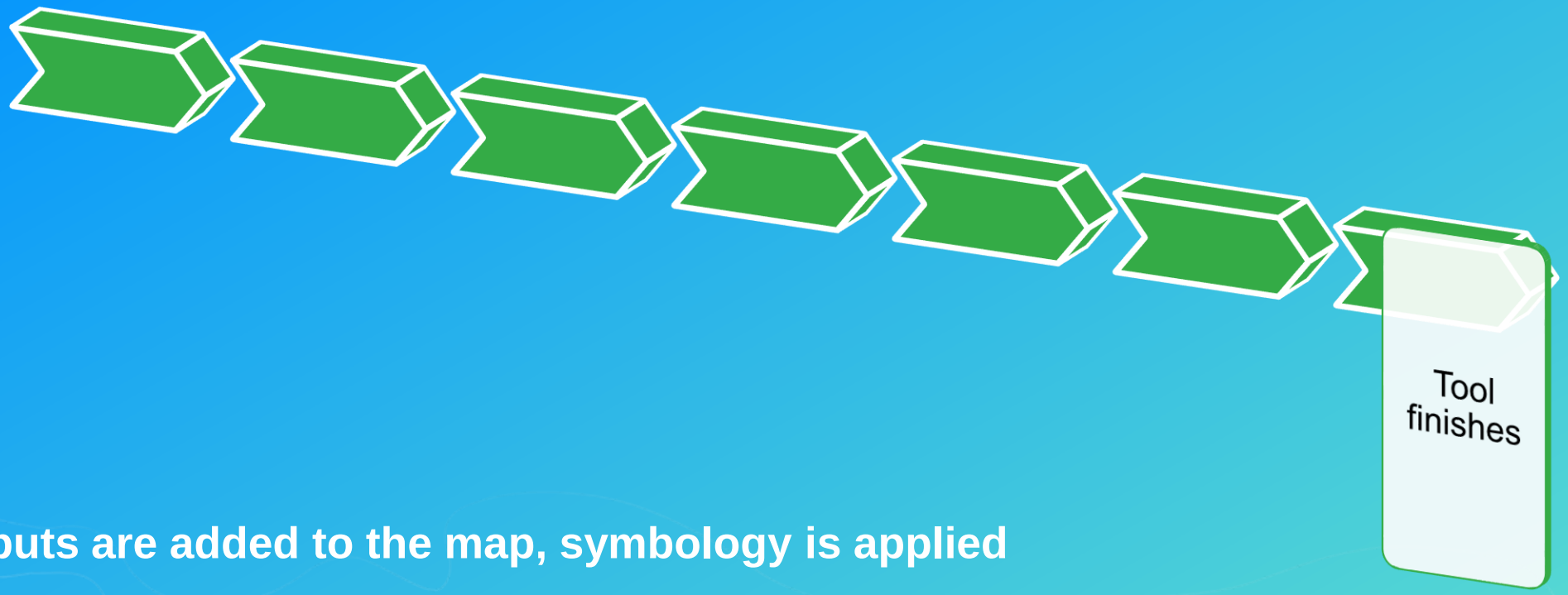
- Script can communicate with the app with messages, and the progressor
- Can respond to a cancellation

How a tool works



- If any, but every tool should have an output

How a tool works



- Outputs are added to the map, symbology is applied
- Tool result is added to the history

Demo

Andrew Ortego

Functional decomposition

A tool does one elemental operation well

Use ModelBuilder or Python to sequence tools into a workflow (compose functionality)

Geoprocessing Tool Commandments

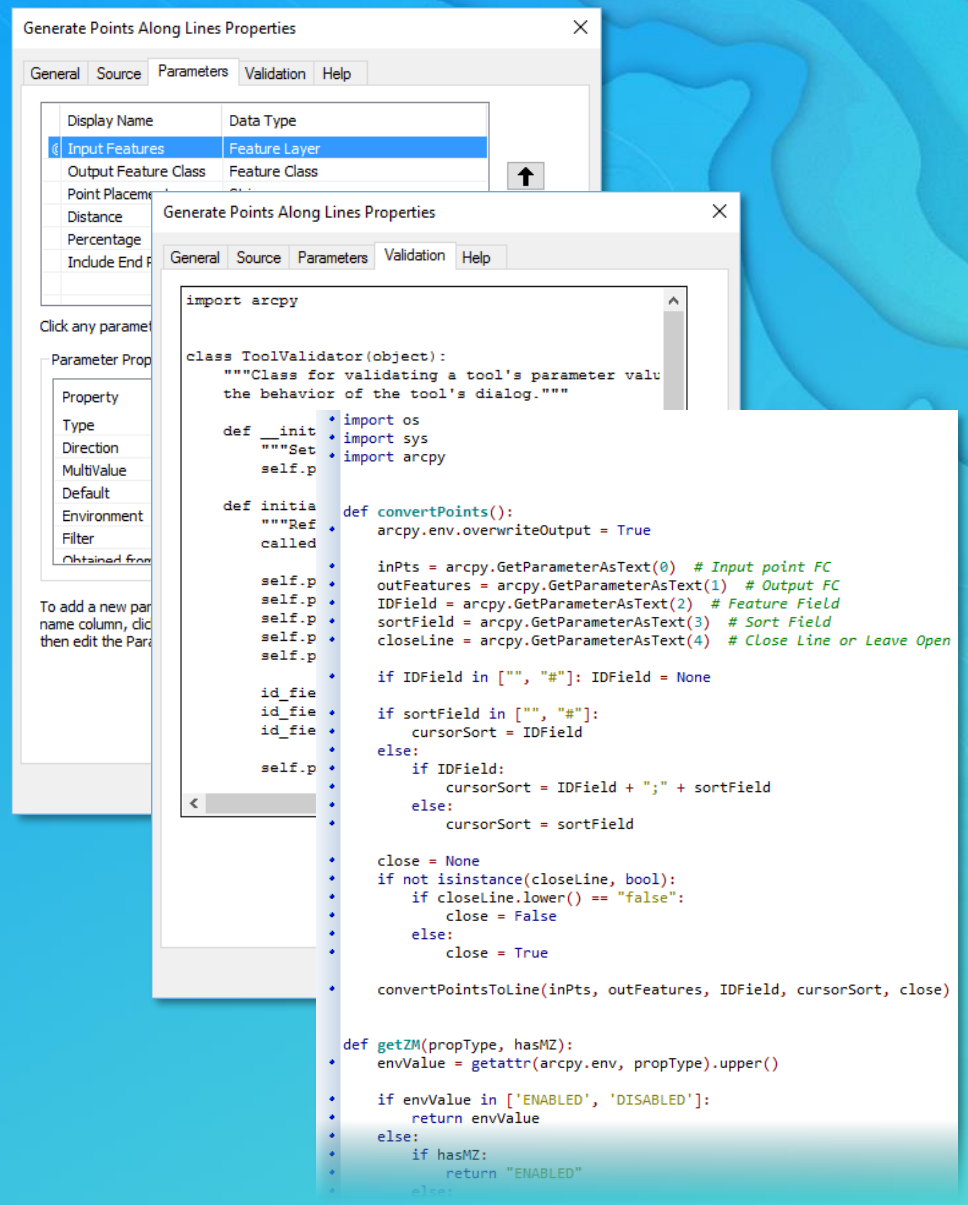
- Thou shall …
  I. … have unique parameter names within the tool
  II. … keep the cost of validation to a minimum
  III. … always have an output, even if it must be derived
  IV. … populate all output data elements within validation
  V. … not test the validity of any derived value within validation
  VI. … have a filter list for every Boolean
  VII. … test the tool from a script, a model and the dialog
  VIII. … not have the tool open with an error
  IX. … document the tool
  X. … give the toolbox an alias

Script tools vs Python toolboxes

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

Script tools ***(since 9.0)***

- 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 ***(since 10.1)***

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

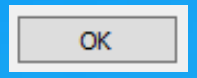
- **Which do I use?**
  - **"A tool is a tool"**

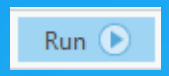
```python
import arcpy


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
```

Script tools vs Python toolboxes

	Toolboxes / Script tools		Python toolboxes
Structure	Parameters	Defined in wizard	All Python
	Validation	Python code added through UI	
	Source code	Python	
Other tools	Can include model tools		n/a
Security	Source code can be embedded in a toolbox and secured		Toolboxes are encrypted in place
Licensing	n/a		Use isLicensed method

Validation

- **Validation is everything that happens…**
- **Before the** 

or 

buttons are pushed

Purpose of validation

- **Better experience**
  - **Check validity of inputs before tool is executed**
  - **Validating relationships between parameters**
  - **Messages**
- **Describe the output of the tool *before* execution**
  - **For chaining in ModelBuilder**

Validation

- In script tools, the ToolValidator class
- In a Python toolbox tool, methods of a tool class
- Validation behaves the same in both, but is structured slightly differently
  - Script tool: parameters are properties of the class
  - Python toolbox: parameters are passed to each method

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

    if parameters[0].value:
        if not parameters[0].altered:
            extent = arcpy.Describe(parameters[0].value).extent
            if extent.width > extent.height:
                parameters[0].value = extent.width / 100.0
            else:
                parameters[0].value = extent.height / 100.0
    return
```

Validation

- Validation is about responding to changes in:
- What, or if a parameter's value is
  - `value` / `valueAsText`
  - Properties of the data (`arcpy.Describe`)
- Has the parameter's value been altered
  - `altered`
- Has the parameter been already validated?
  - `hasBeenValidated`

```
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[0].altered:
            extent = arcpy.Describe(self.params[0].value).extent
            if extent.width > extent.height:
                self.params[2].value = extent.width / 100.0
            else:
                self.params[2].value = extent.height / 100.0
    return
```

```
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
```

updateMessages

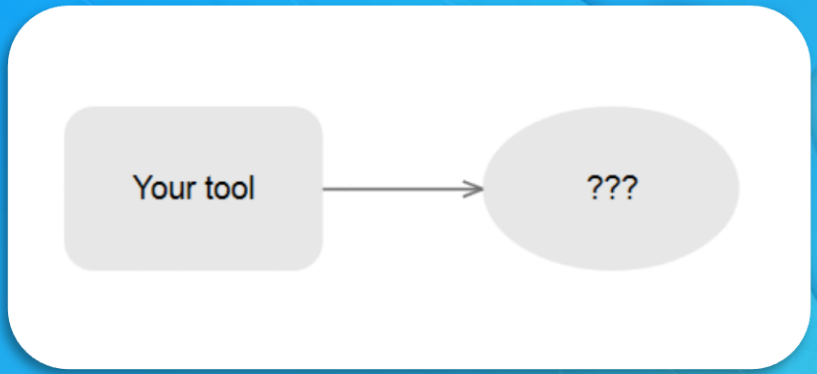
- Control over messages
  - Evaluate system messages, and relax or change
  - Add your own messages or errors based on your own criteria

```
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
```

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

    p0 = parameters[0]

    # ERROR 000800: The value is not a member of ...
    if p0.hasError() and p0.message.find('000800') > -1:
        p0.setWarningMessage(p0.message)

    return
```

Validation: ModelBuilder

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




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

id_field = arcpy.Field()
id_field.name = 'FID_1'
id_field.type = 'Integer'

parameters[1].schema.additionalFields = [id_field]
```

Demo

Andrew Ortego

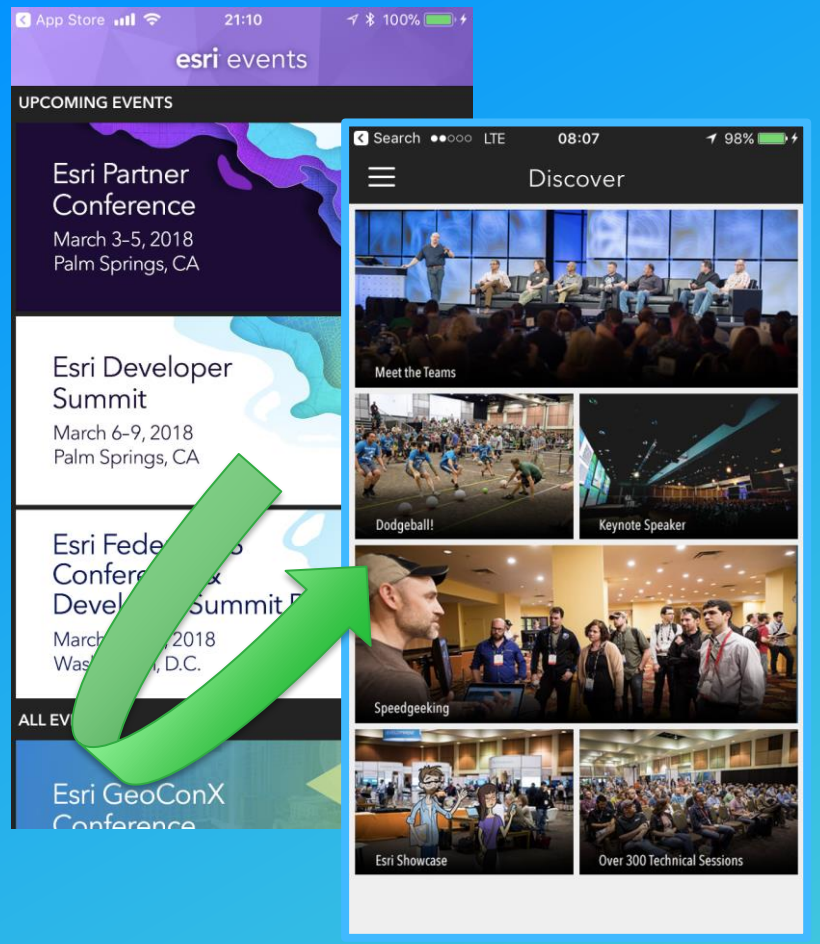
10.x to ArcGIS Pro migration

- Use the Analyze Tools For Pro tool, to identify…
  - arcpy differences
  - Python 2 to 3 compatibility issues

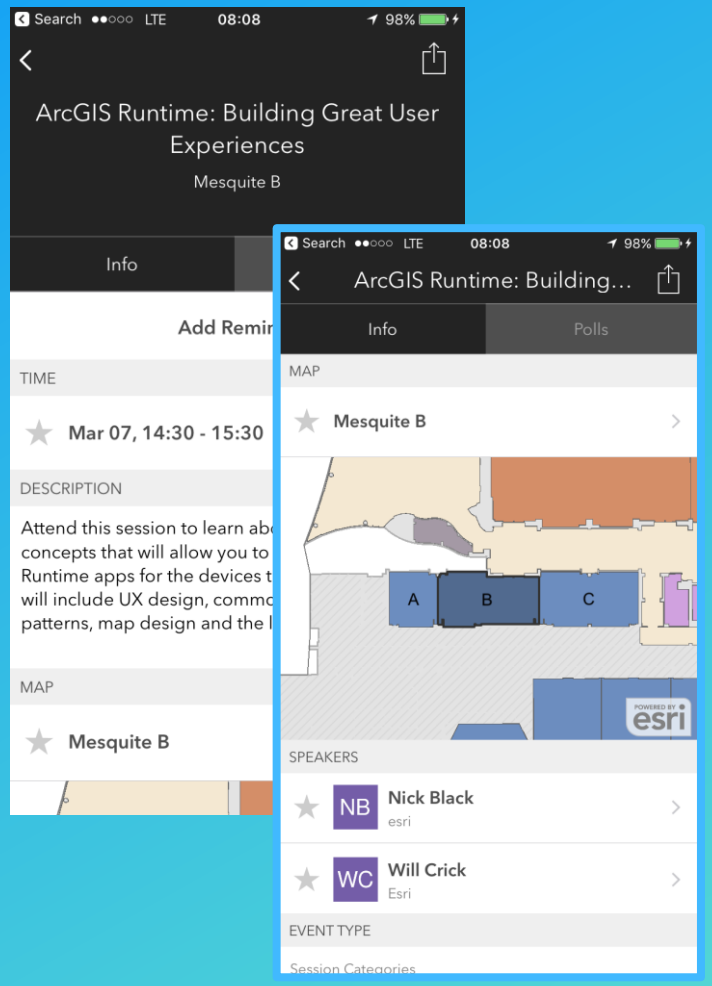
- For Python differences
  - See http://python3porting.com/strategies.html
  - Useful for writing code that will work in both Python 2 (ArcGIS) and Python 3 (Pro)

Please Take Our Survey!

Download the Esri Events app and find your event

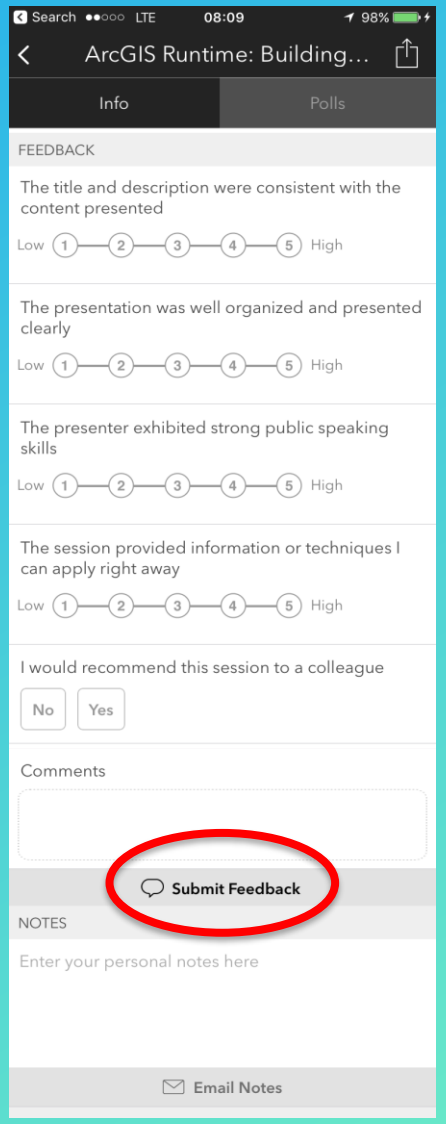


Select the session you attended



Scroll down to the "Feedback" section

Complete Answers, add a Comment, and Select "Submit"



esri

THE SCIENCE OF WHERE