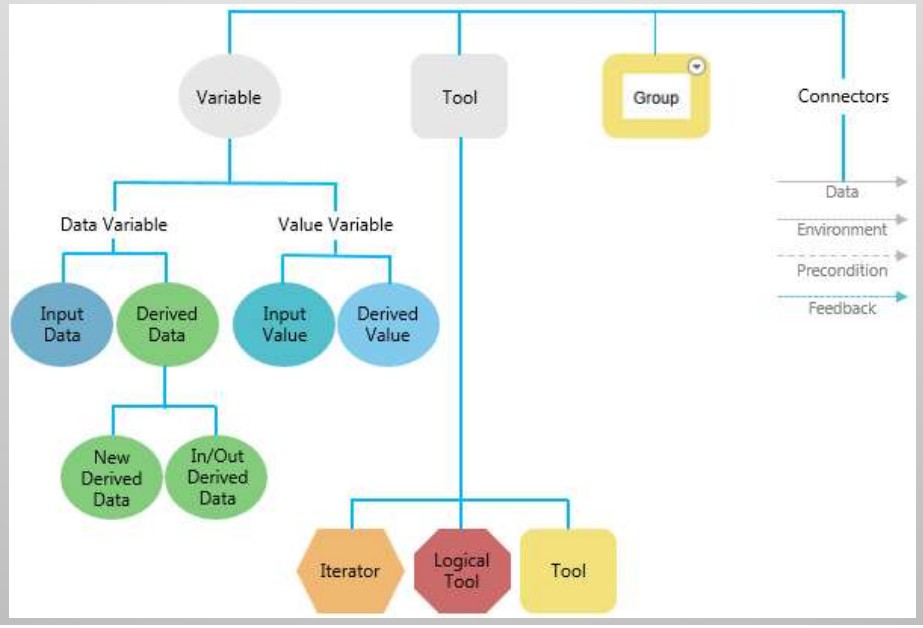
# GEOGRAPHY 9029:
## Processing GIS Data with Model Builder and Python

**Dara O'Beirne**
email: darao@mail.sfsu.edu

**10/26-10/27/2018**



# INTRODUCTIONS

- Alumni of SFSU Geography Undergrad & Grad
- MA Geography 2012
- 13 years of GIS experience
- Currently work for the City of Sacramento's GIS team
- What's your name and interest/experience in GIS?

# LOGISTICS

- Two day class from 8:30 - 5:30 Friday and Saturday
- Course structure:
    - Lecture/discussion
    - Hands on exercises
- Morning/Afternoon breaks
- 1 hour lunch

# AGENDA

**DAY 1**

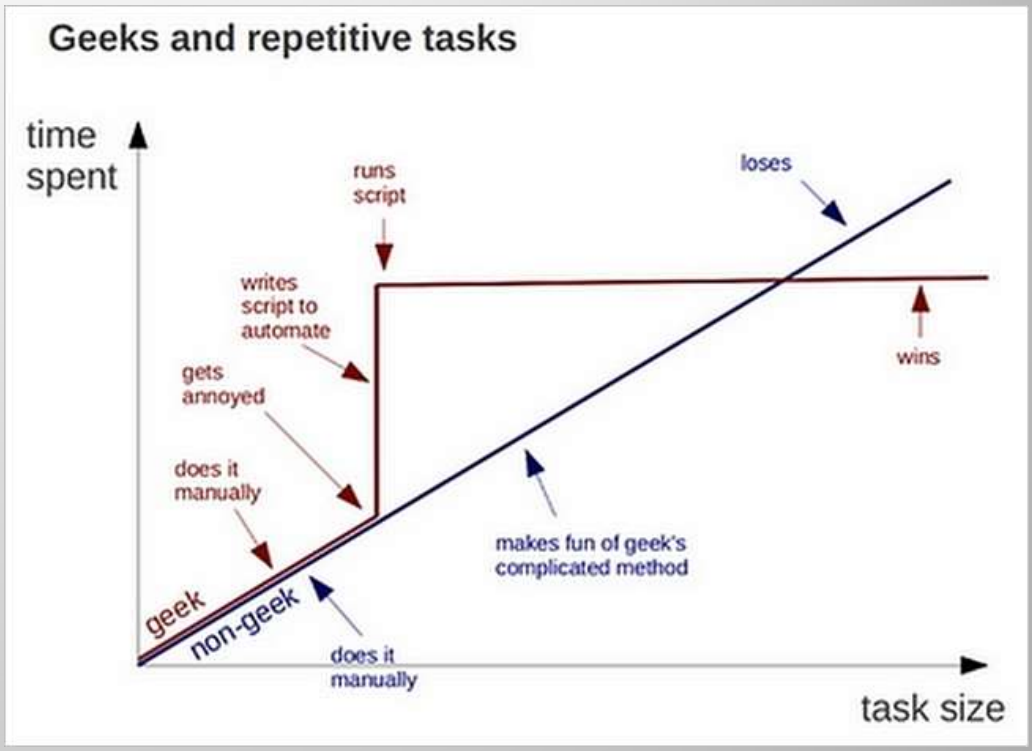
- Review of ArcGIS Pro
- Geoprocessing Framework
- Tools, Models and Scripts
- Introduction to Model Builder
- Model Parameters and Model Tools
- Advanced Functions in Model Builder

**DAY 2**

- Introduction to Python
- ArcPy and scripting in ArcGIS Pro
- Lists, conditionals and loops
- ArcPy Cursors
- Creating a script tool
- Introduction to the new ArcGIS API for Python

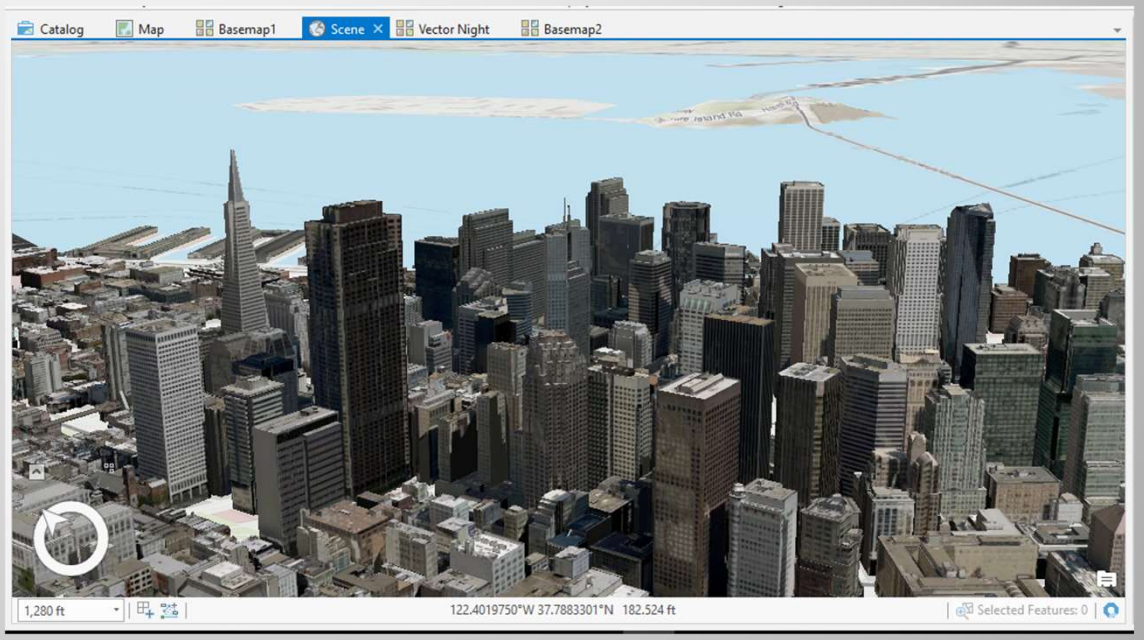
# COURSE OBJECTIVES

- Automation with GIS data and workflows
- Learn the concepts related to Geoprocessing in the ESRI platform
- Become familiar with ArcGIS Pro in the context of Model Builder and Python scripting
- Learn the basic concepts related to Python, ArcPy and ArcGIS API for Python



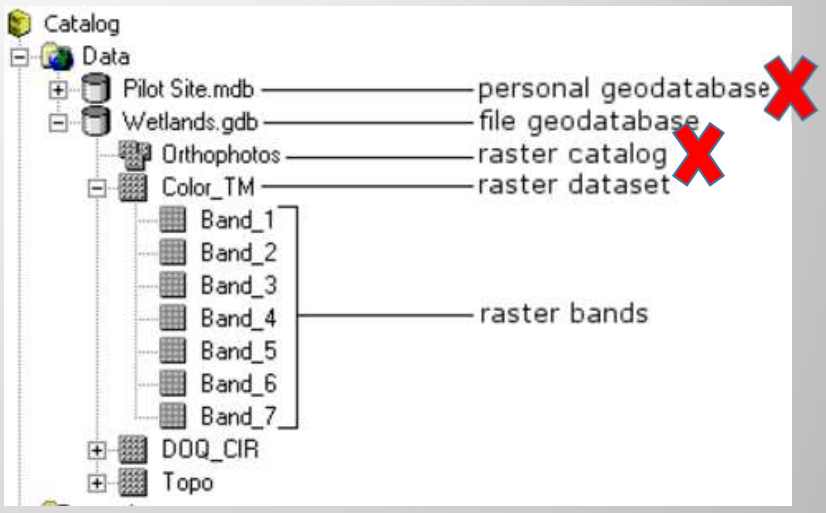
# ArcGIS PRO REVIEW

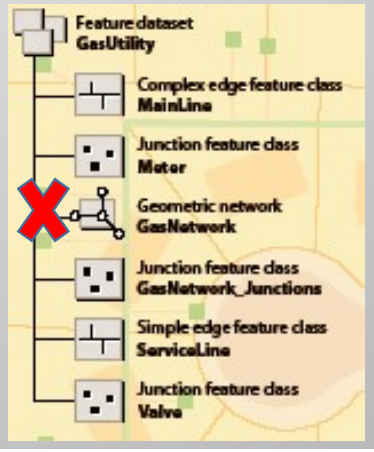
- The future of ESRI Desktop Software
- Faster (64 bit and Multi-thread)
- Python 3.x
- 2D and 3D Analysis and Visualization
- Multiple layouts (Maps and Scenes)
- The fusion of all 3 applications:
  - ArcMap
  - ArcScene
  - ArcCatalog
- Integrated with ArcGIS Online



# ArcGIS PRO REVIEW

- **NOT** supported in ArcGIS Pro:
  - Personal Geodatabase (.mdb)
  - Geometric Networks
  - Raster Catalogs
  - ArcMap Document Templates
  - ArcReader Documents
  - Tiled map packages
  - Topologies
  - Graphs (from ArcMap)
  - Batch Processing
  - Model Export to Python



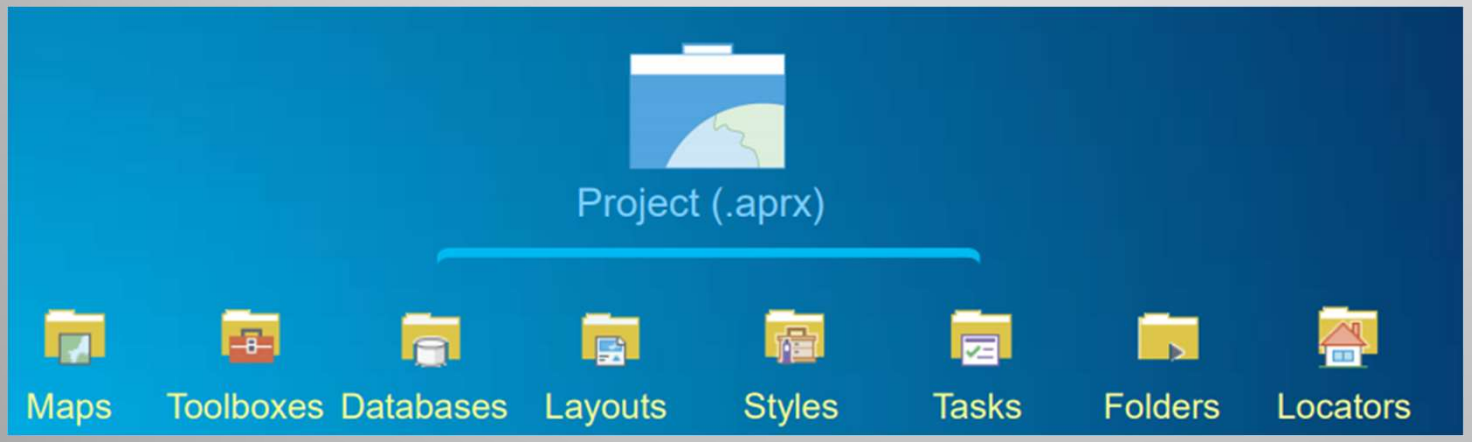



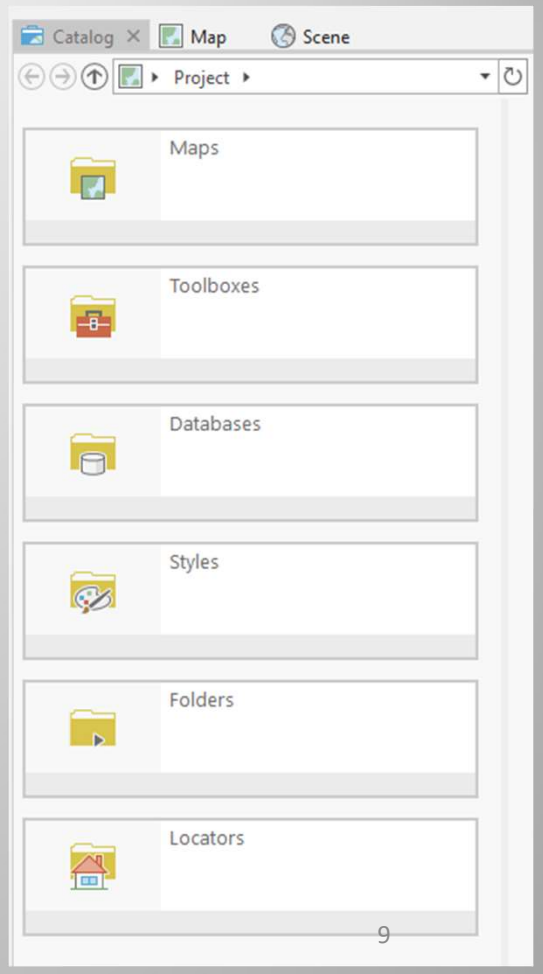
# ArcGIS PRO TERMINOLOGY

| ArcGIS Desktop Term (Old) | ArcGIS Pro Term (New) |
|---|---|
| Data Frame | View |
| Toolbar | Tab on a ribbon |
| Table of Contents Window | Contents Pane |
| Catalog Window in Desktop | Catalog Pane |
| Data View | Map View |
| Default Geodatabase | Default Map Document Geodatabase |
| Attributes and Identify Results Window | Attributes Pane |
| Find Tool | Locate Pane |
| Search Tools; Toolbox Window etc. | Geoprocessing Pane |
| Results Window | Geoprocessing History |
| Right click layer -> Data -> Export Data | Data Tab -> Export Features (copy tool) |
| Feature Service | Web Feature Layer |
| Tile Service/Cached Map Service | Web Tile Layer |

# ArcGIS PRO PROJECTS

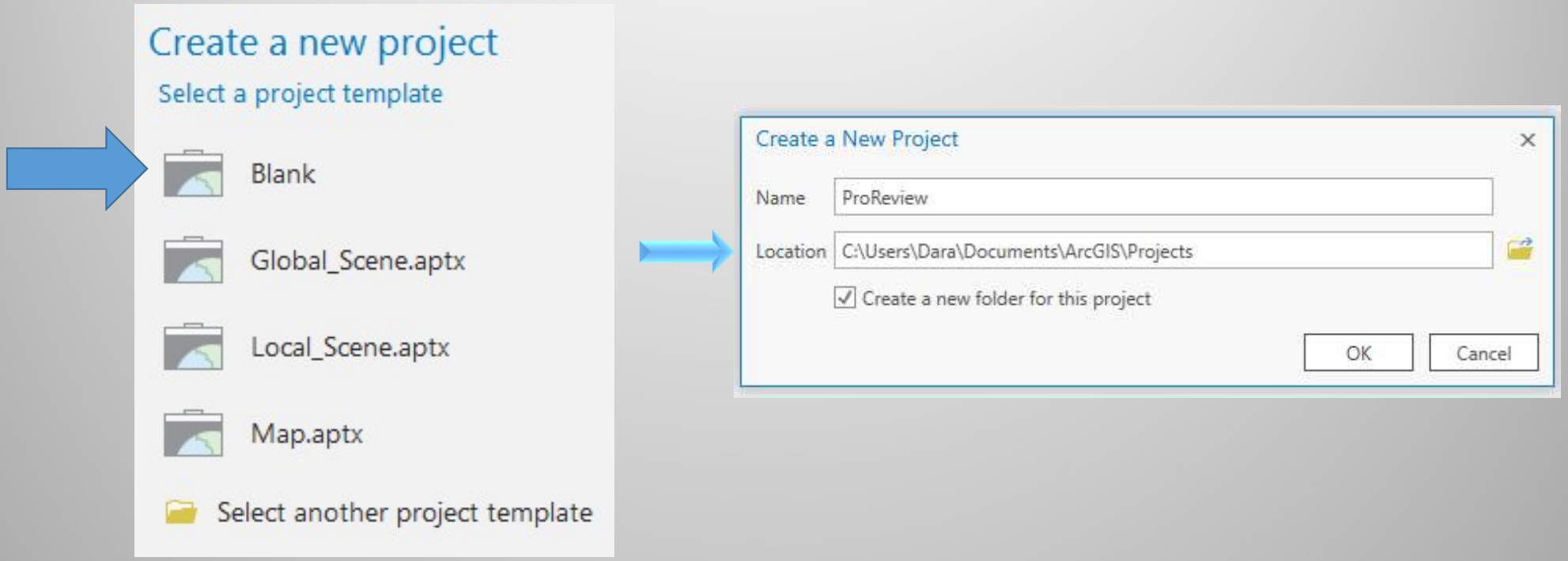
- Container of your work
- (.aprx) is file extension for Pro project
- ArcGIS Pro project is similar to an ArcMap \ ArcScene \ ArcGlobe document (.mxd, .sxd, .3dd)



# ArcGIS PRO PROJECTS

**Launch Pro and create a new project**





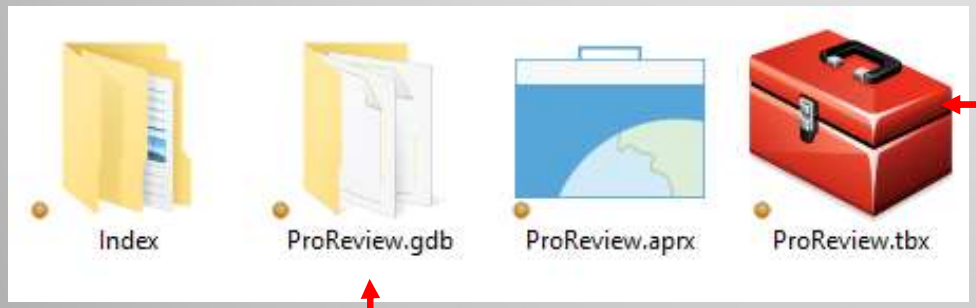

http://pro.arcgis.com/en/pro-app/help/projects/create-a-new-project.htm

# ArcGIS PRO PROJECTS

**In your new project home folder:**

**C:\Users\Dara\Documents\ArcGIS\Projects\ProReview**

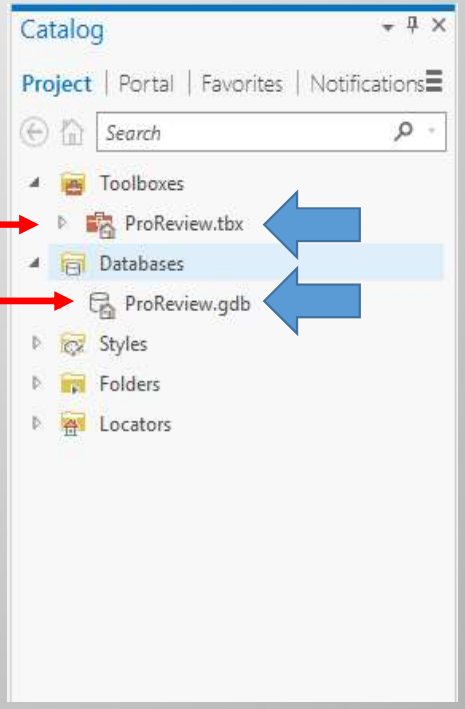
- **Index Folder**
- **Project File**
- **Project Geodatabase**
- **Project Toolbox**




**Catalog Pane:**

**-Default Toolbox**

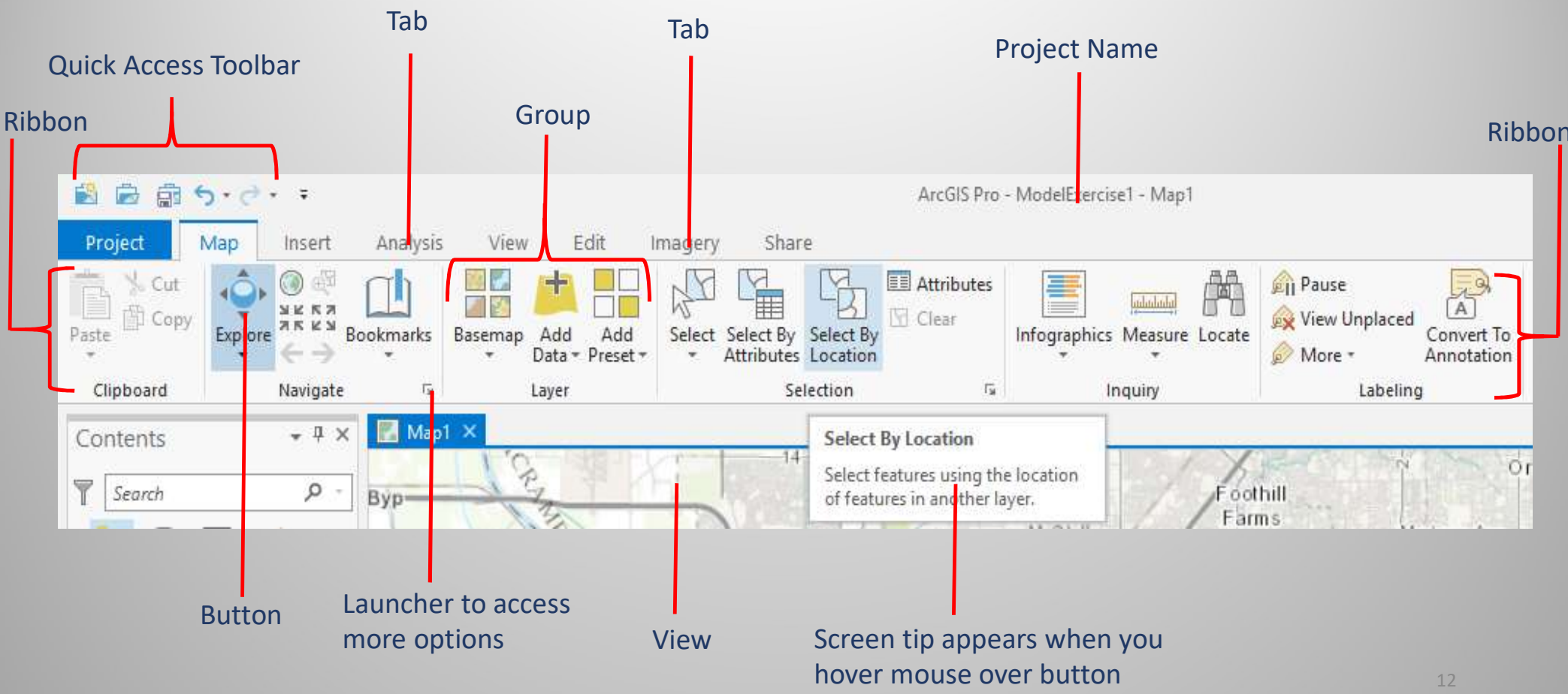
**-Default Geodatabase**






http://pro.arcgis.com/en/pro-app/help/projects/create-a-new-project.htm

# ArcGIS PRO RIBBON

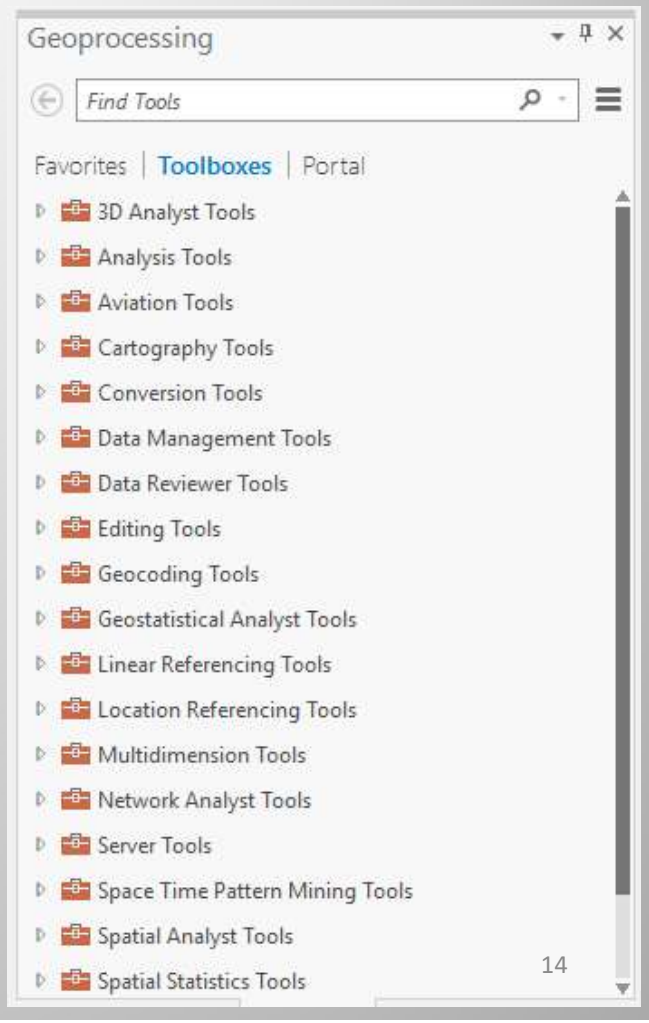


# SECTION 1

- What is geoprocessing?
- What are built-in tools, models and scripts?
- What are the advantages of using model builder and python?

# GEOPROCESSING?

- A set of tools and framework that enable you to process geographic data.
  - Data Management
  - Analysis and Models
- The “tools” in ArcGIS Pro are referred to as “geoprocessing tools”.



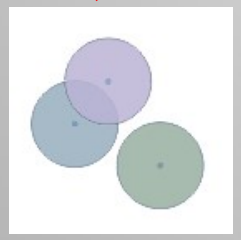
# GEOPROCESSING: INPUTS -> OUTPUTS

**Creating new data:**

*Parameter*

Input  + distance

Buffer

Output 

**Modifying data:** *Parameter*

Input  + coordinate system

Define Projection

Output 

**Synthesizing data:**

Input

Field: Add Delete Calculate | Selection: Zoom To Switch

| OBJECTID | Shape | time | latitude | longitude | depth |
|---|---|---|---|---|---|
| 1 | Point | 2014-09-21T20:08:46.... | 38.240334 | -122.34417 | 5.72 |
| 2 | Point | 2014-09-21T19:19:26.... | 36.619667 | -121.230164 | 5.59 |
| 3 | Point | 2014-09-21T19:05:45.... | 38.829166 | -122.808998 | 1.72 |
| 4 | Point | 2014-09-21T18:05:29.... | 33.0515 | -116.176333 | 1.67 |

Summary Statistics

Output

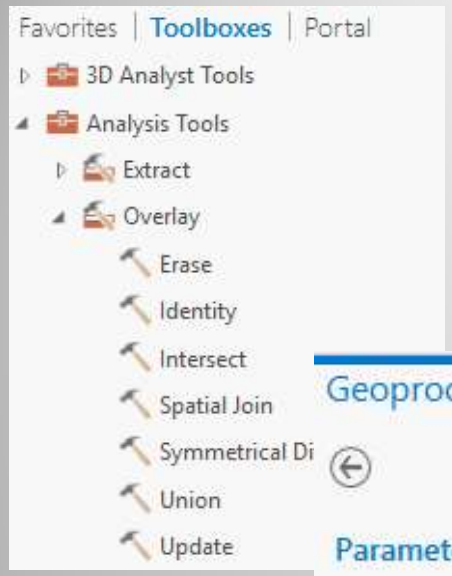
Field: Add Delete Calculate | Se

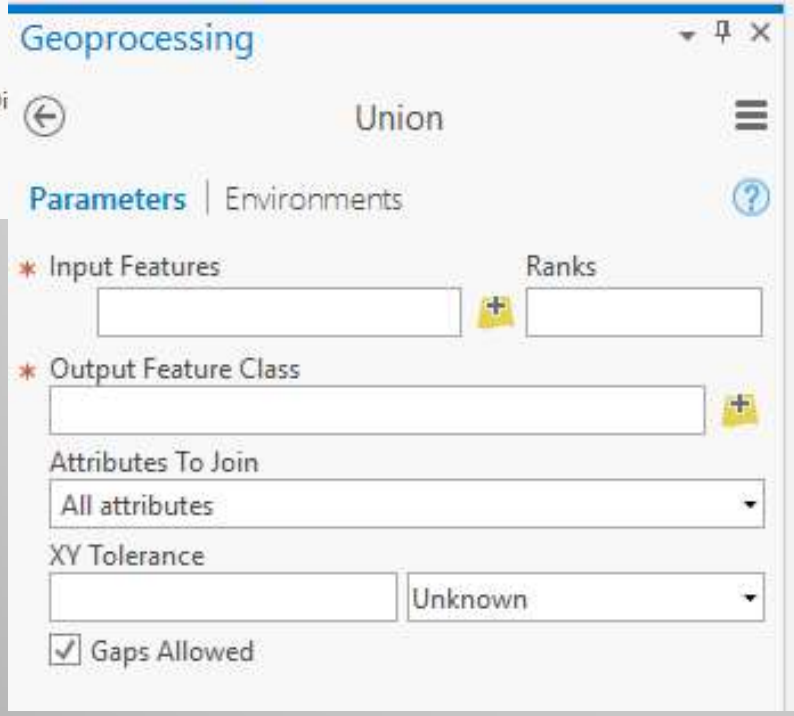
| OBJECTID | FREQUENCY | RANGE_mag |
|---|---|---|
| 1 | 484 | 3.15 |
| Click to add new row. | | |

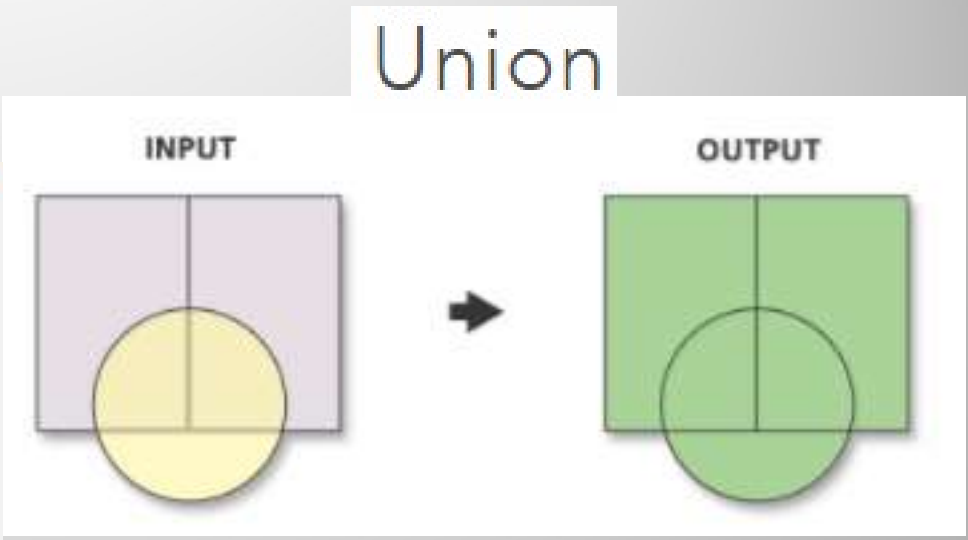
# ArcGIS GEOPROCESSING TOOLS

- Built-in Tools (Toolboxes) 
  - Installed with ArcGIS also referred to as system tools
  - Read Only (cannot edit)
    - Model tools (ModelBuilder) 
      - Custom built and stored in custom toolboxes
      - Designed like a workflow diagram
      - Are editable
        - Script tools (Python) 
          - Based entirely on custom code
          - Written in Python
          - Also editable and stored with custom toolboxes

# BUILT-IN TOOLS

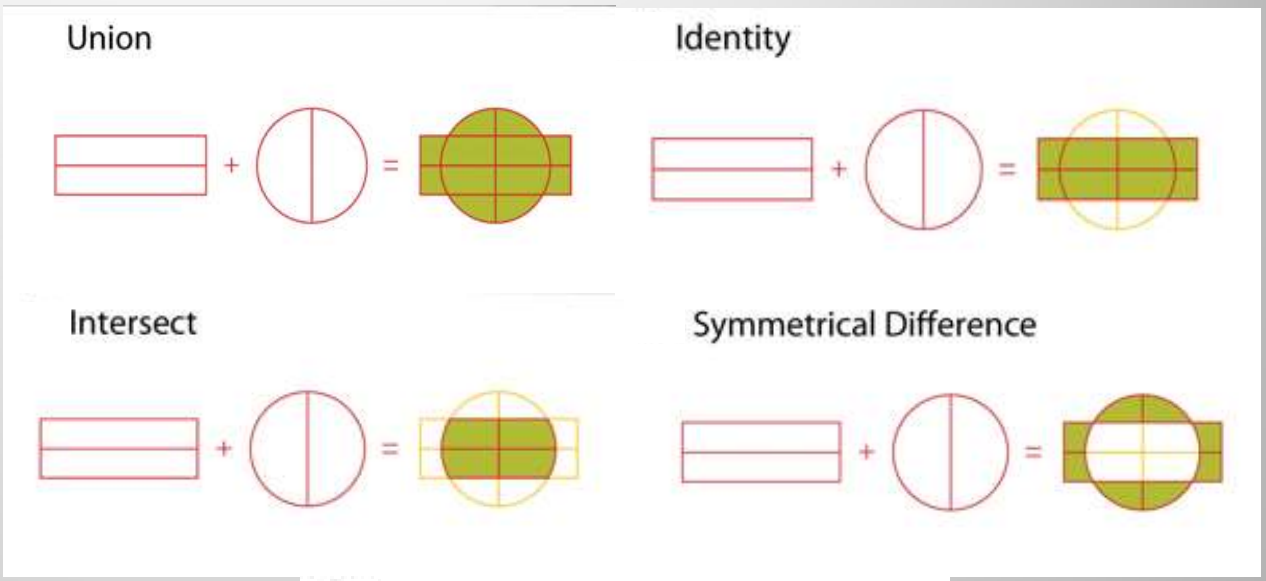


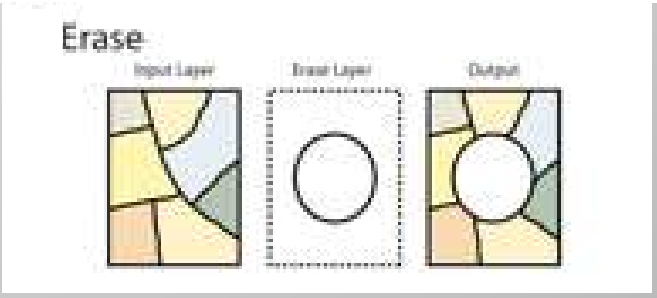



# GEOPROCESSING: OVERLAY ANALYSIS

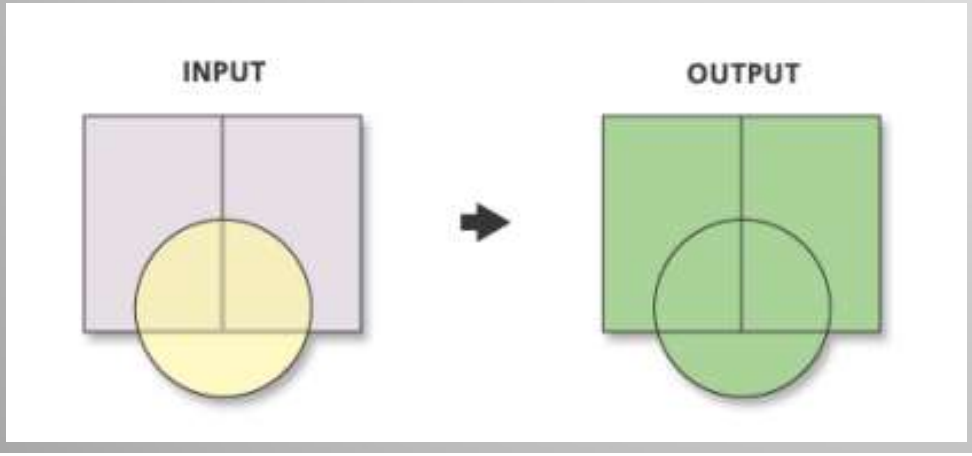
- Useful for creating data based upon the geometric/geographic relationship to other datasets.
- Extremely important to understand the difference between each of the tool's geometric dependencies.

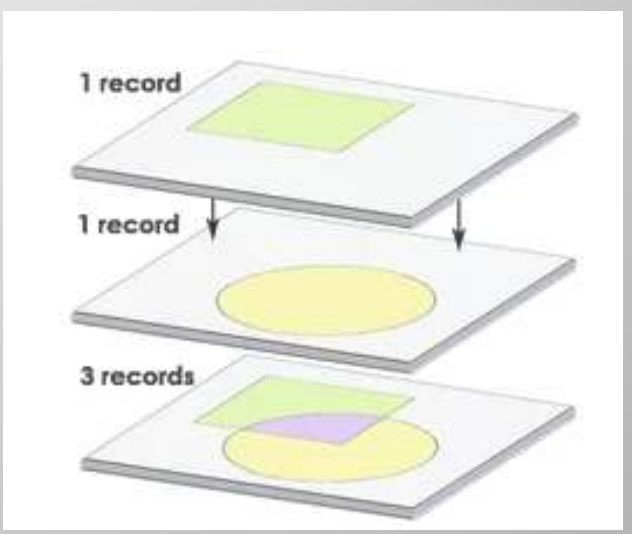


# UNION

- Overlays two polygon layers
- Resulting output layer has combined attribute data of the two inputs
- Contains all the polygons from the inputs, whether or not they overlap

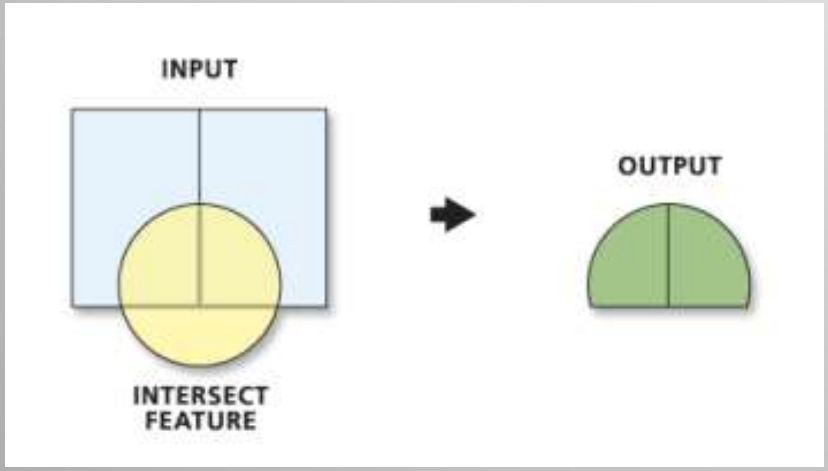


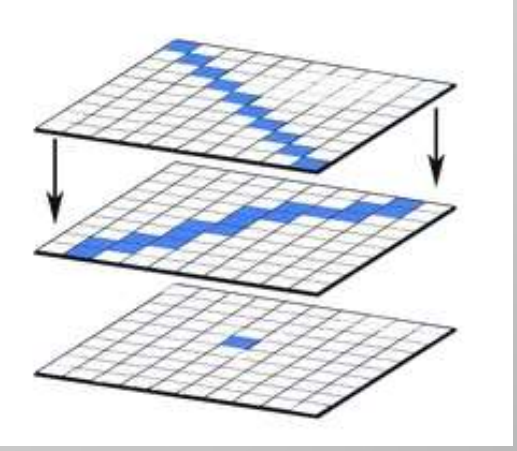




**The Union Tool maintains all input features boundaries and attributes in the output feature class.**

# INTERSECT

- Computes a geometric intersection of the input features
- Features or portions of features which overlap in all layers

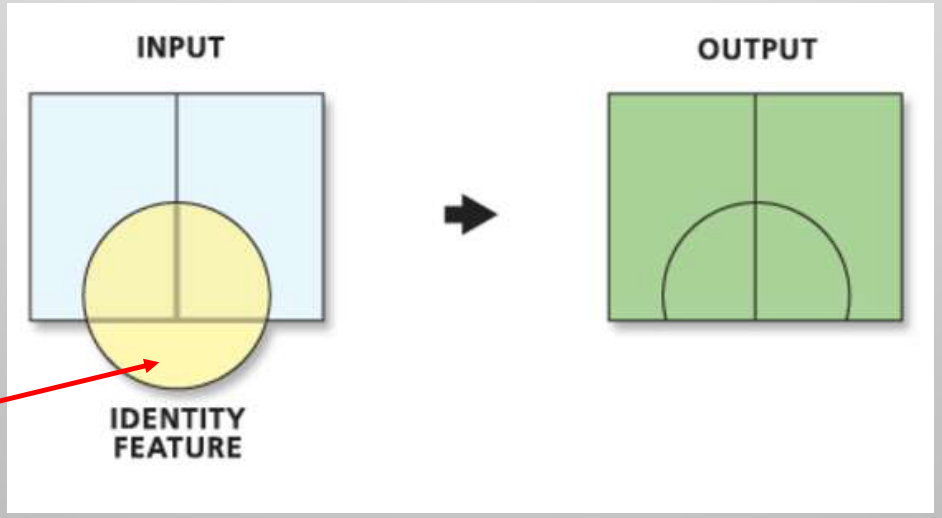


**The Intersect Tool performs a geometric overlap. All features that overlap in all layers will be part of the output feature class – attributes preserved.**

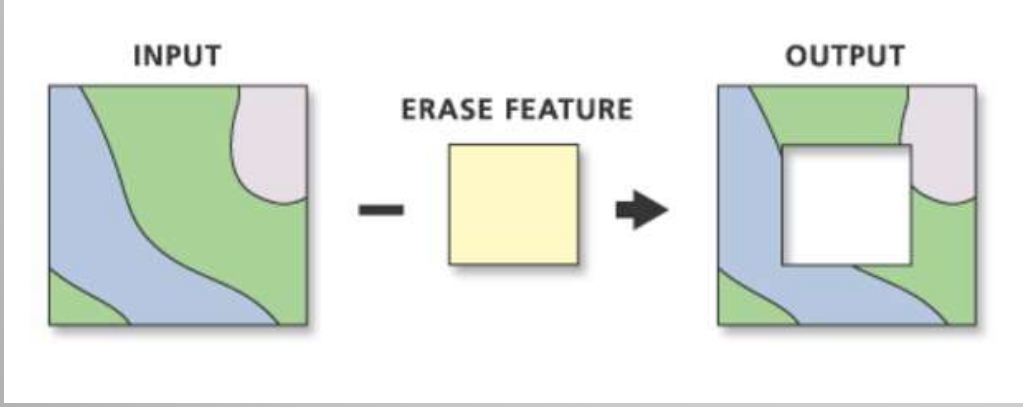
# IDENTITY

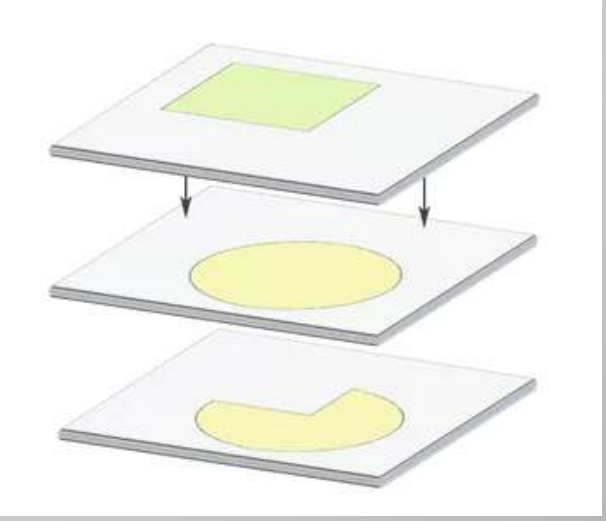
- Computes a geometric intersection of the input features and identity features. The input features or portions thereof that overlap identity features will get the attributes of those identity features



# ERASE

- Creates a feature class by overlaying the input features with the polygons of the erase features. Only those portions of the input features falling outside the erase features outside boundaries are copied to the output feature class

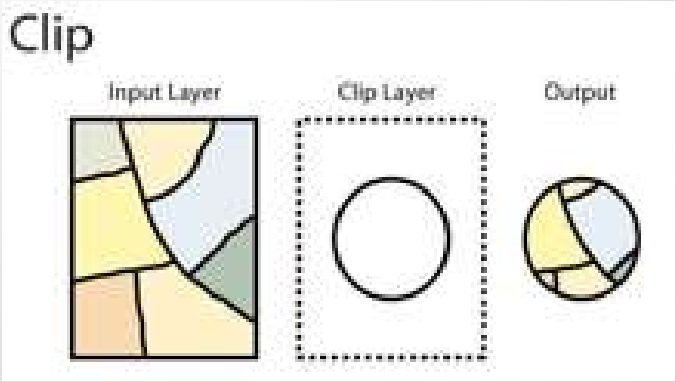


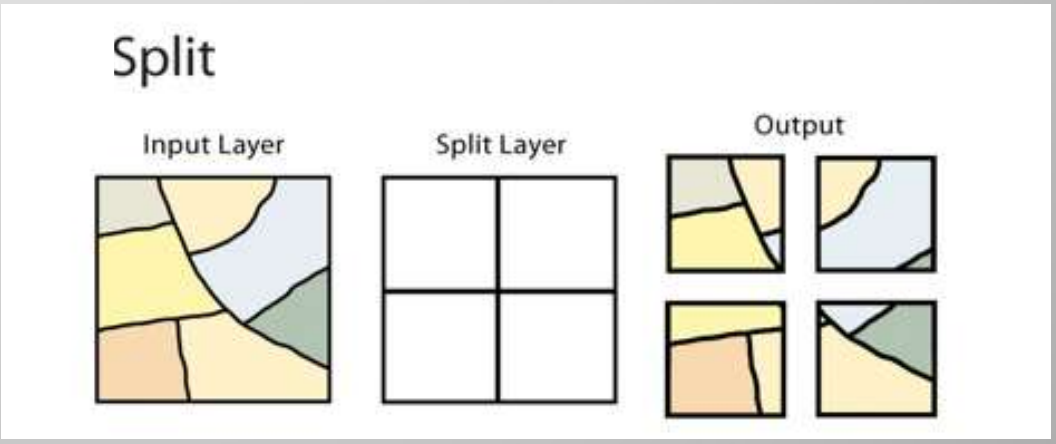
**The Erase Tool removes features that overlap the erase features. This geoprocessing tool maintains portions of input features falling outside the erase features extent.**

# GEOPROCESSING: EXTRACT

- Extracting geographic data based another features geometry

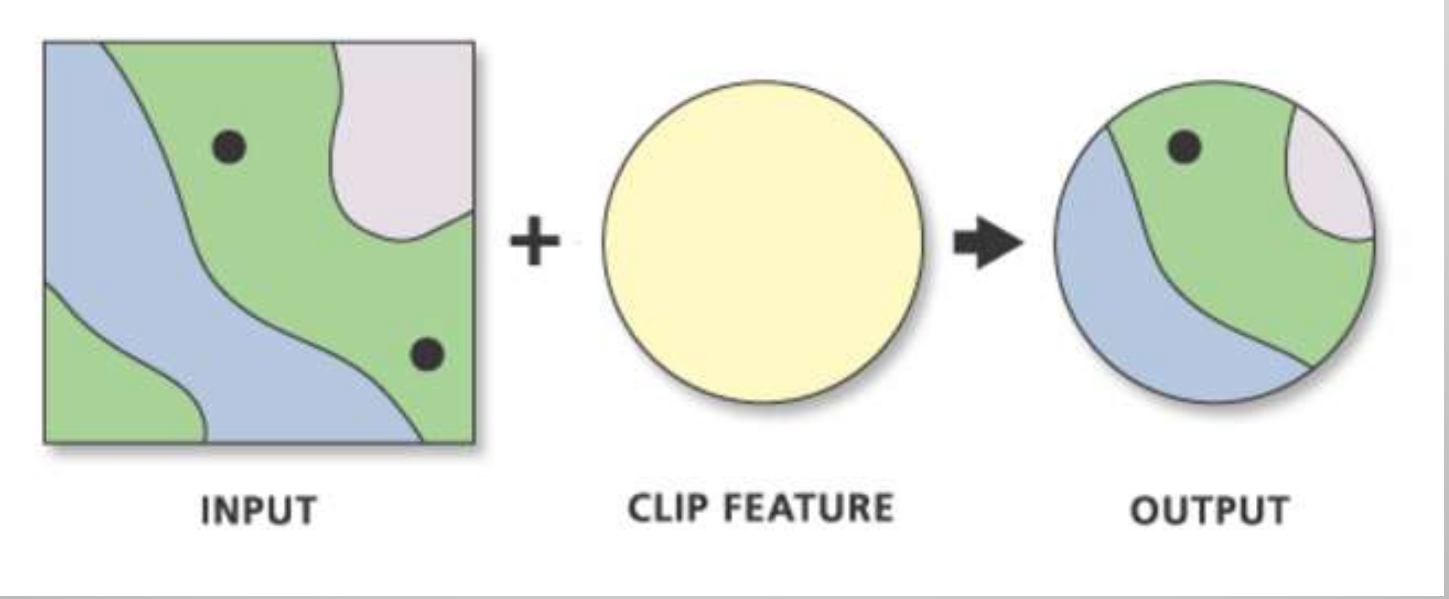



- Useful for isolating large datasets to area of interest or analysis



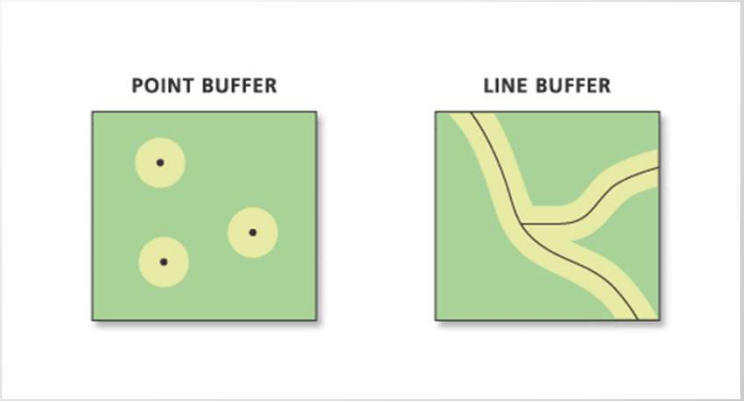
# CLIP

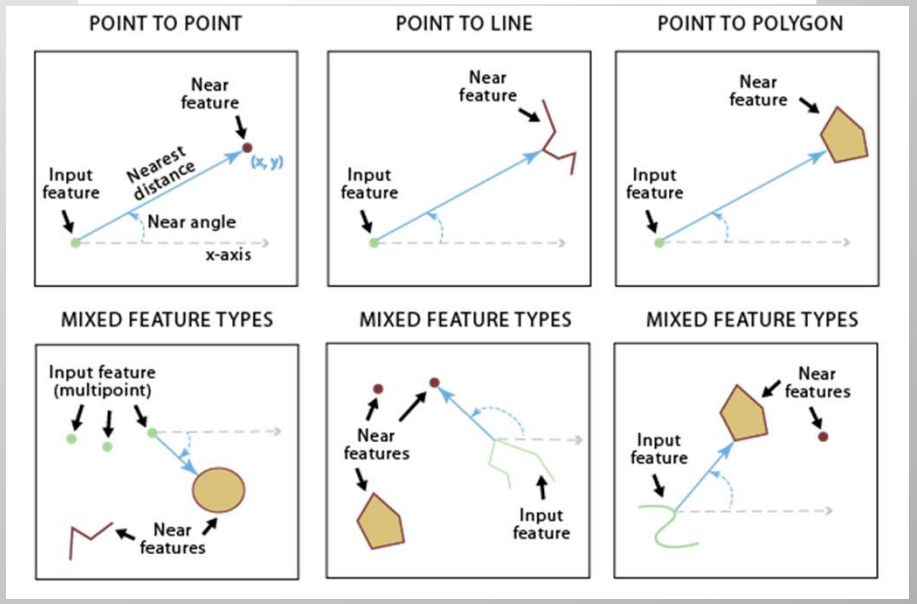
- Uses “cookie cutter” to select or isolate features



# GEOPROCESSING: PROXIMITY ANALYSIS

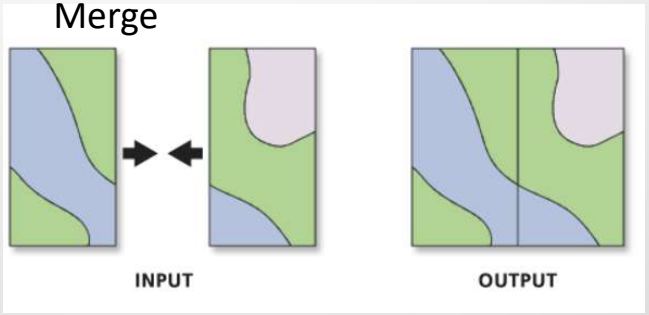
- Buffer, Near Analysis, Multi-ring Buffer, Thiessen Polygons
- Analyzing data based upon distance
- Finding the nearest object to something
- Finding all objects within a given distance

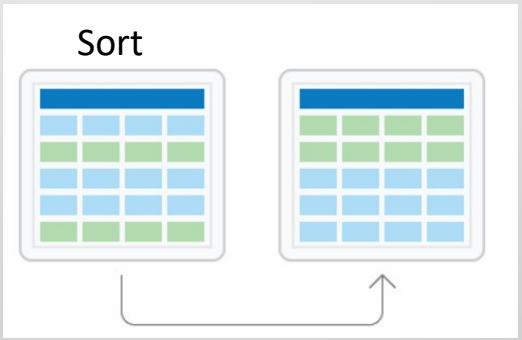


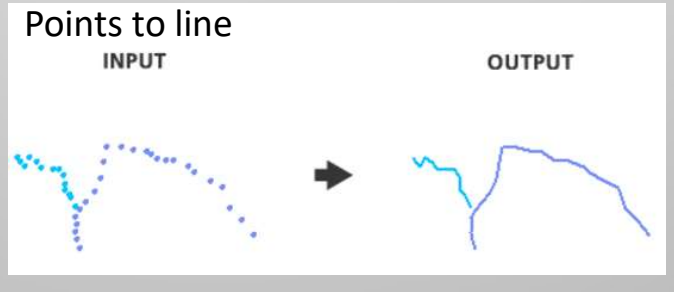



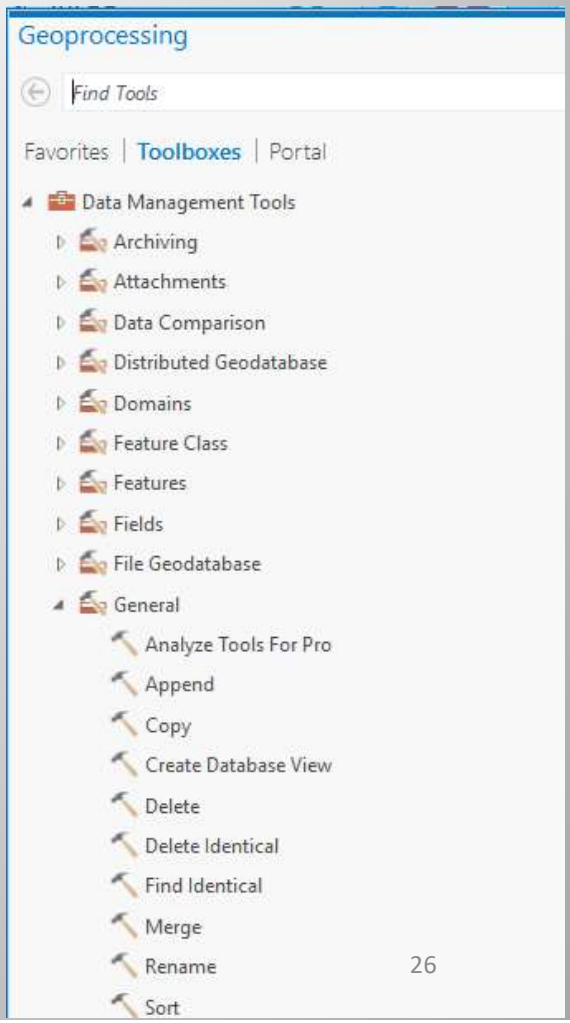
# GEOPROCESSING: DATA MANAGEMENT

- Tools that are used to develop, manage, and maintain feature classes, datasets, layers, and raster data structures.
- Some of the most commonly used geoprocessing tools by GIS professionals
- Converting between data types



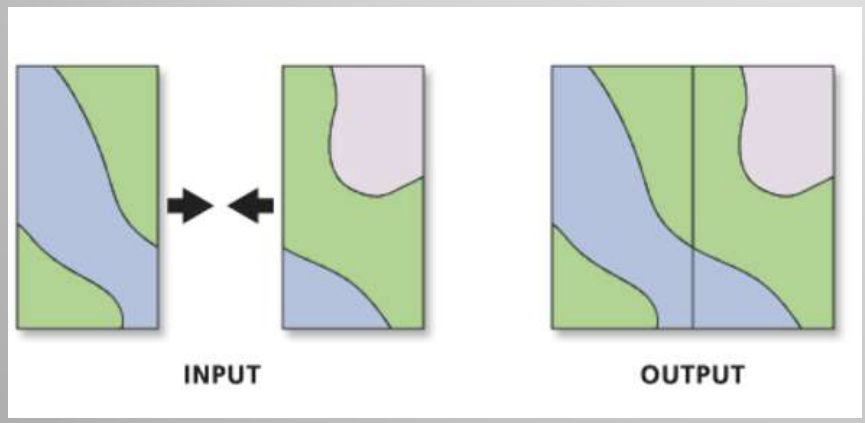



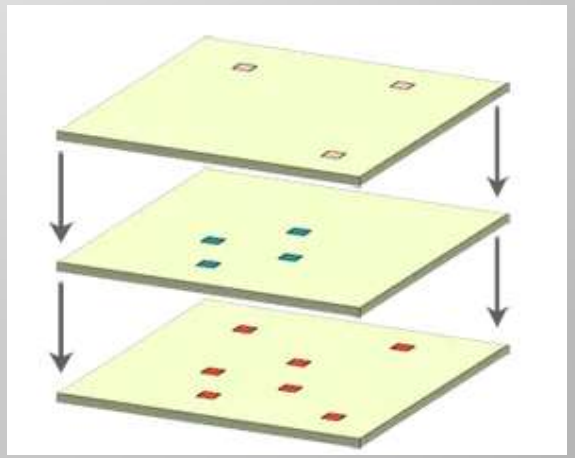



# MERGE

- What do you do when you have hundreds of data sets, and you want them in a single data set?
- You run the  Merge Tool .
- The merge geoprocessing tool combines data sets that are the same data type (points, lines or polygons). When you run the merge tool, the resulting data will be merged into one.



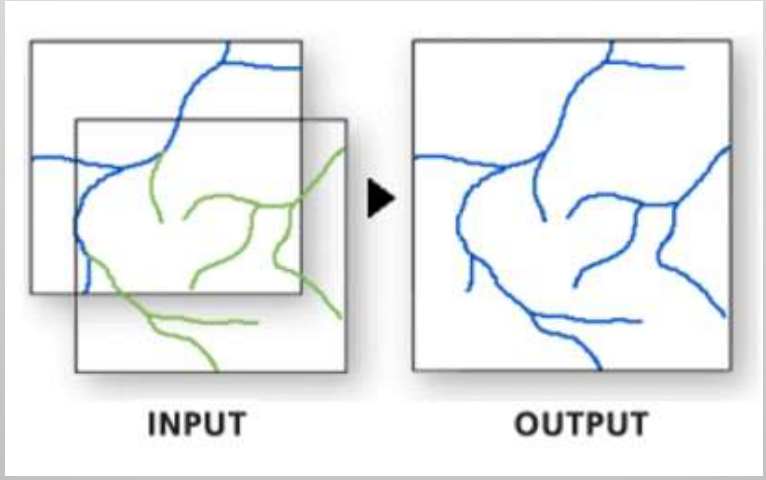



**The Merge Tool combines input features from multiple input sources. It creates a single, new, output feature class.**

# APPEND

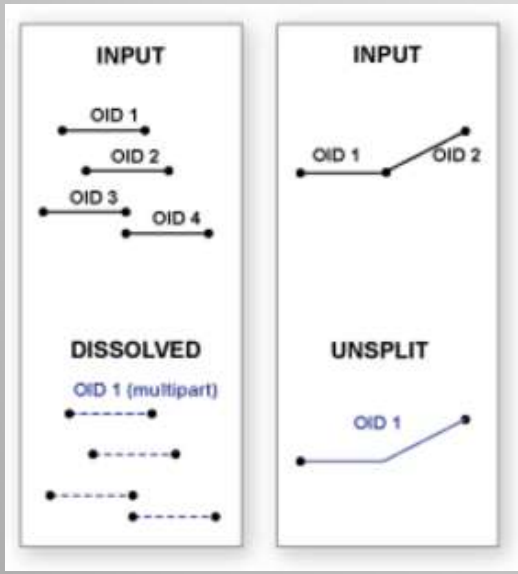
Appends one or more datasets into an existing dataset

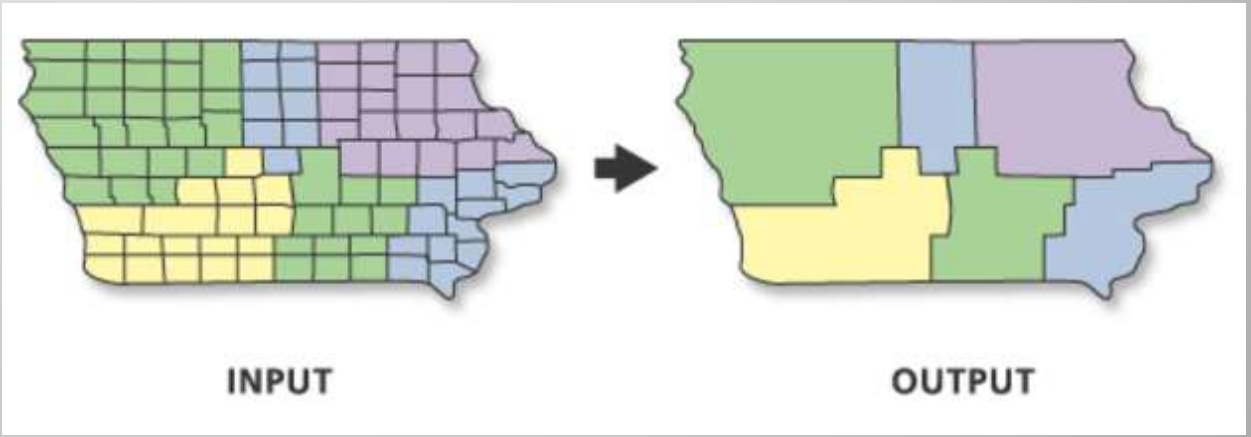
- features must be of the same feature type
- input datasets may overlap one another and/or the target dataset
- **TEST** option: fields must be the same and in the same order
- **NO TEST** option: fields do not have to match



# DISSOLVE

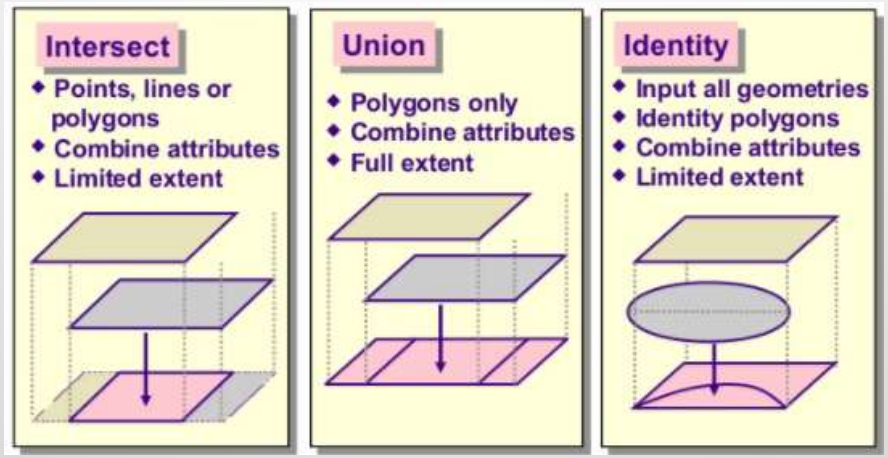
- combines adjacent polygons to create new, larger polygons
- uses common field value to remove interior lines within each polygon, forming new polygons
- can also aggregate (sum) data while dissolving



# CONFUSION AMONG THESE TOOLS

UNION vs.
INTERSECT vs.
IDENTITY



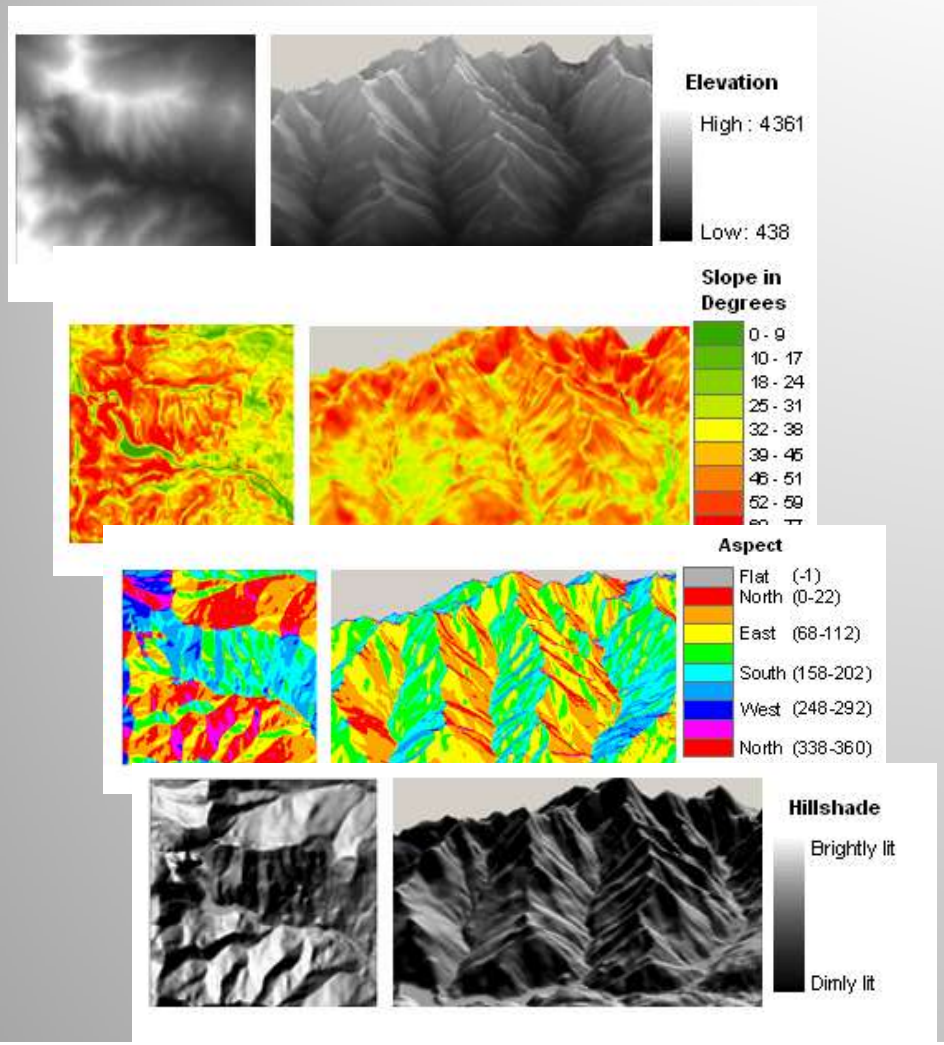

## MERGE vs. APPEND:

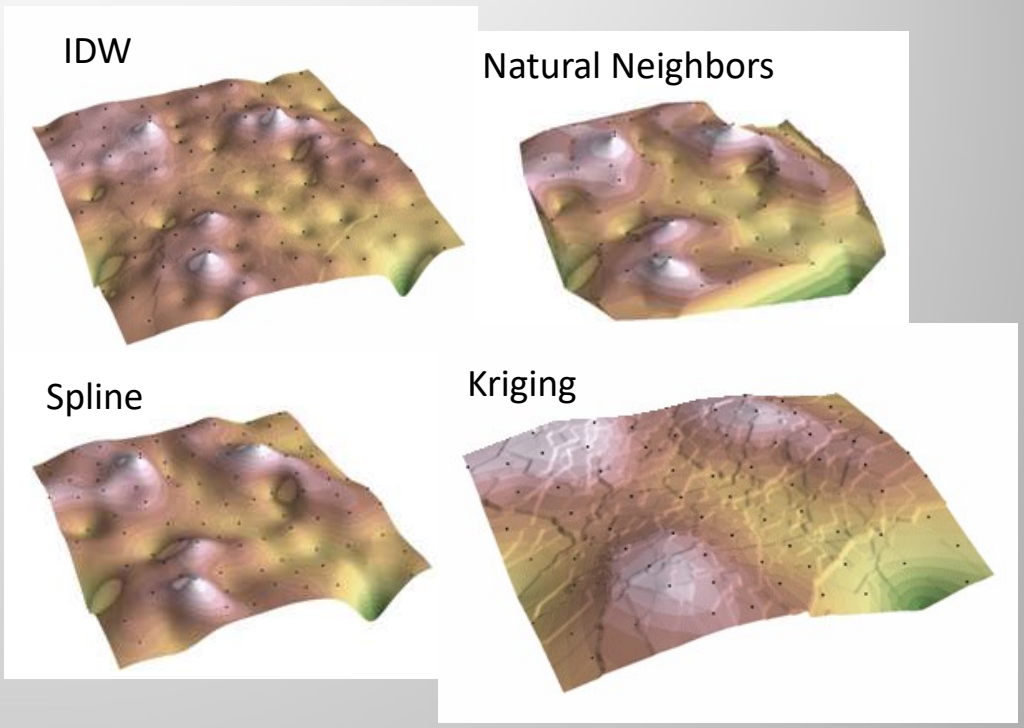
**Merge** takes both geometry and attributes and combines (merges) the entire dataset into a new feature dataset. **Merge** can be faster when dealing large amounts of data.

**Append** is good way to join extra data to an existing dataset - it can have options to control subtypes of features being appended. **Append** works best when you have domains and subtypes predefined.

**Merge** = Creates new data **Append** = takes existing data and updates it by "appending" another dataset

# GEOPROCESSING: SURFACE CREATION & ANALYSIS

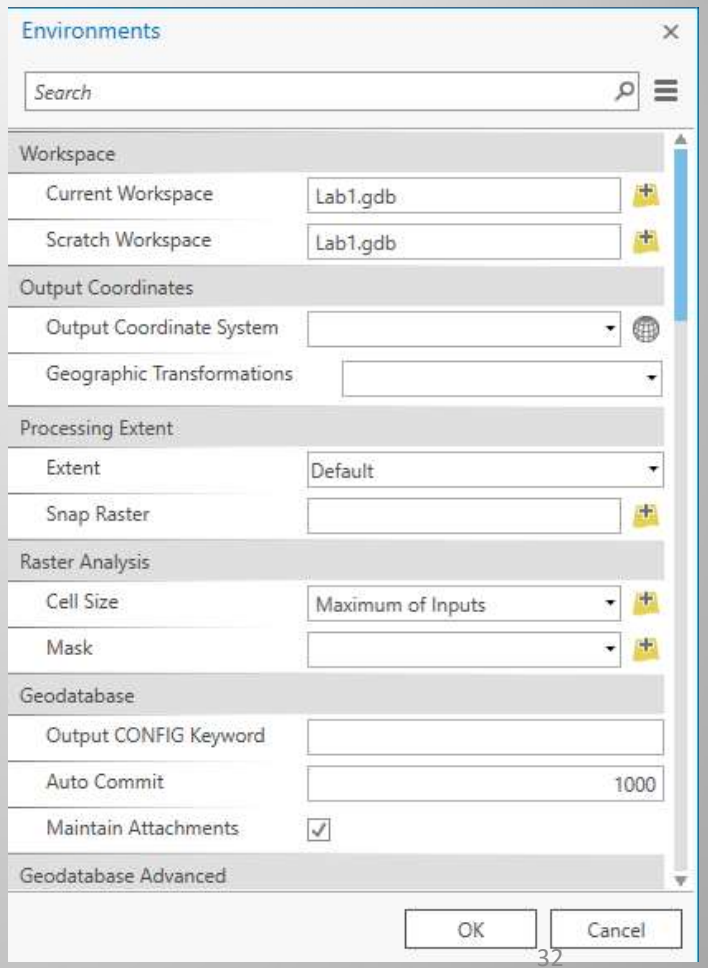


# GEOPROCESSING ENVIRONMENTS

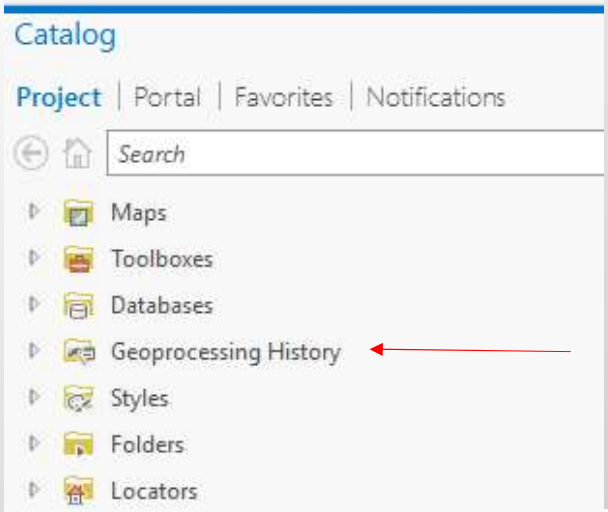
- In addition to inputs and outputs specified by a user, there are background environment settings at work
  - Extent
  - Projection
  - Cell Size
  - Workspace

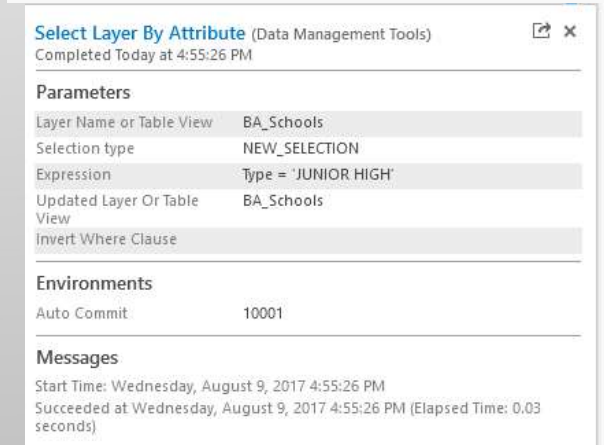
- Exist at different hierarchical levels. Multiple places to set environments:
  - Map
  - Tool
  - Model

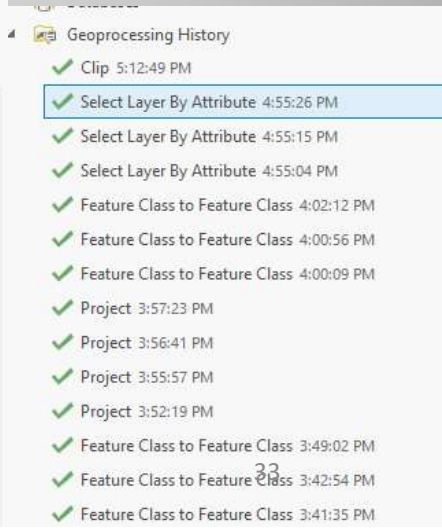



# GEOPROCESSING HISTORY

- Under Catalog Pane -> Geoprocessing History
- Extremely useful for seeing history of tools ran and error messages
- Can double click on each message and will bring up tool
- Easy to re-run tools without having to search for them

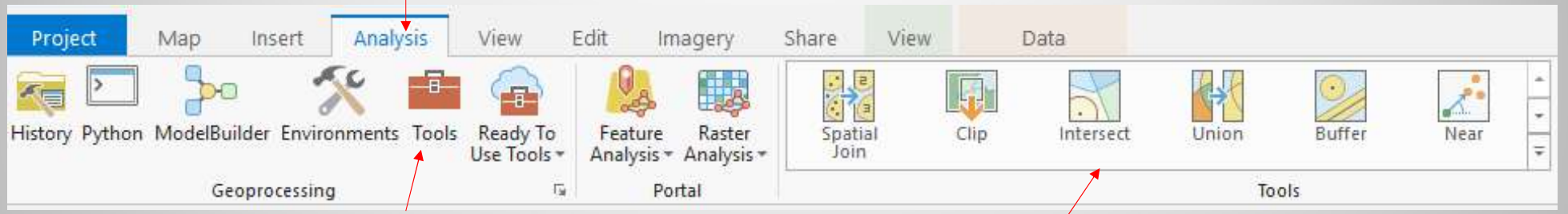


# ACCESSING TOOLS

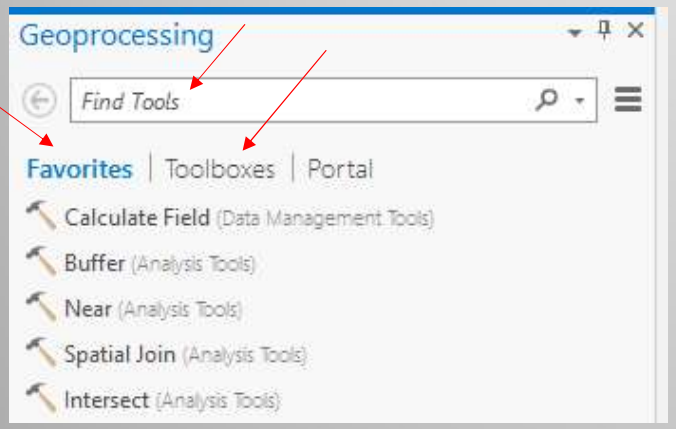
Analysis Tab



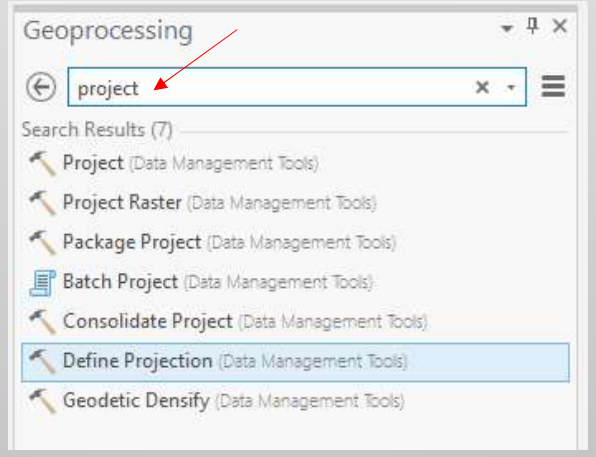

Tools Button

Commonly Used Tools Group

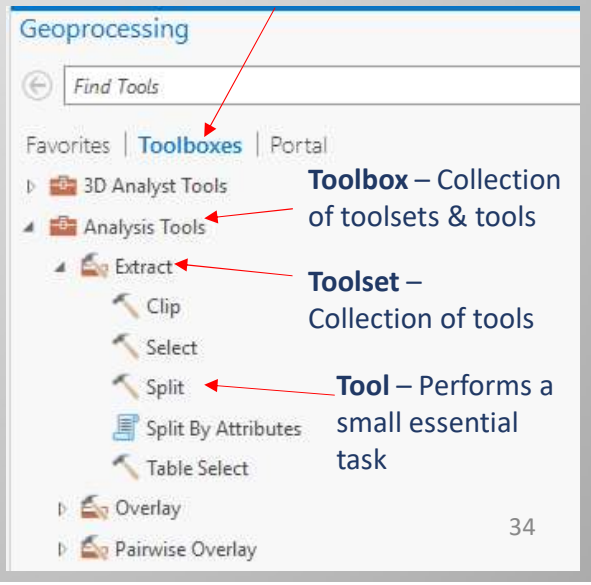
Geoprocessing Pane




Search all tools




ArcGIS Toolboxes




**Toolbox** – Collection of toolsets & tools

**Toolset** – Collection of tools

**Tool** – Performs a small essential task

# ArcGIS GEOPROCESSING TOOLS

- Built-in Tools (Toolboxes) 

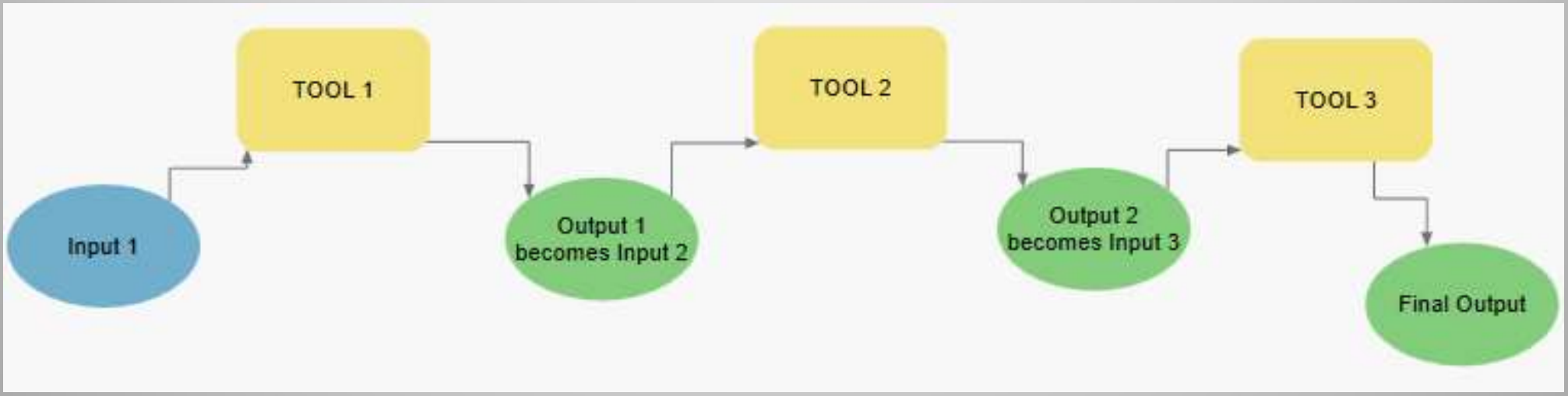
★ Model tools (ModelBuilder)  ★

★ Script tools (Python)  ★

# MODELS: INPUTS -> OUTPUTS

Outputs of one geoprocessing task are often inputs of the next geoprocessing task.

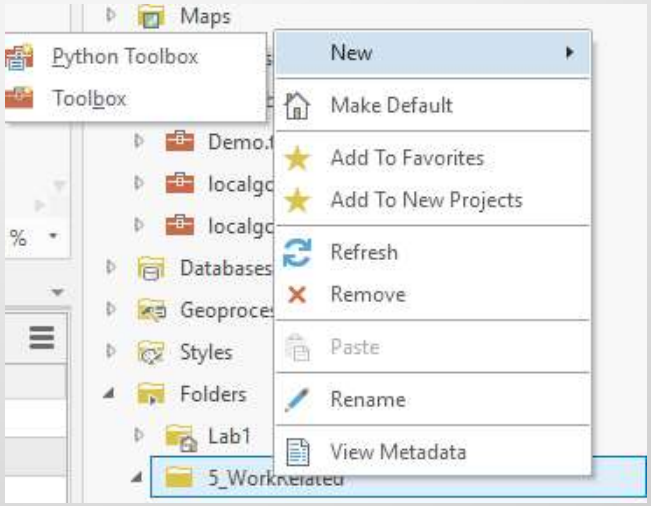
Model tools and script tools allow you to combine multiple geoprocessing tasks into one workflow



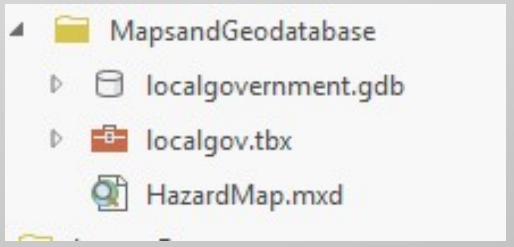
# CUSTOM TOOLBOXES

- Custom toolboxes store models and script tools
- Toolboxes can be stored in a folder or file geodatabase
  - Right click -> New -> Toolbox
- Use the Catalog Pane in ArcGIS Pro to manage

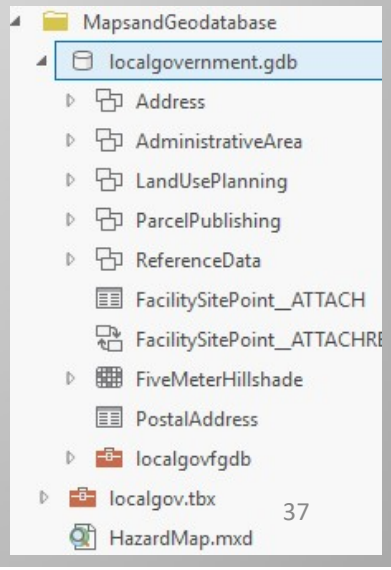
Create a new toolbox




In a Folder




In a File Geodatabase

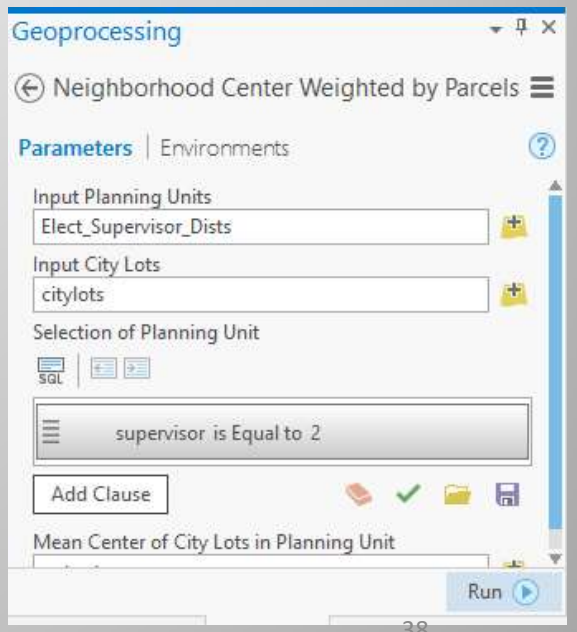


# MODEL TOOLS

- Created in Model Builder and allow you to specify a sequence of geoprocessing tasks
- Can be run like a built-in tool, or from Model Builder

Model Builder


Model Tool



# SCRIPT TOOLS

- Stored inside a custom toolbox or a Python toolbox
- Can be run just like a model tool or built-in tool
- Programming language is Python

```
#import modules
import arcpy
student_id = "Dara"

#define variables
schools_clipped = "C:/Users/" + student_id + "/Documents/ArcGIS/Projects/MB_Data/Lab1/Schools_Clip_Script.shp"
ca_schools = "C:/Users/" + student_id + "/Documents/ArcGIS/Projects/MB_Data/Lab1/Cal_Public_Schools.shp"
ba_counties = "C:/Users/" + student_id + "/Documents/ArcGIS/Projects/MB_Data/Lab1/ba_counties.shp"

#delete the file if it already exists
if arcpy.Exists(schools_clipped):
    arcpy.Delete_management(schools_clipped)

#clip the the data
#syntax is arcpy.Clip_analysis(input features, clip features, output data)
arcpy.Clip_analysis(ca_schools,ba_counties,schools_clipped)

print("Script complete")
```

# ADVANTAGES OF MODELS & SCRIPTS

1. Easy to replicate analysis/management tasks or re-run them with new inputs (reusable)
2. Provides a big picture overview of a project or workflow
3. Process runs faster once developed
4. Documents exact steps taken
5. Can be easily shared with others (including online)
6. Allows you to build customized tools
7. Makes it easier to manage intermediate data
8. Reduces potential for human errors

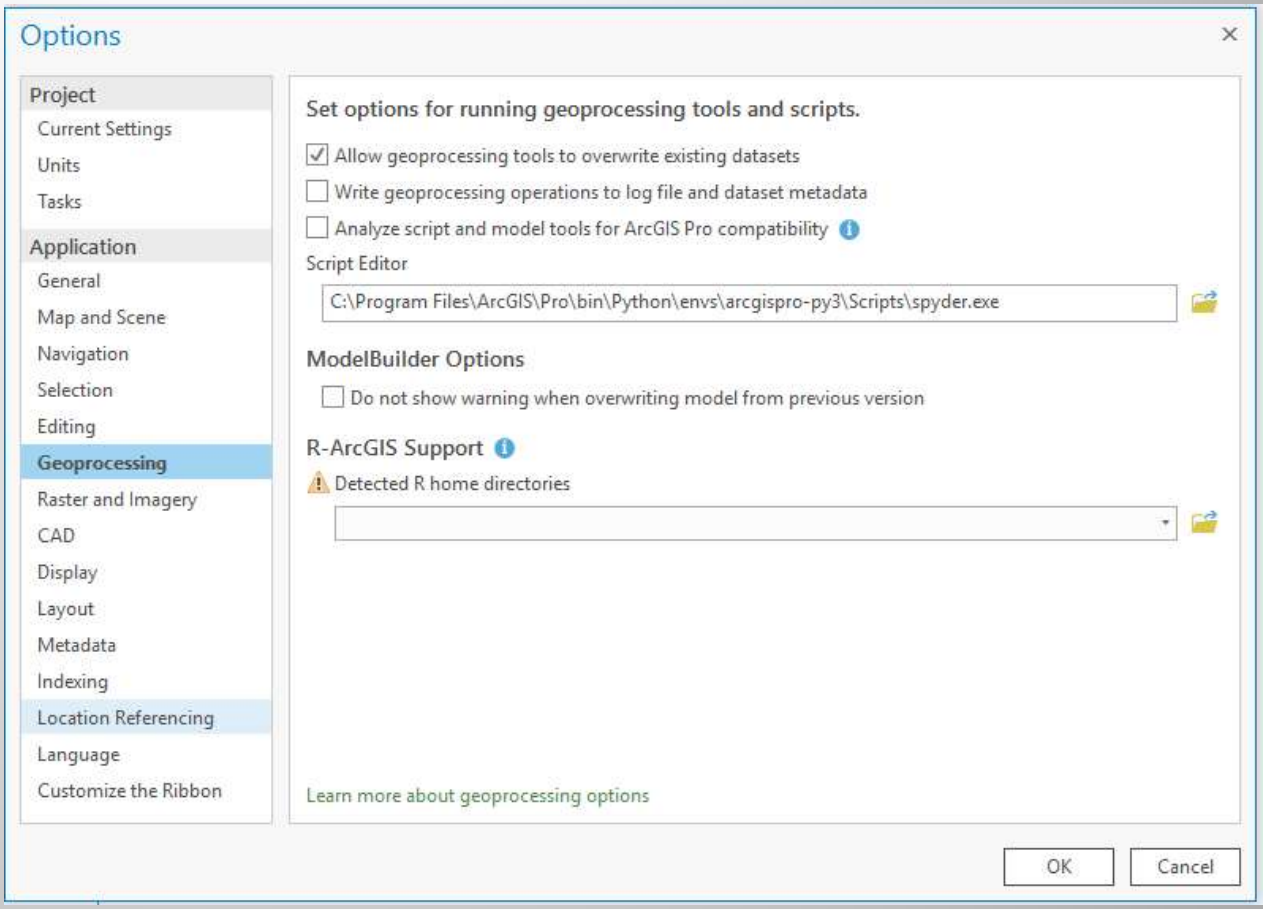
# LAB 1

- Run the same procedure in all aspects of the geoprocessing framework

- But first…
  - Move the course data from Q:\Courses\CEL\G9029\ModelBuilderPython\MB_Data to your local machine, C:\...\Projects
  - Open ArcGIS Pro and set up Spyder as the default script editor
  - Specify environment settings
  - Double check inputs and output locations for data in model

# SET UP SCRIPT EDITOR

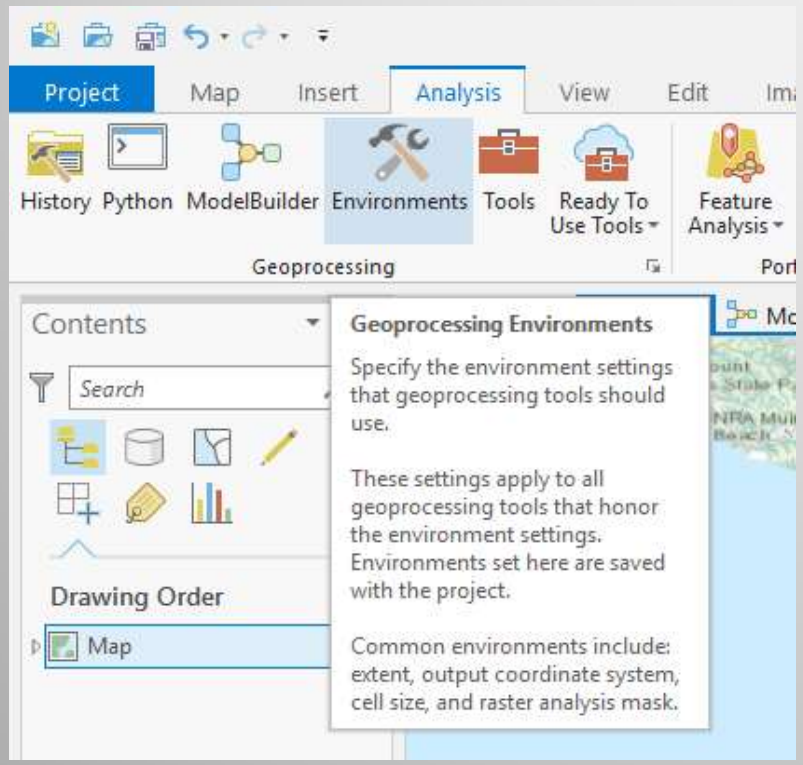
- Project Tab
- Click on Options
- Choose Geoprocessing
  - Script Editor

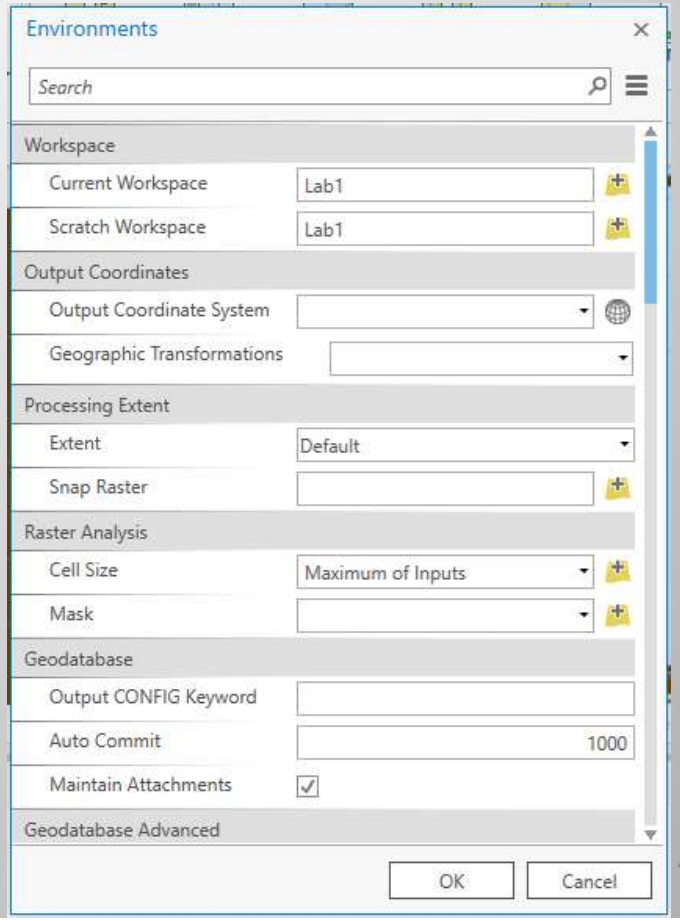



- C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\Scripts\spyder.exe

# ENVIRONMENT SETTINGS

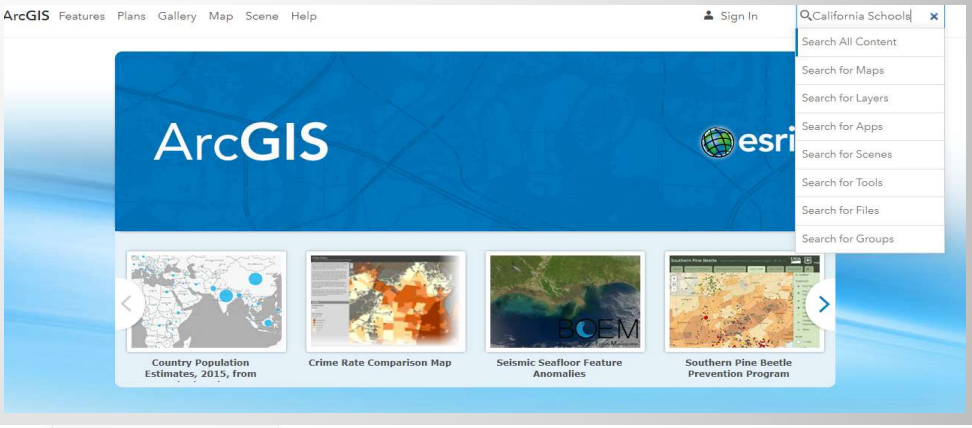
- Set the default workspace to C:\...\MB_Data\Lab1
- Under the Analysis Tab



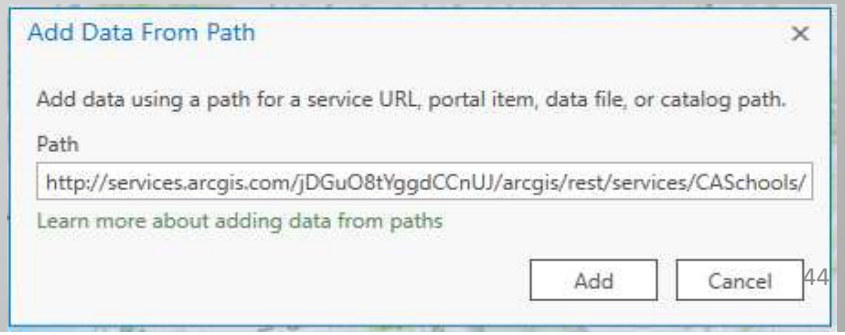



# DEMONSTRATION 1

- Adding a layer to ArcGIS Pro from an ArcGIS Online Feature Layer
- Using these layers as the inputs to geoprocessing tools
- Outputs from feature class or shapefile





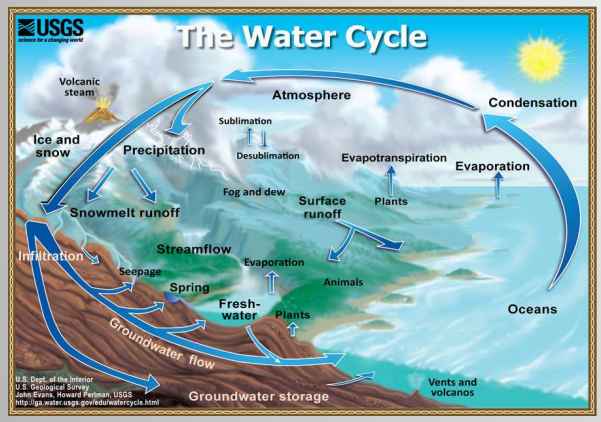


# SECITON 2

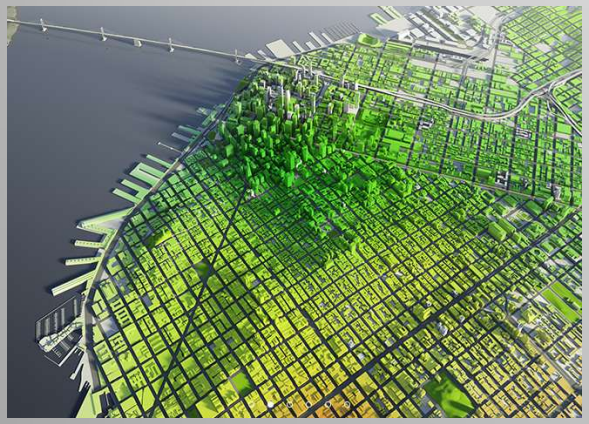
- What is a model?
- Planning an analysis
- Getting started with ModelBuilder

# TYPES OF MODELS

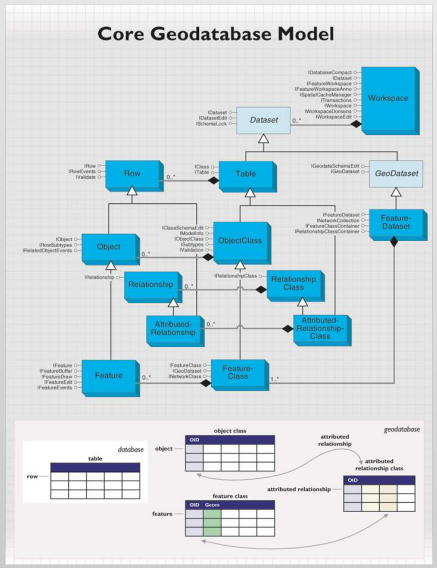
- In GIS, Models can mean many different things.



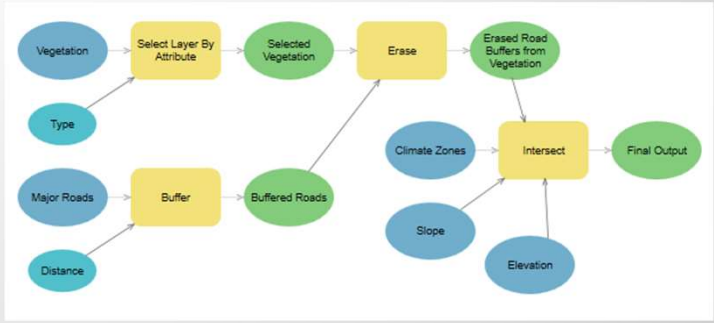

Physical Models



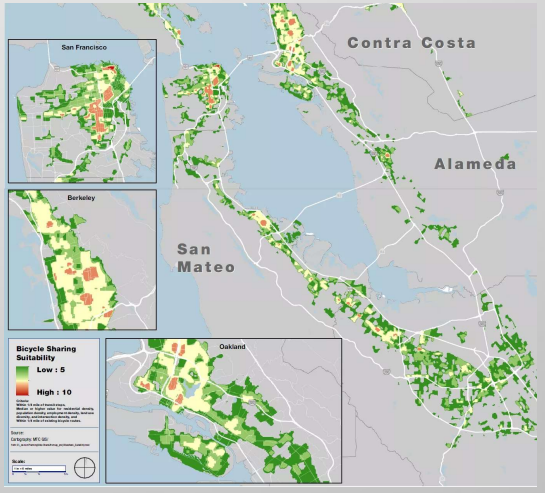
Representative Models



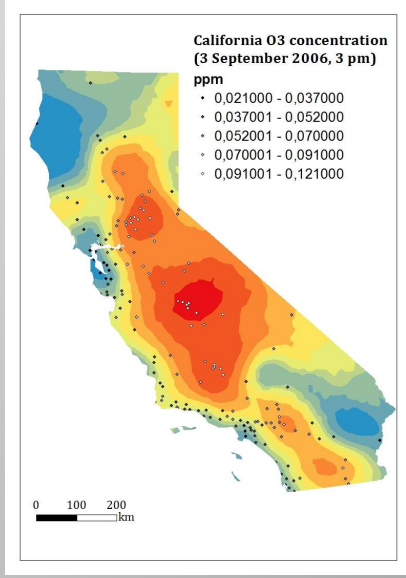

Data Models




Procedural Models




Site Suitability Model



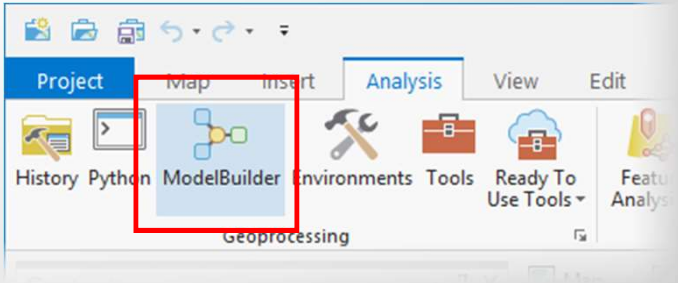

Statistical Models

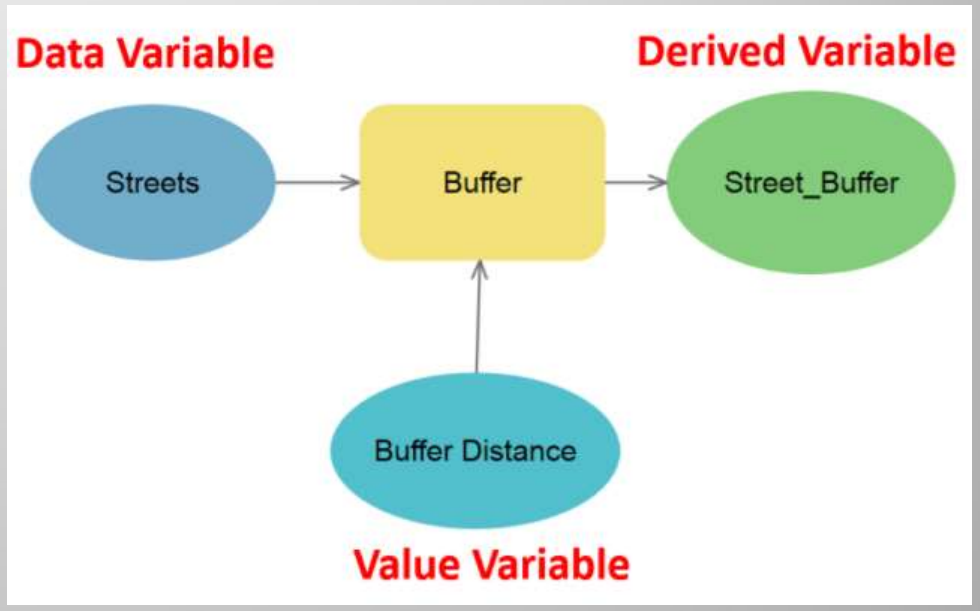
# DEVELOPING YOUR PROCEDURAL MODEL

- Think about the steps ahead of time, make a plan!
- Make sure you have all your data inputs.
- What's the objective? Can it be accomplished with a Model?
- Is the effort spent developing a model worth the output? (i.e. running one tool once is not effective for model builder.)
- Always good practice to draw our write out the steps, tools and data you'll need for the model.

# MODELBUILDER

- **ModelBuilder** is a visual workflow diagram composed of a collection of geoprocessing operations that automatically execute in sequence when the model is run to produce a final output dataset.
- Inputs are connected to tools with connectors and tools are directed to outputs, which in turn can be used as inputs.



# MODELBUILDER VOCABULARY

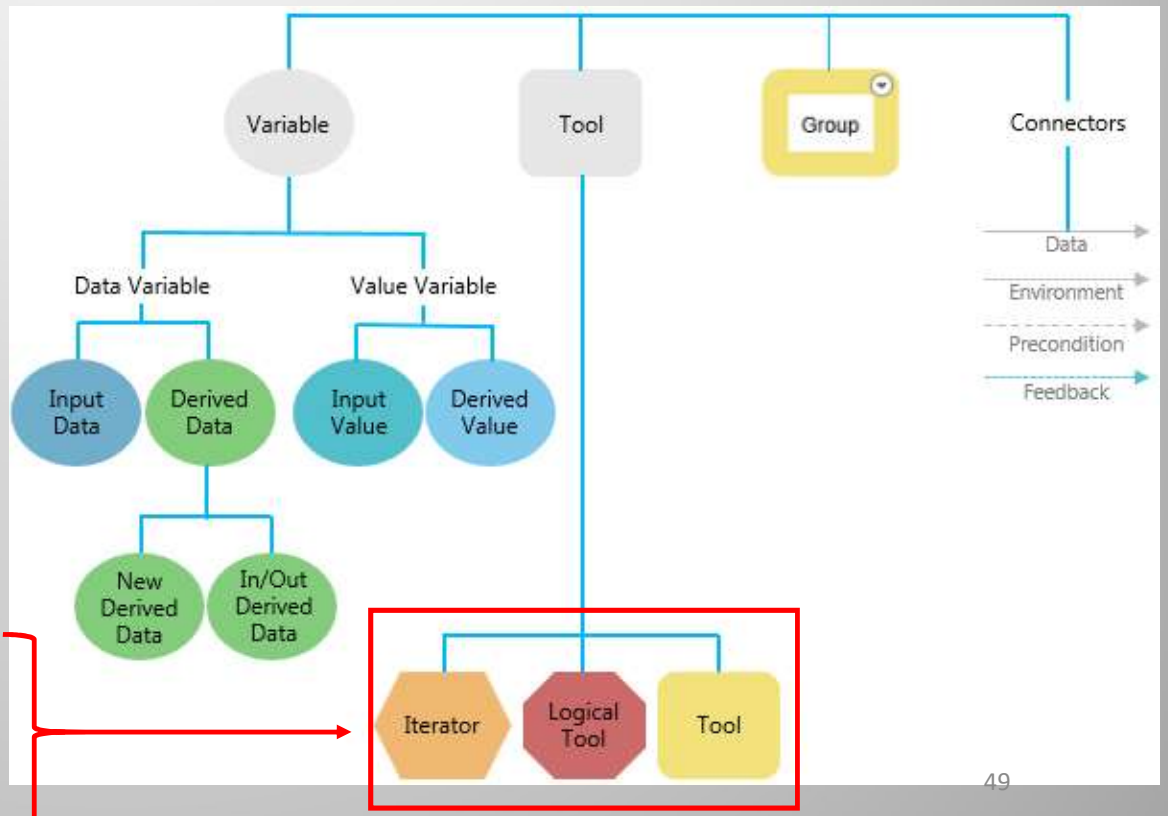
| | | |
|---|---|---|
| **DATA VARIABLE** | Input Data | Common data variables include feature class, feature layer, raster dataset, and workspace. |
| **DERIVED or OUTPUT DATA VARIABLE** | Derived Data | When a geoprocessing tool is added to a model, variables for the tool's output parameters are automatically created and connected to the tool. |
| **VALUE VARIABLE** | Input Value | Value variables contain anything but references to layers or data stored on disk. |
| **DERIVED VALUE VARIABLE** | Derived Value | Values that are the result of a tool. Derived values can be inputs to other tools. |
| **GROUP** | Group | Groups are visual categories that include other elements in the model. |
| **TOOL** (Iterator, Utilities and Logical) | Tool | Tools are geoprocessing tools added to the model. These include all tools you will find in a system toolbox as well as custom model and script tools. |

**Variable** – Values and Data

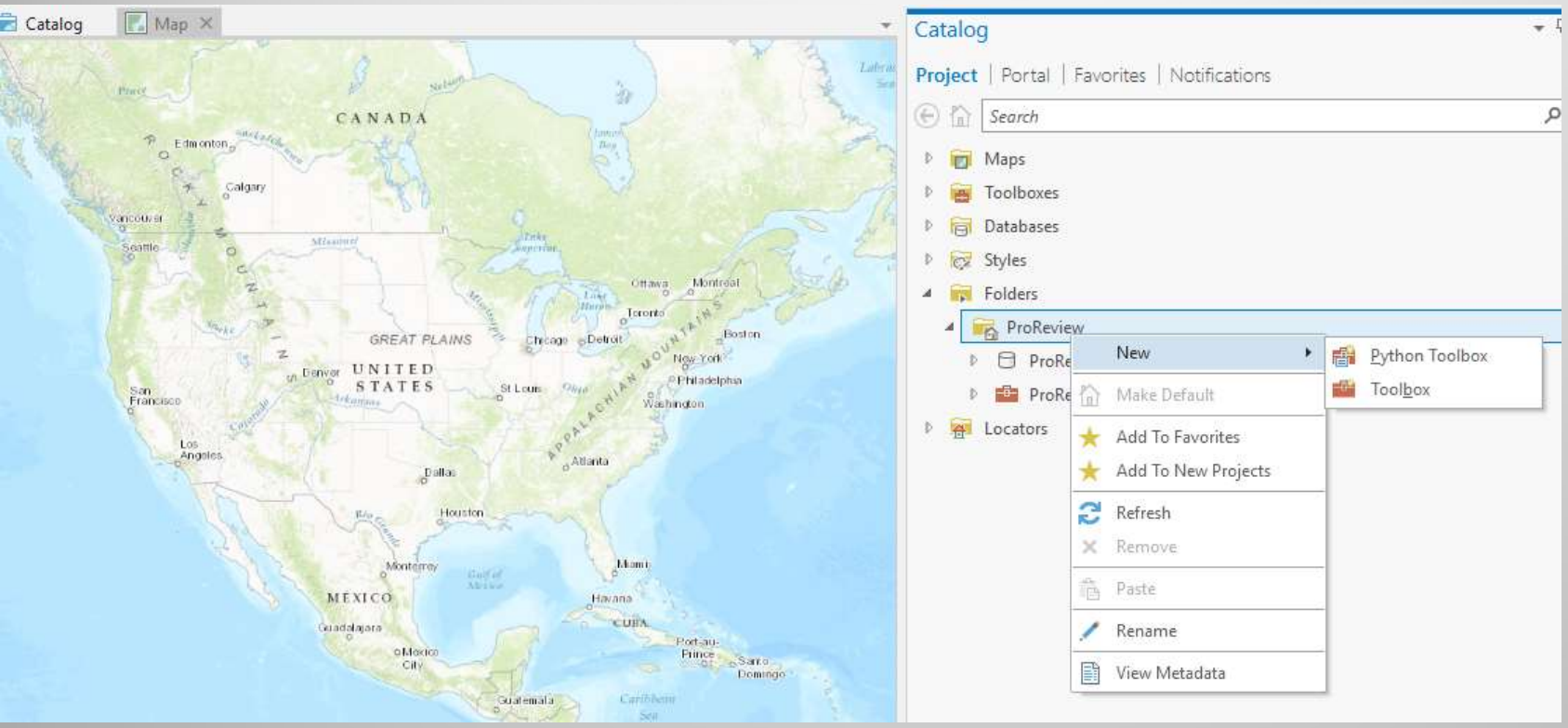
**Tool** – Geoprocessing functions

**Group** – Container of other grouped model elements

**Connectors** – show the direction of processing



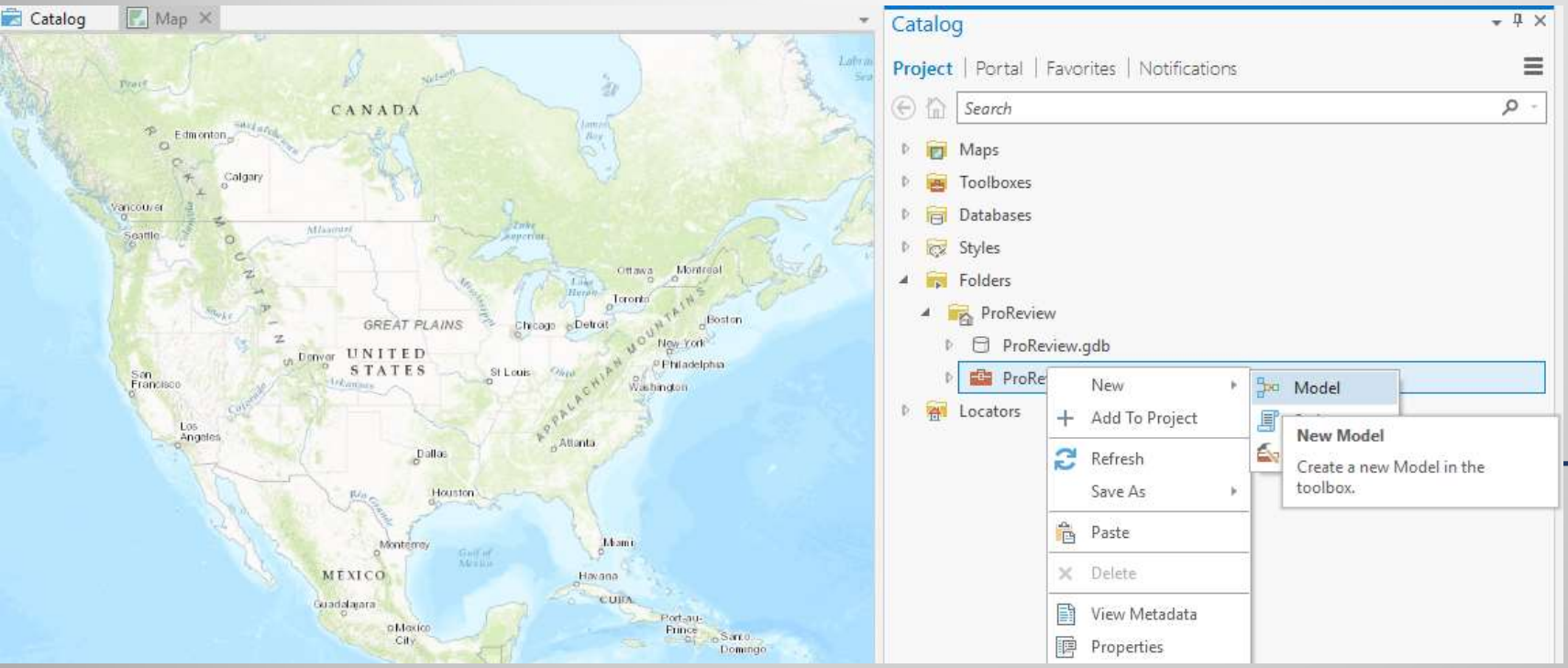
# CUSTOM TOOLBOX




Create a custom tool box (Catalog Pane)

File → New →Toolbox

# CREATE A MODEL

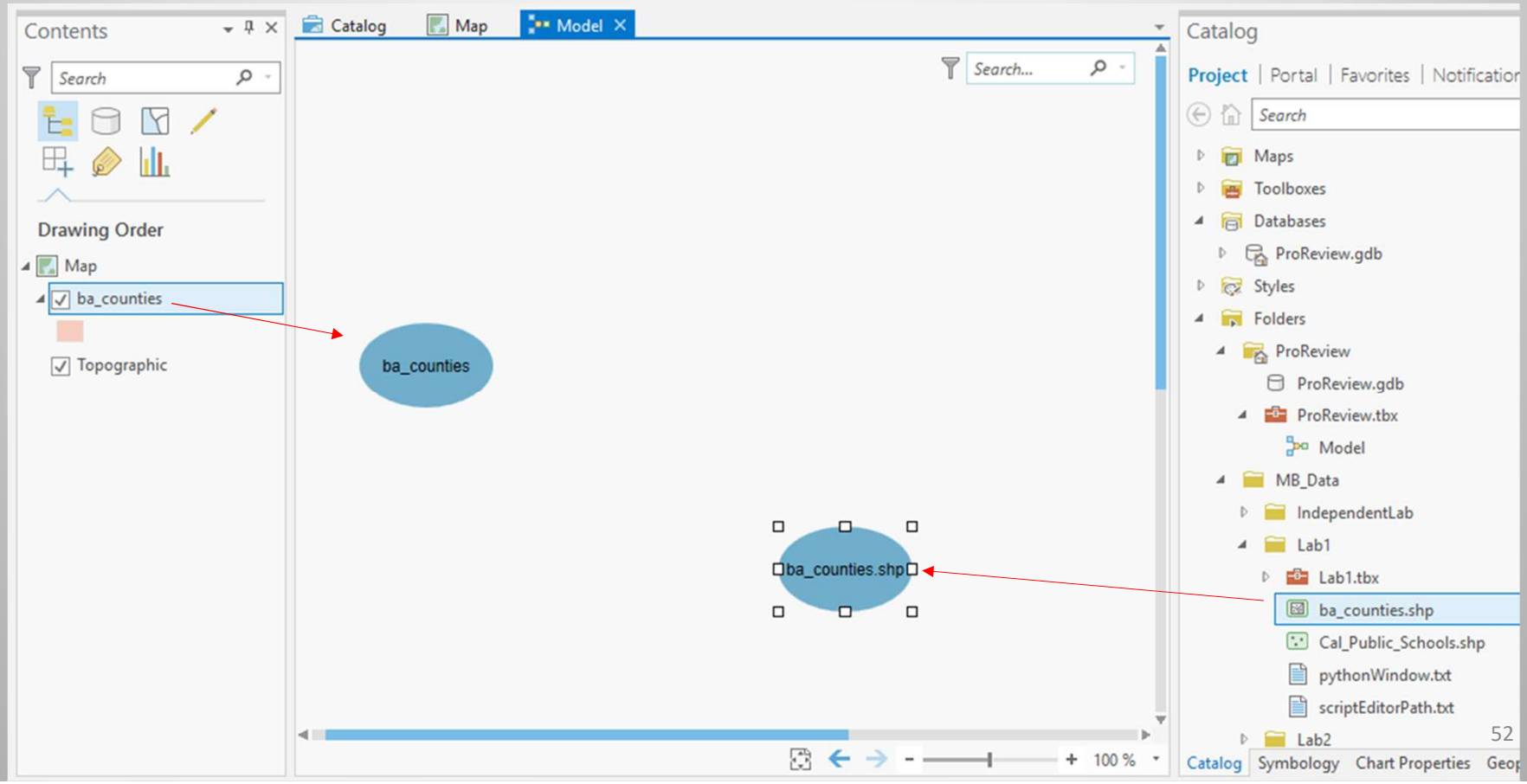



Add a new model to the custom tool box (Catalog Pane)

Custom Toolbox → New → Model

# ADDING DATA AND TOOLS TO A MODEL

Drag & Drop from Contents Pane
or Catalog Pane



# THREE STATES OF A MODEL

1) Not Ready to Run
2) Ready To Run
3) Run and Completed

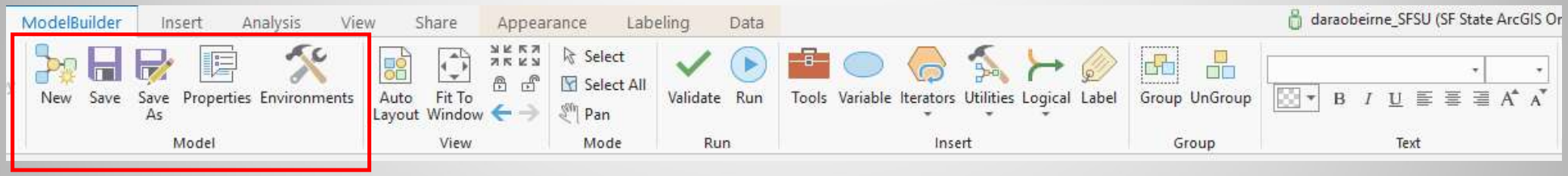
Completed Running – Elements are Colored with Shadow


Not Ready – Elements are grey


Ready to Run – Elements are Colored

# MODELBUILDER TAB RIBBON

Only available when in Model View

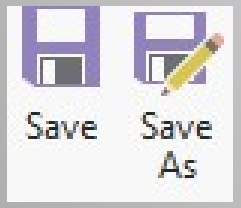



## MODEL GROUP

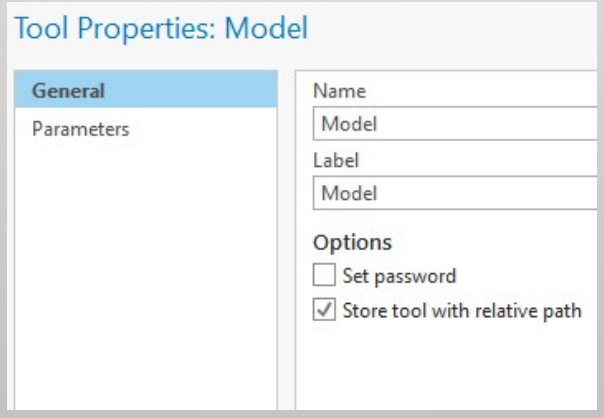

Create a new model

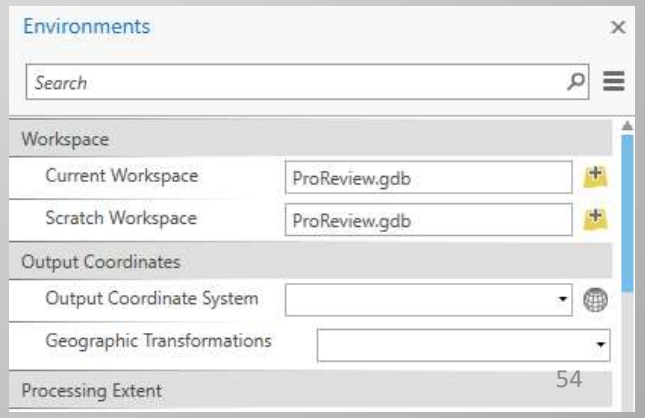

Model Properties (naming etc.)


Model Environments

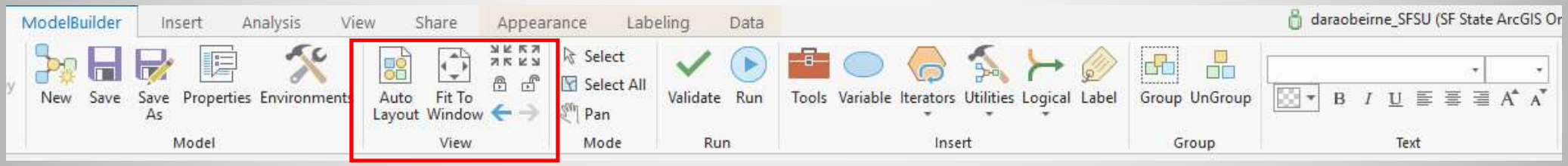



Save the model



# MODELBUILDER TAB RIBBON




## VIEW GROUP


Auto Layout (re-arranges elements)

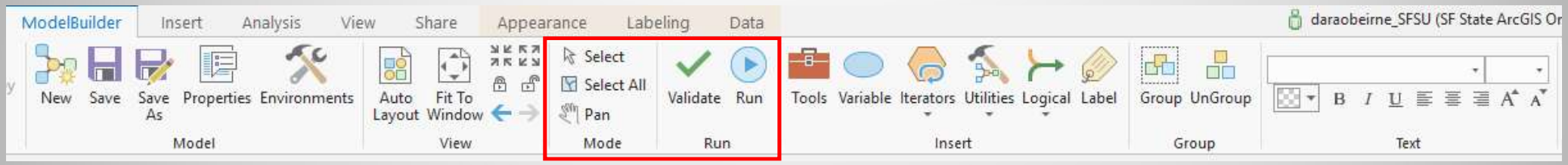

Fits all elements to the model view

Zooms in & Zooms Out (use mouse wheel too)

Lock and unlock elements (don't participate in auto layout)

Previous and Next Extents

# MODELBUILDER TAB RIBBON



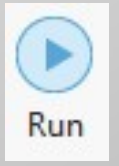

## MODE & RUN GROUPS


Selects elements of the Model to move


Pan around the Model view

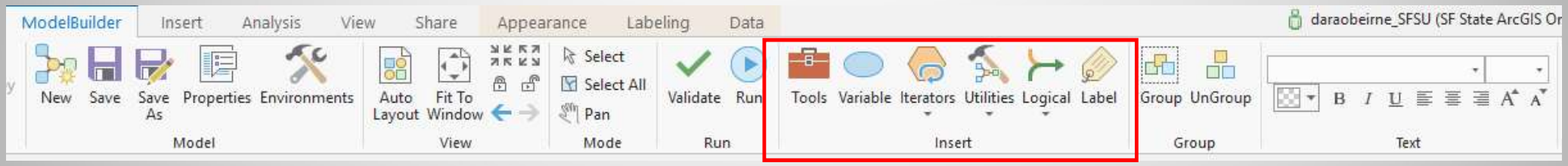

Validates the model before you run to confirm no errors


Selects all elements of Model to move at once

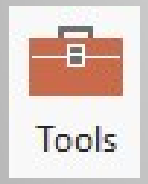



Runs the Model

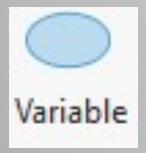
# MODELBUILDER TAB RIBBON




## INSERT GROUP

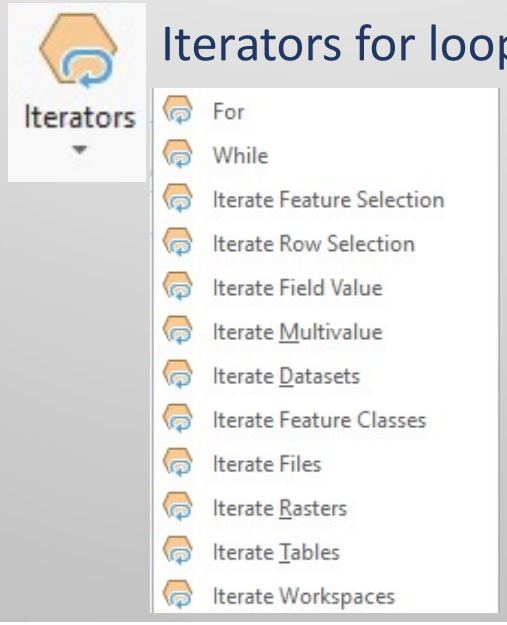



Shows tools in the Geoprocessing Pane

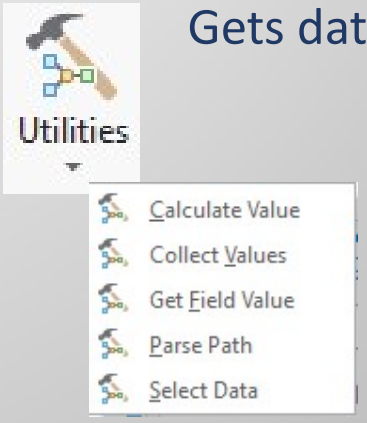



Creates a variable that stores values (distance)

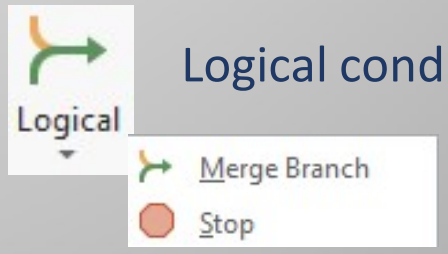

Label elements of the Model




Iterators for loops

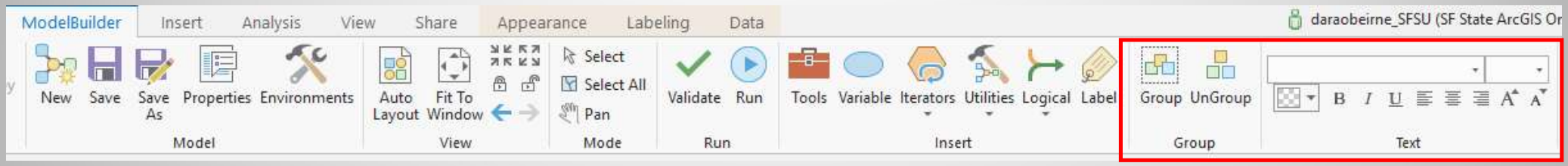



Gets data

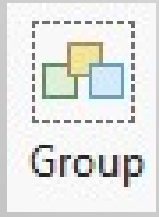



Logical conditionals

# MODELBUILDER TAB RIBBON

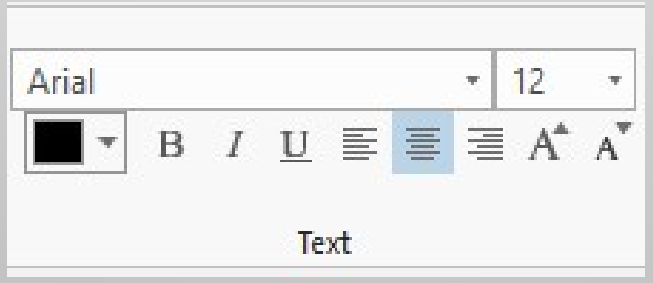



## GROUP & TEXT GROUPS




Group elements of Model

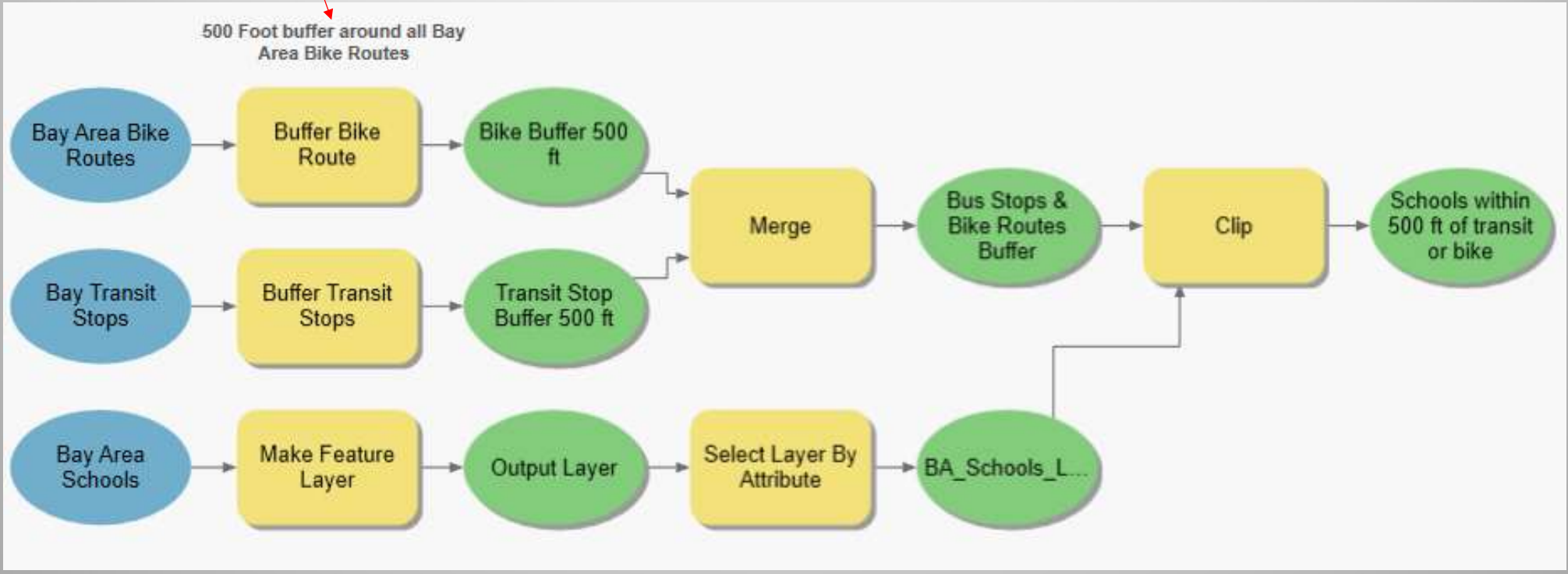

UnGroup elements in a Group



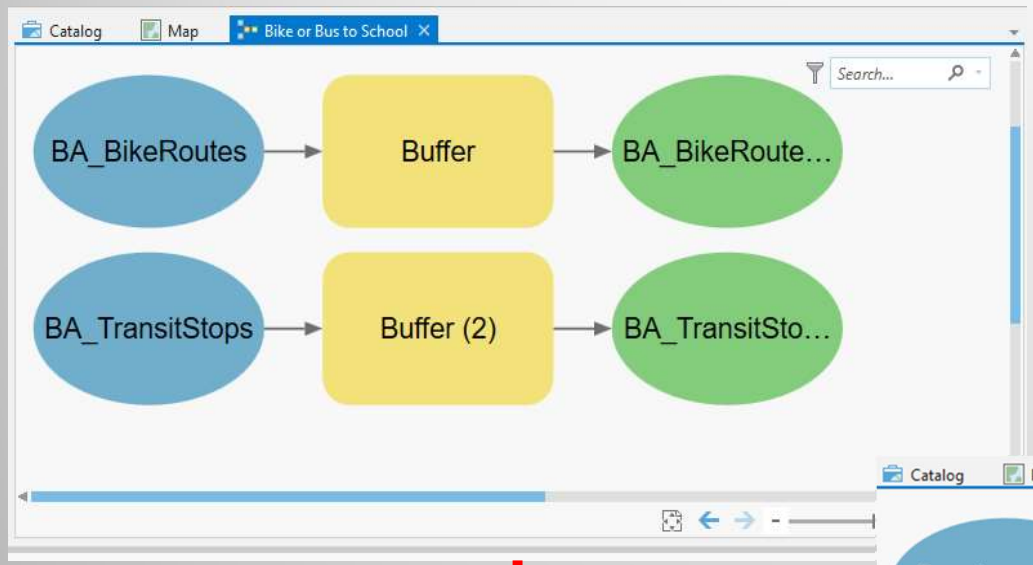

Font and size of labels

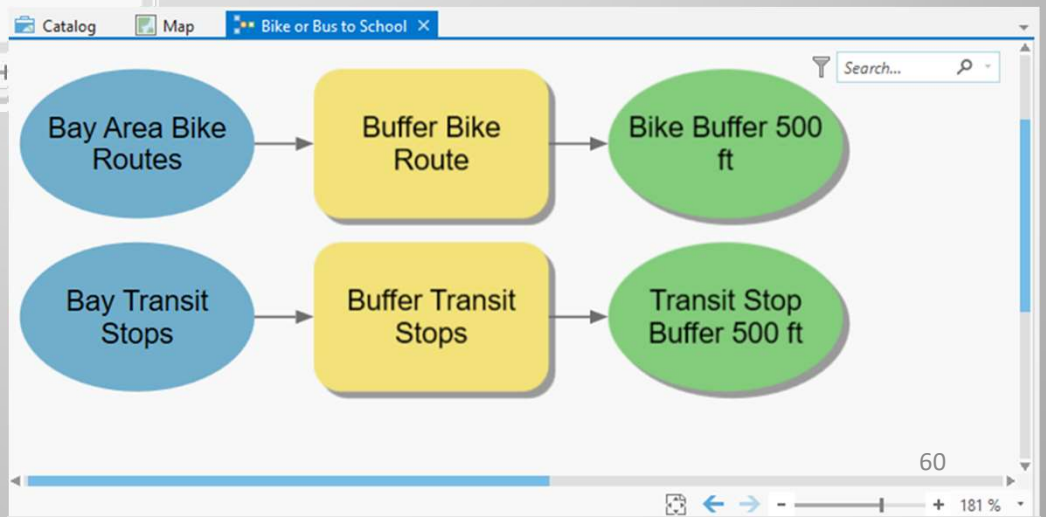
# LABELING

Labels will help you and anyone else who uses the model remember what things are!



# RENAMING MODEL ELEMENTS

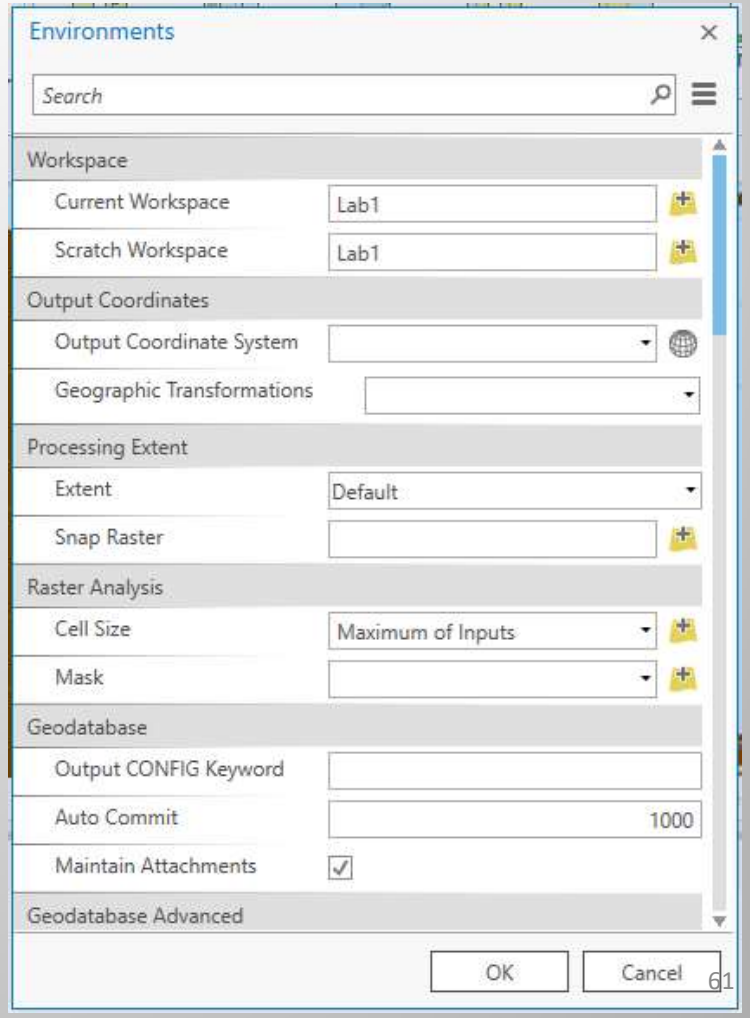


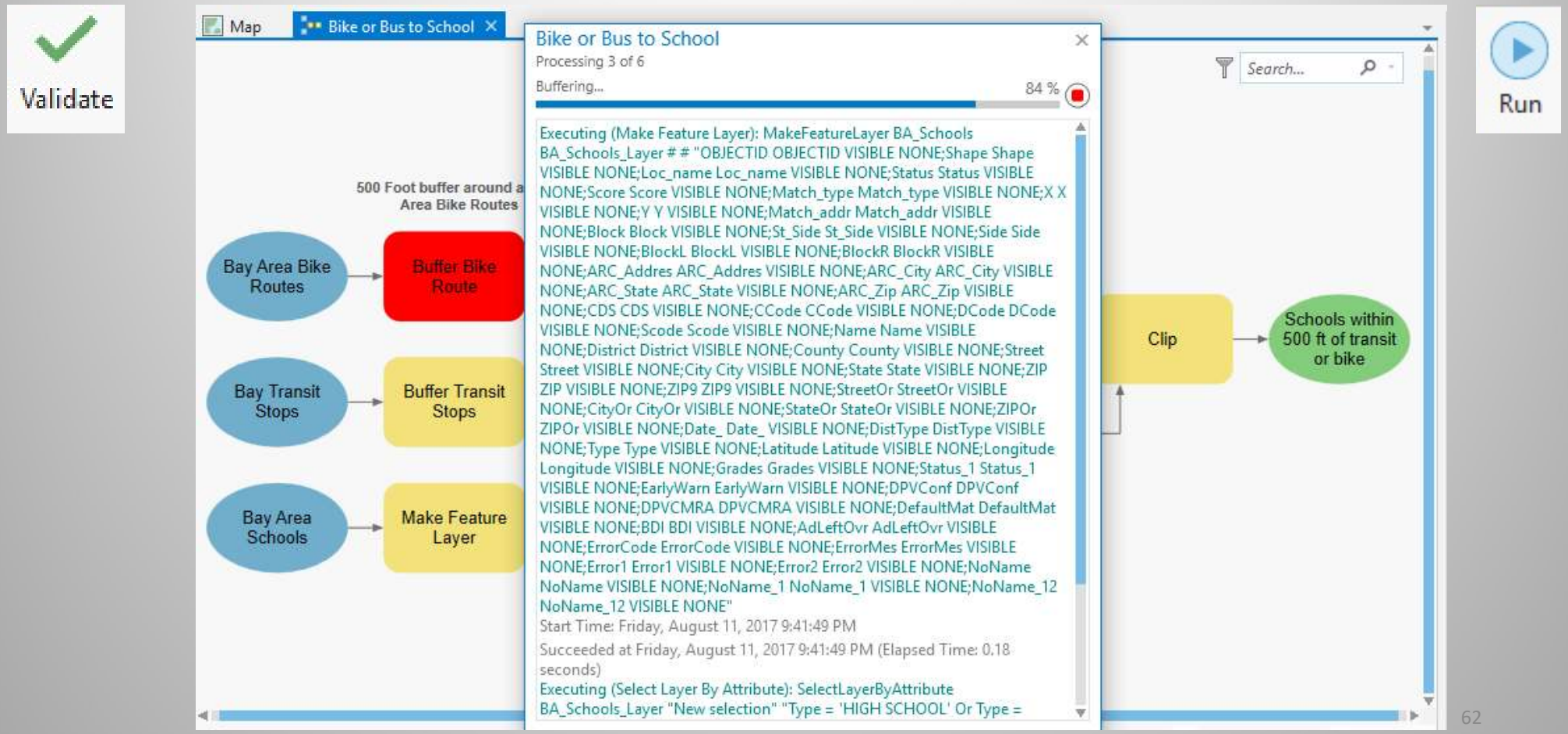

- ModelBuilder will automatically name components
- Try renaming to improve readability

# ENVIRONMENT SETTINGS

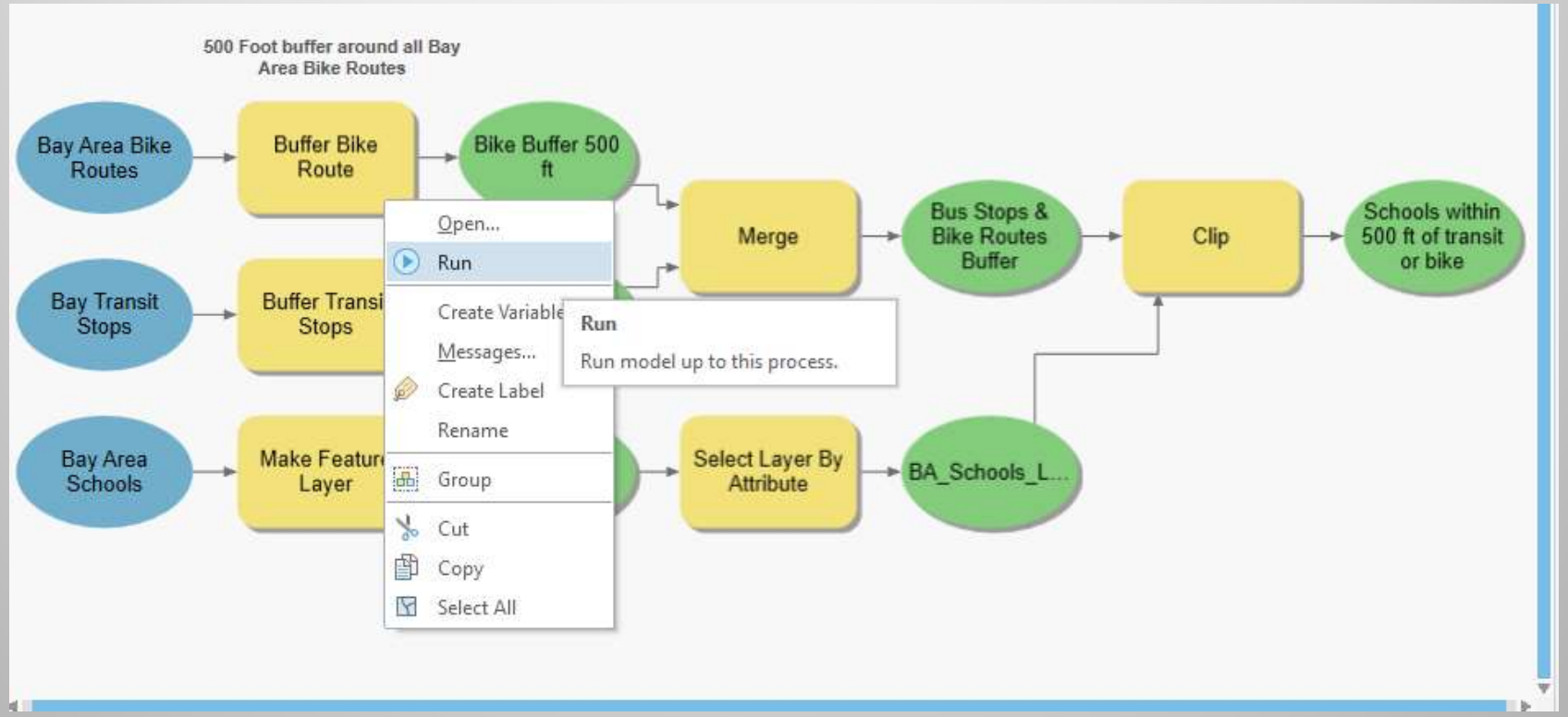
- Set the default workspace to a designated location
- Button is located under the Analysis Tab
- Useful location for setting advanced tool parameters (for example define cell size or output coordinate system)
- Remember 3 places you can set environment settings (map, tool & model), here we can do it for the model



# RUNNING A MODEL

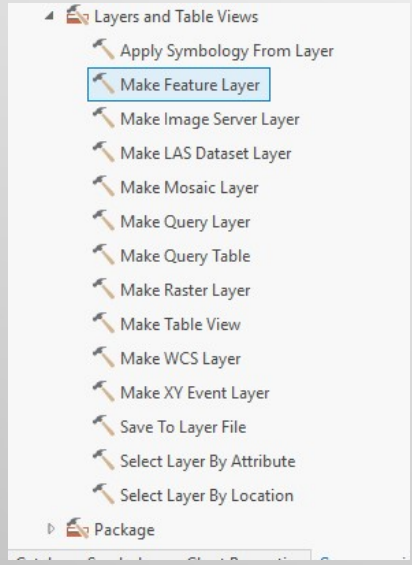


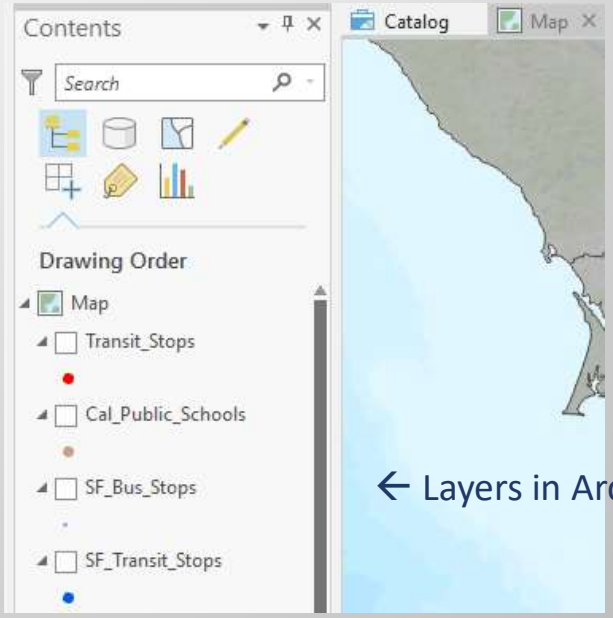
# RUNNING INDIVIDUAL TOOLS



# FEATURE LAYERS

- Some tools work with feature layers—not on the data itself
- Examples:



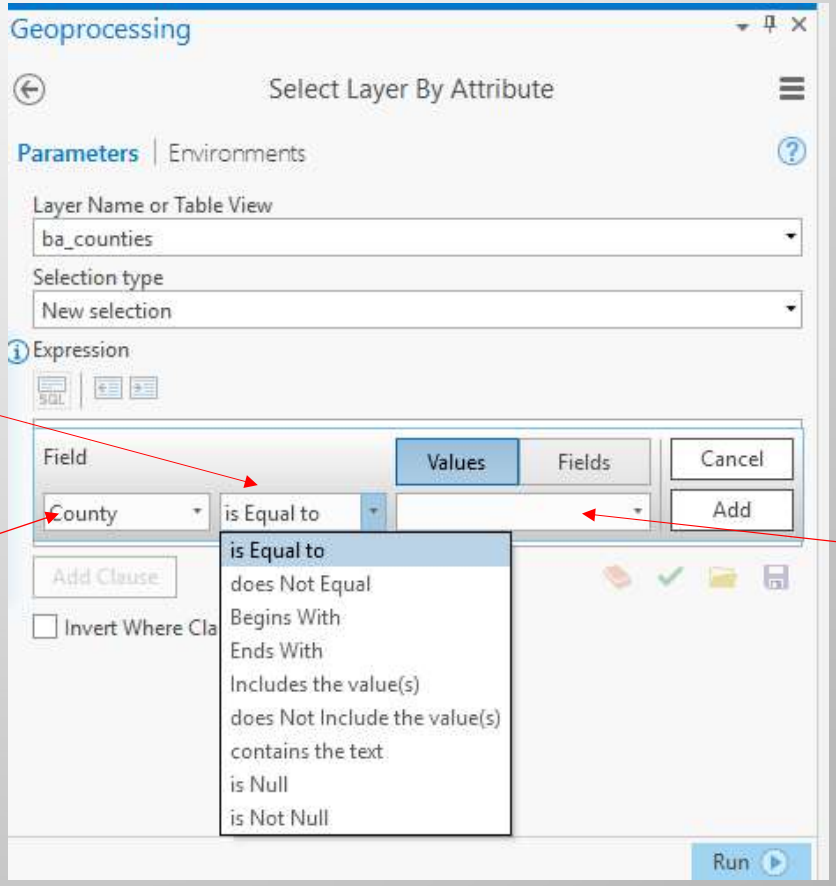




← Layers in ArcGIS Pro

- Therefore you will need to create a feature layer from the input dataset first

# EXPRESSIONS

- Can do SQL or ADD Clause
- SQL = Structured Query Language
- Used to subset features and table records
- column name + operator + value



# LAB 2

- Set up and build a simple model
- Run multiple processes simultaneously

# LAB 2: Questions

1. In this particular scenario can you think of two examples of why it would be advantageous to have the workflow in a model rather than in a running the same tools manually?

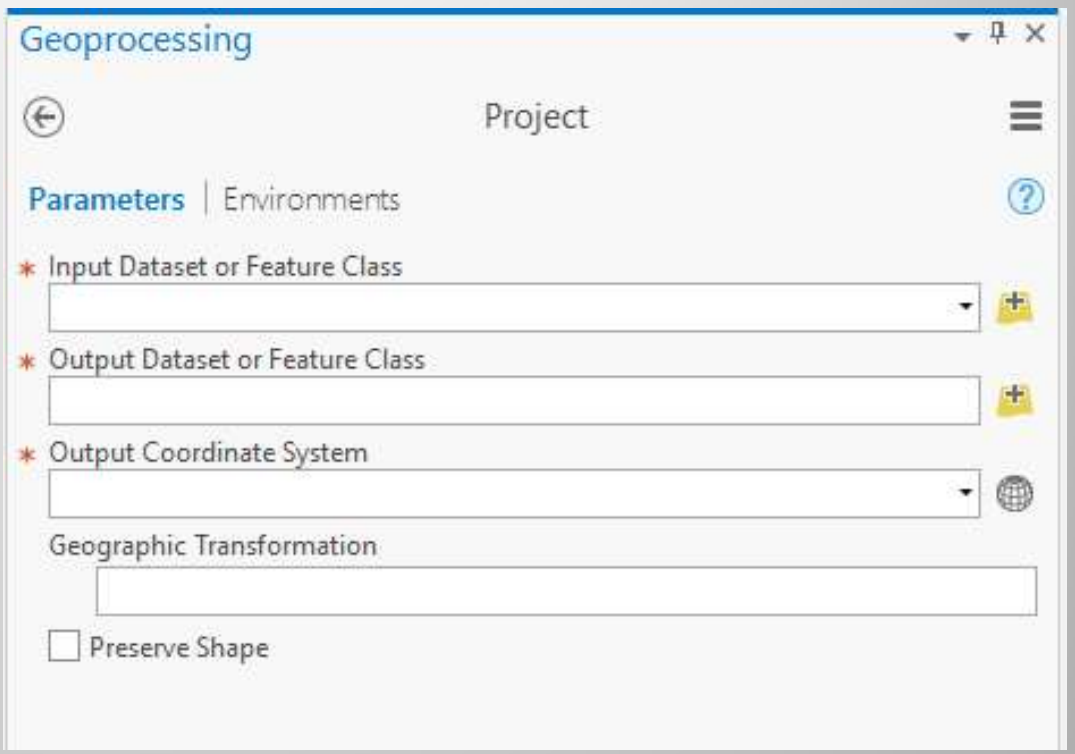
# SECTION 3

1. Model Parameters
2. Intermediate Data
3. Relative Paths for Sharing

# Parameters

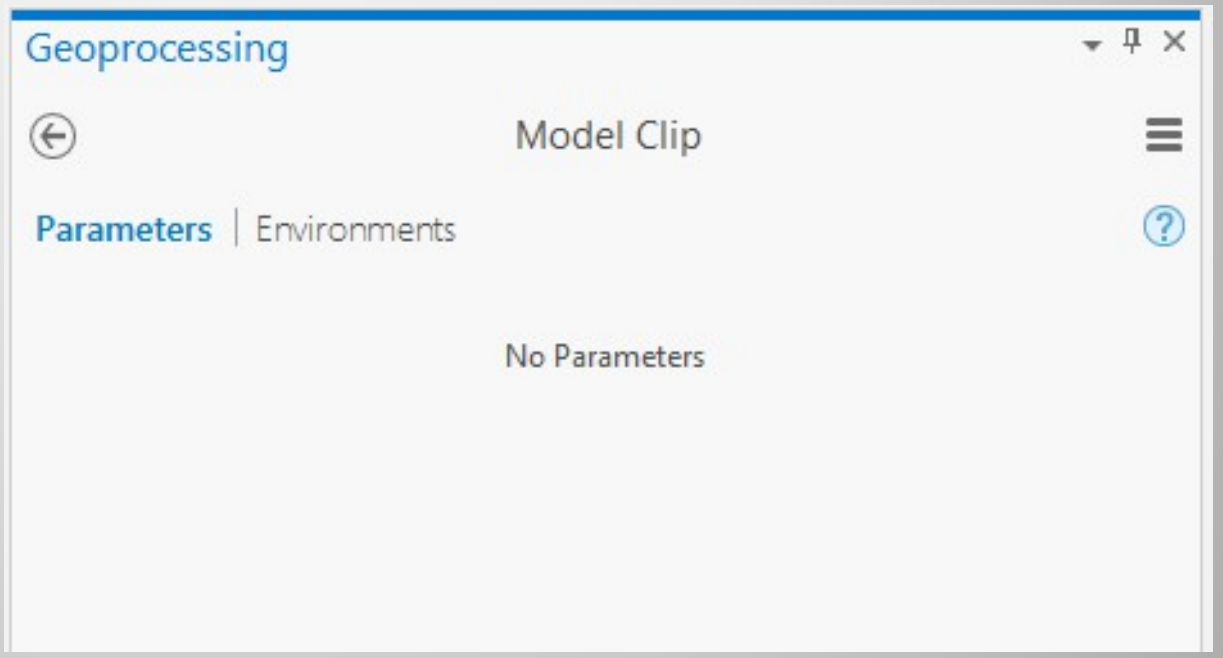
- Allow you to vary inputs and outputs
- Allow you to vary values, expressions, etc.
- Without parameters your model can't be easily adapted to new situations

Built-in tools have required ( * ) and optional parameters

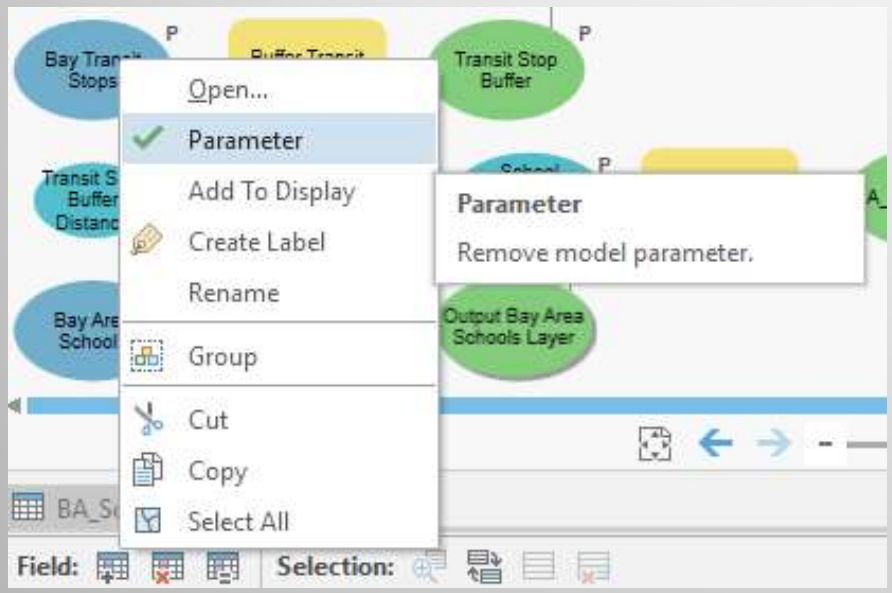


# A model tool without parameters:

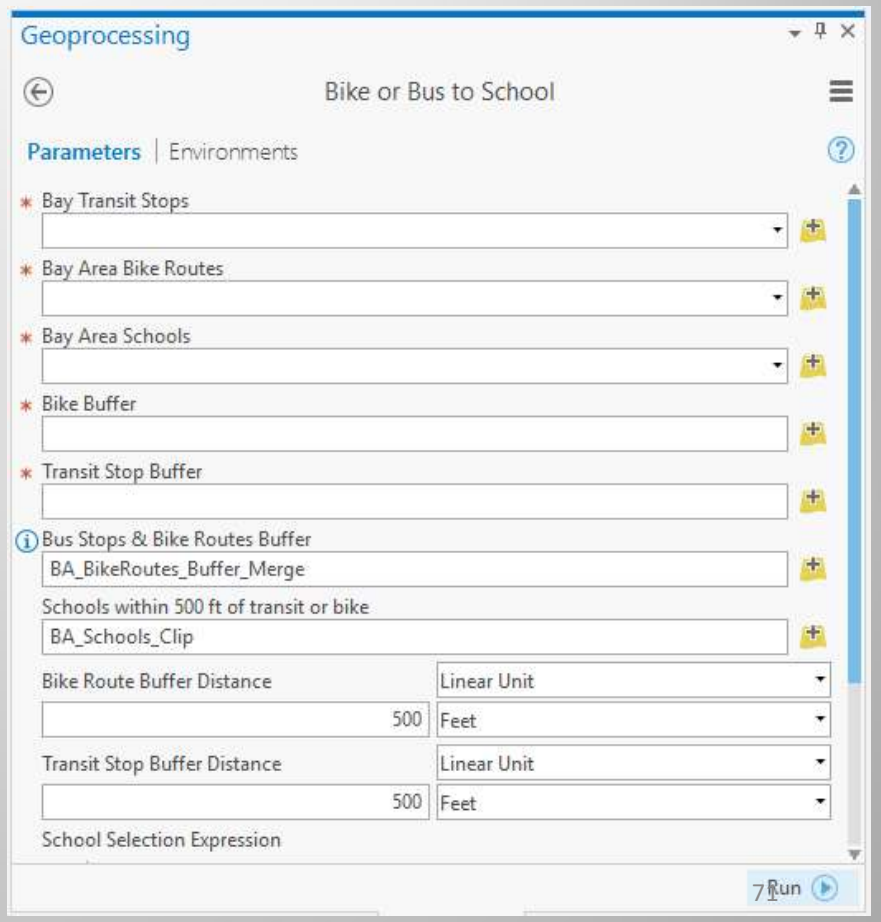
- Must edit the model itself to make any changes to inputs/output
- Makes sharing the model more difficult
- Inputs and outputs are not flexible



# Designating parameters in ModelBuilder

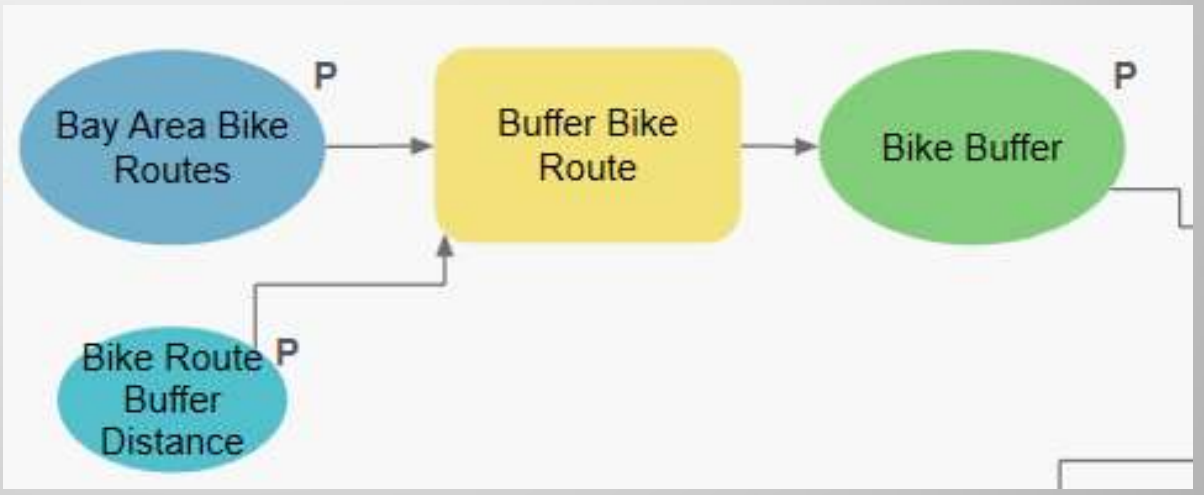



Note that whatever you name the parameter ovals in ModelBuilder, is how they will display in the model tool



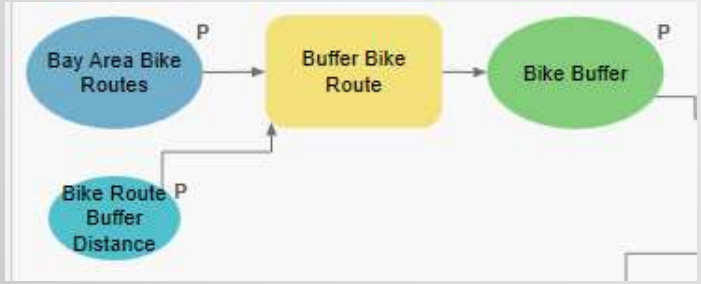
# Examples of parameters

- Input data
- Values
- Output data

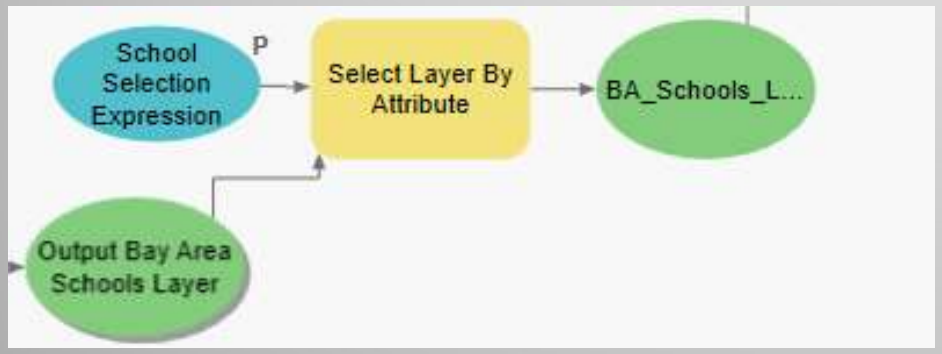


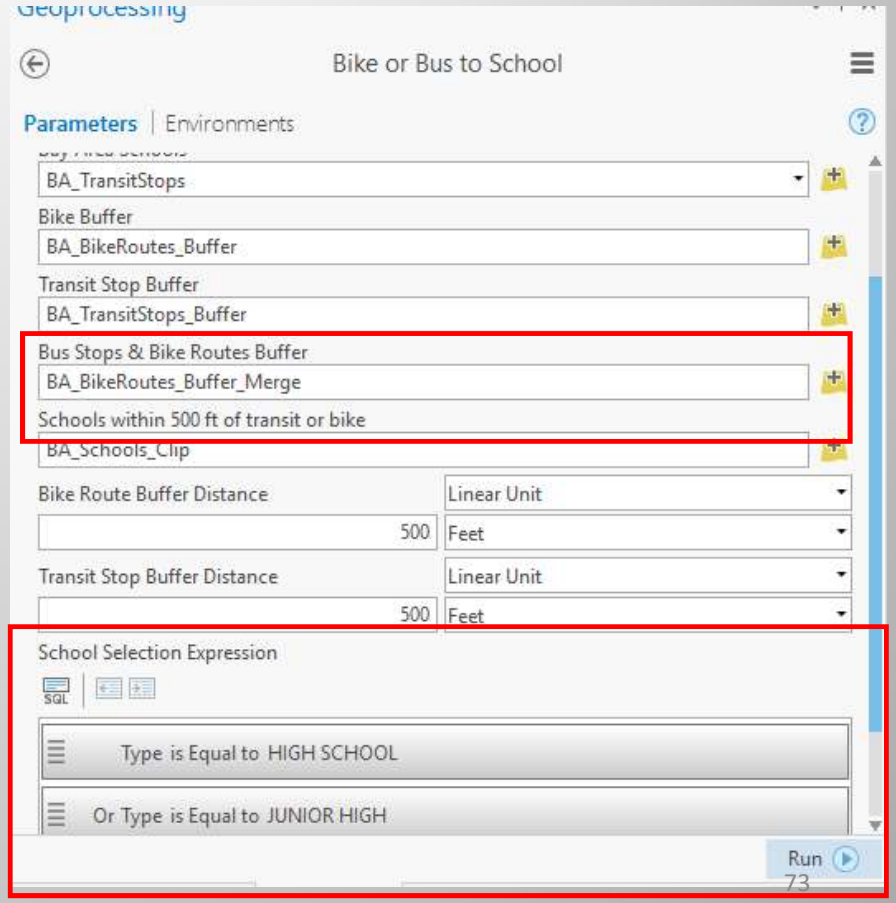
# Examples of parameters

- Distance



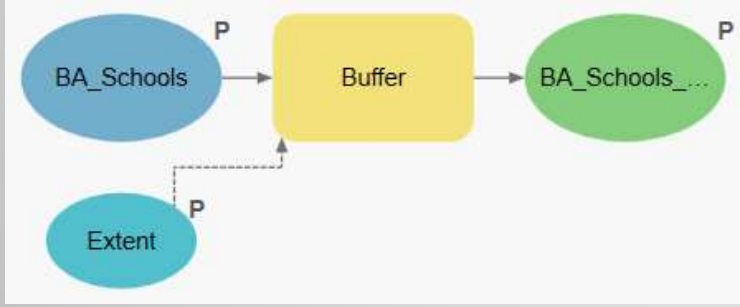

- Expressions

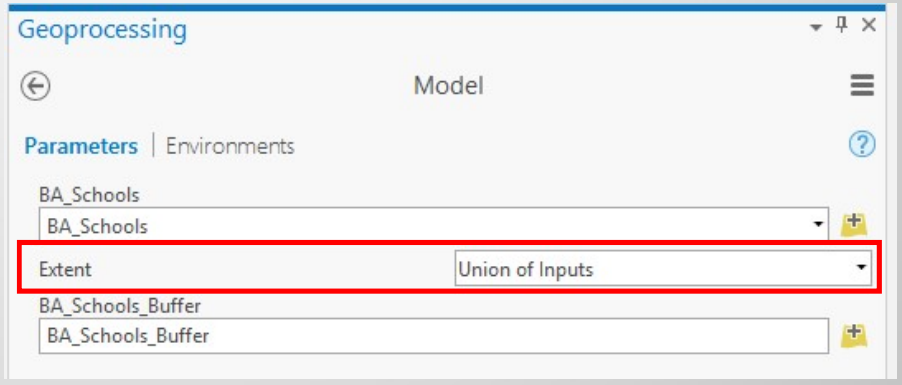


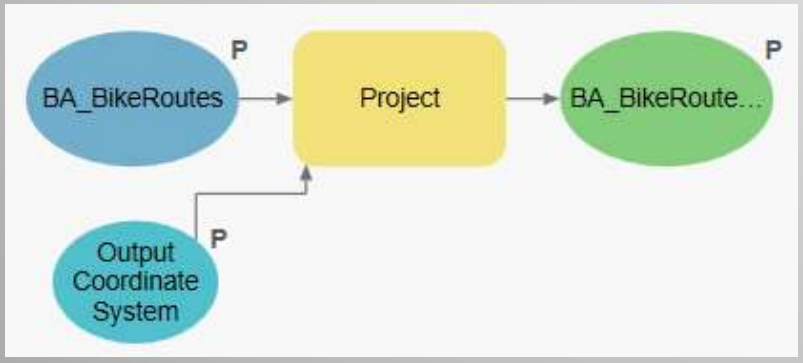
# Examples of parameters

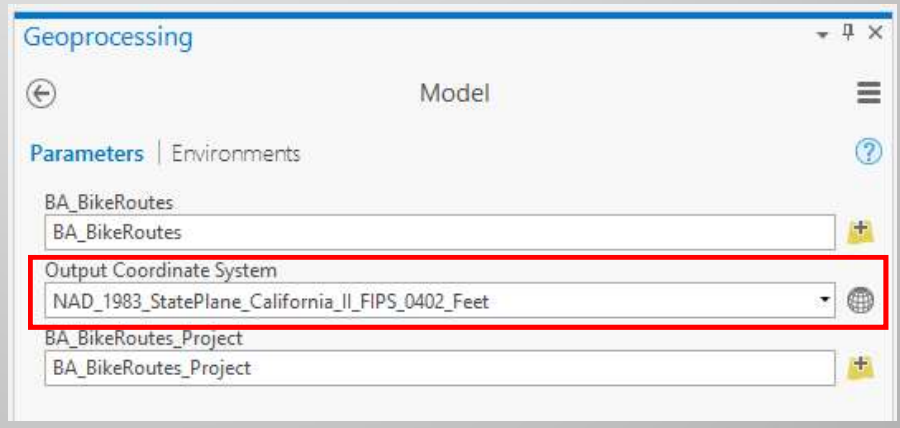
- Environment settings



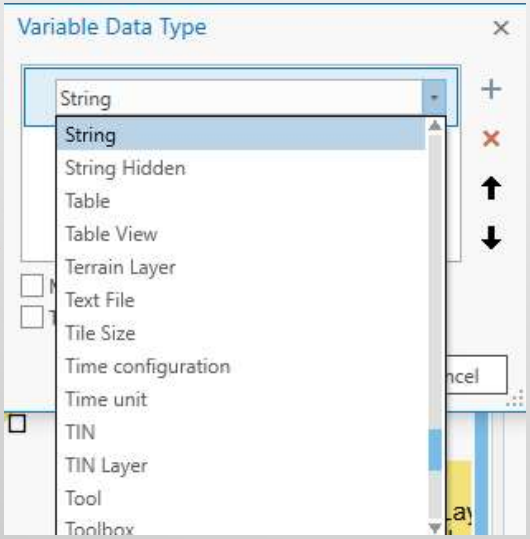

- Coordinate systems



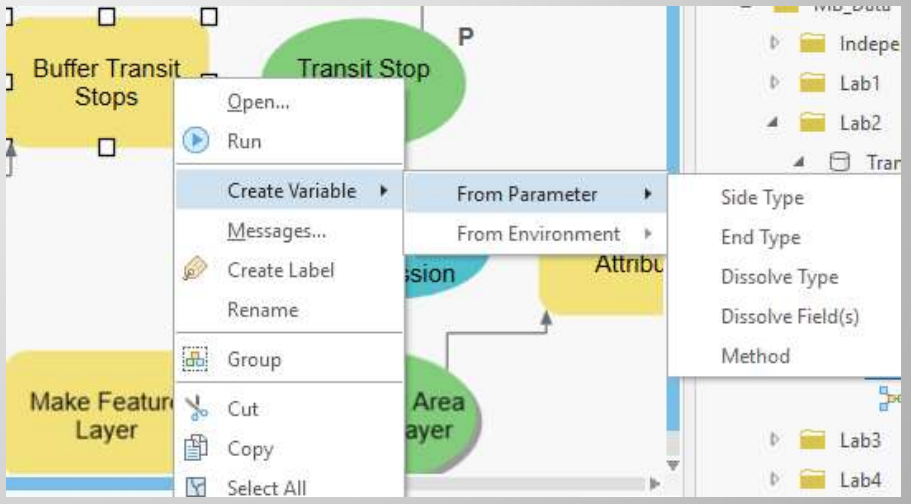



# Examples of parameters

- and many more……

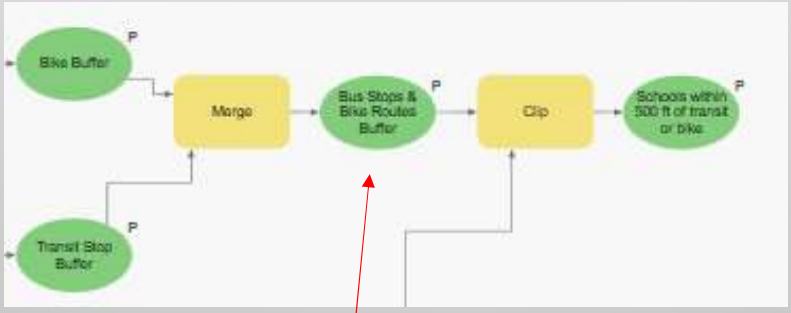

1. Insert a variable




2. Right-click on a tool and make a variable from a tool parameter

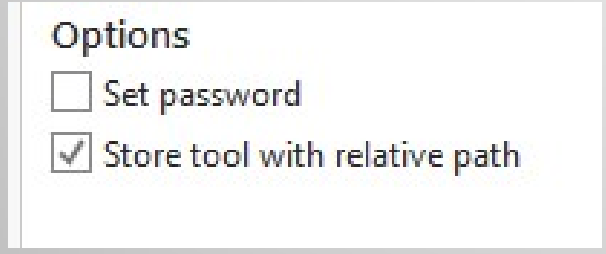
# IN-MEMORY WORKSPACE

- a system memory-based workspace where output feature classes, tables, and raster datasets can be written—the in_memory workspace
- Useful for intermediate data you don't want stored
- Is deleted once application closes
- Faster




# SHARING MODELS

- Provide the custom tool box that the model tool is stored in
- Provide any data that model needs
- Make sure to check relative paths!



# RELATIVE PATH NAMES

**Absolute Path**

- C:\LocalMachine\ProjectData
- Links will break when you move the folder
- Use absolute paths if data is stored in a central repository, like  a server

**Relative Path**

- ….\ProjectData
- Will reference anything in the same folder or folder group regardless of where the folder is stored
- Use relative paths if data is stored locally or is being transferred with the map document

# Exercise 3

- Add model parameters
- Document a model and provide parameter descriptions
- Add another tool to determine the number of vulnerable infrastructure points for each earthquake

# Exercise 3 Questions:

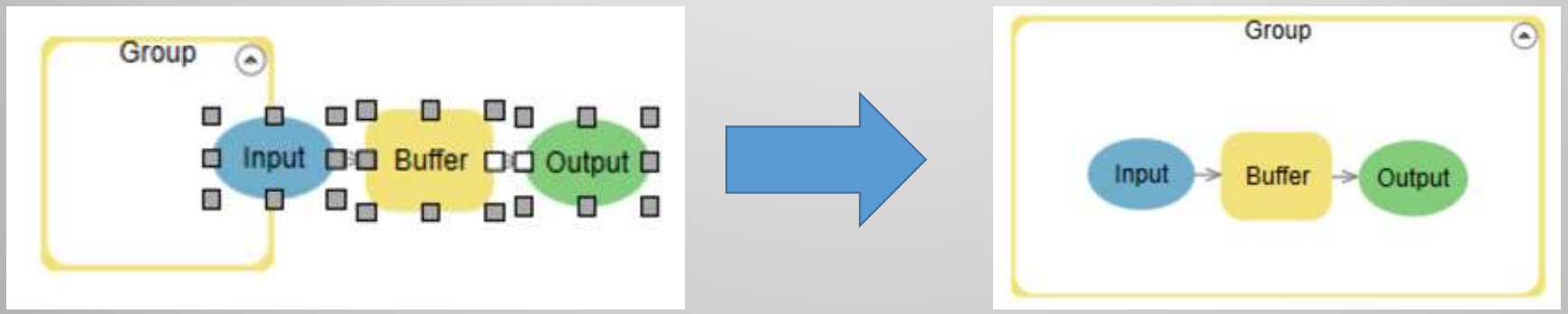
1. What is a model parameter and why should your model have them?
2. What's the difference between a tool parameter and a model parameter?
3. Why is it important to document your model (overall) and its parameters?

# Section 4

- Groups
- Iterators
- Nesting models
- Model Only Tools
- Branching

# GROUPS

- Allows you to collapse model elements into logical groupings
- Can be used to maximize space for complex models

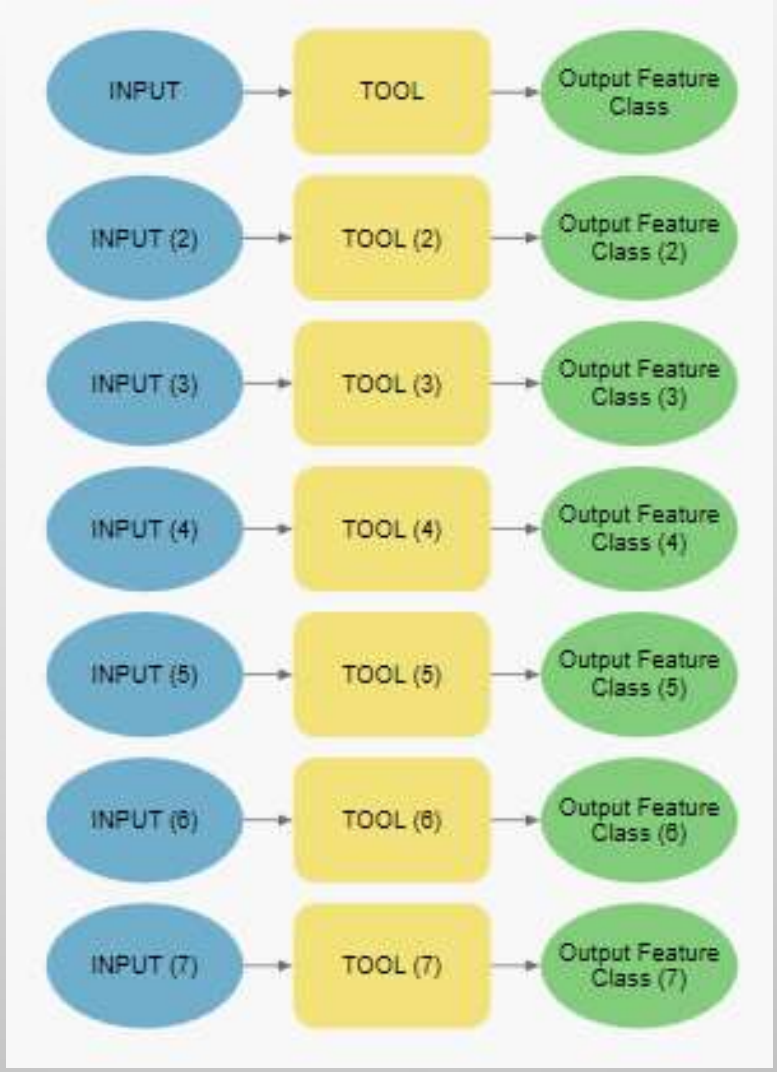


# ITERATORS

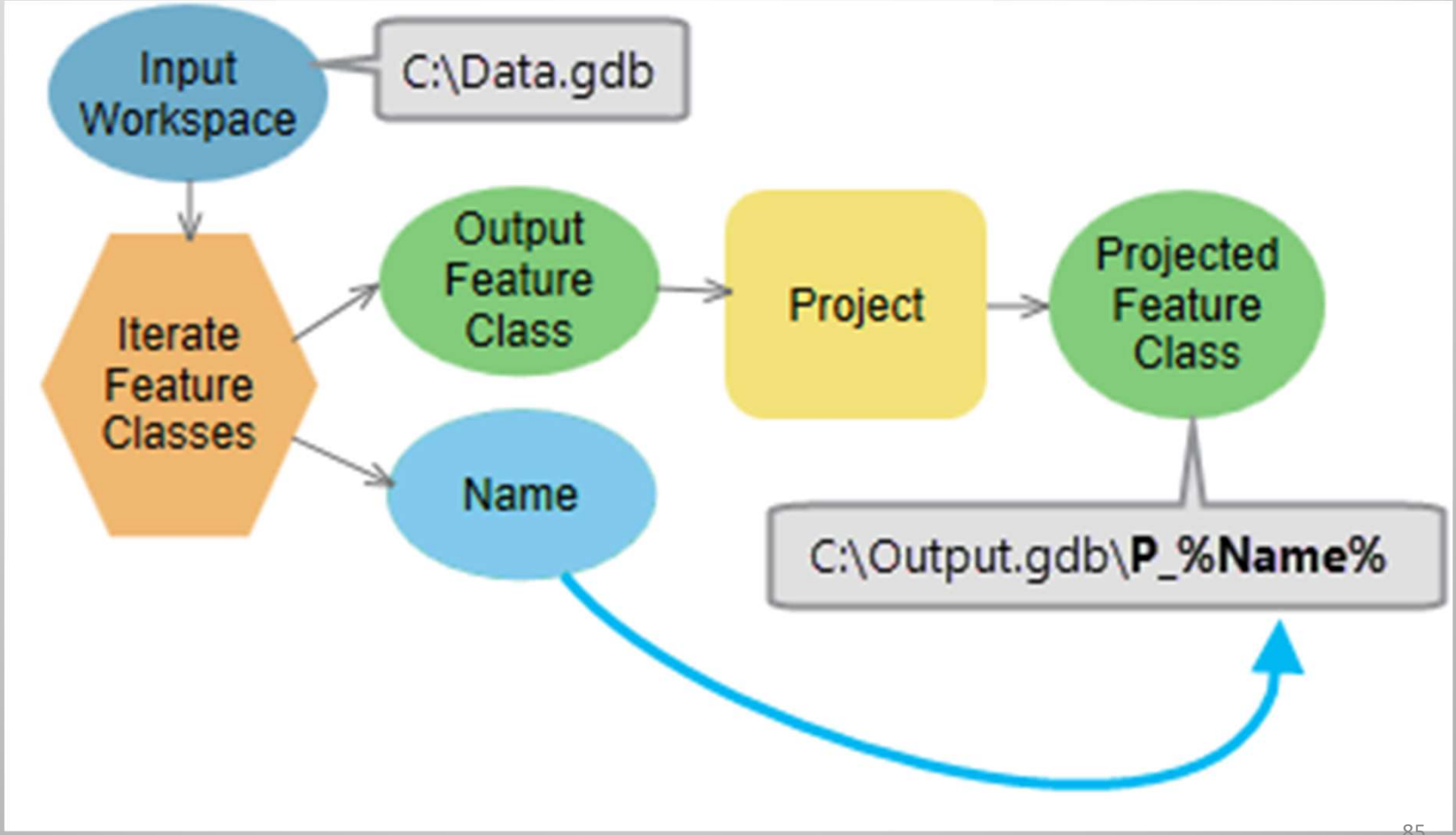
- Iteration = looping = repeat a process over and over
- Iteration in ModelBuilder = Run entire model or a single tool or a set of tools repeatedly
- Tools to iterate in ModelBuilder = Iterators


- Example = Iterate over a list of feature classes and project them

# Without an iterator:

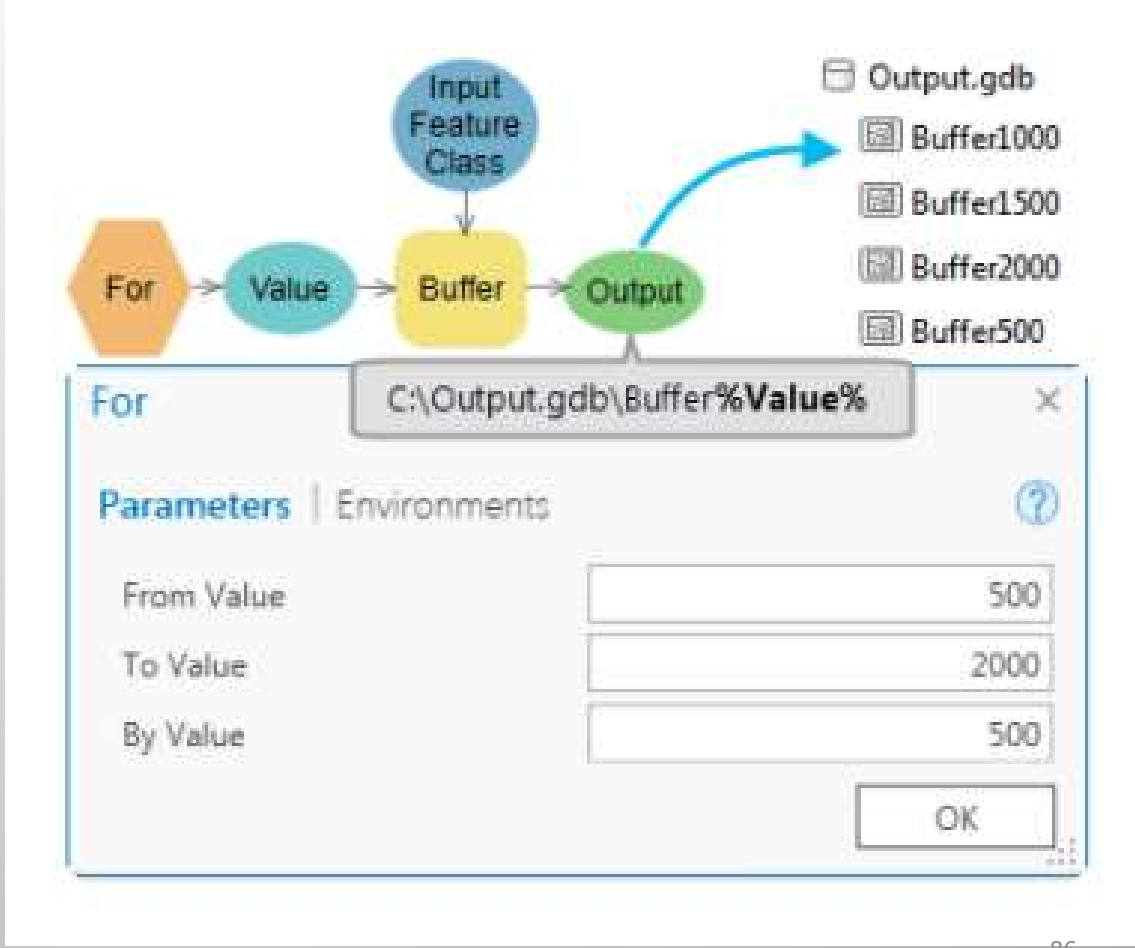


# With an iterator:



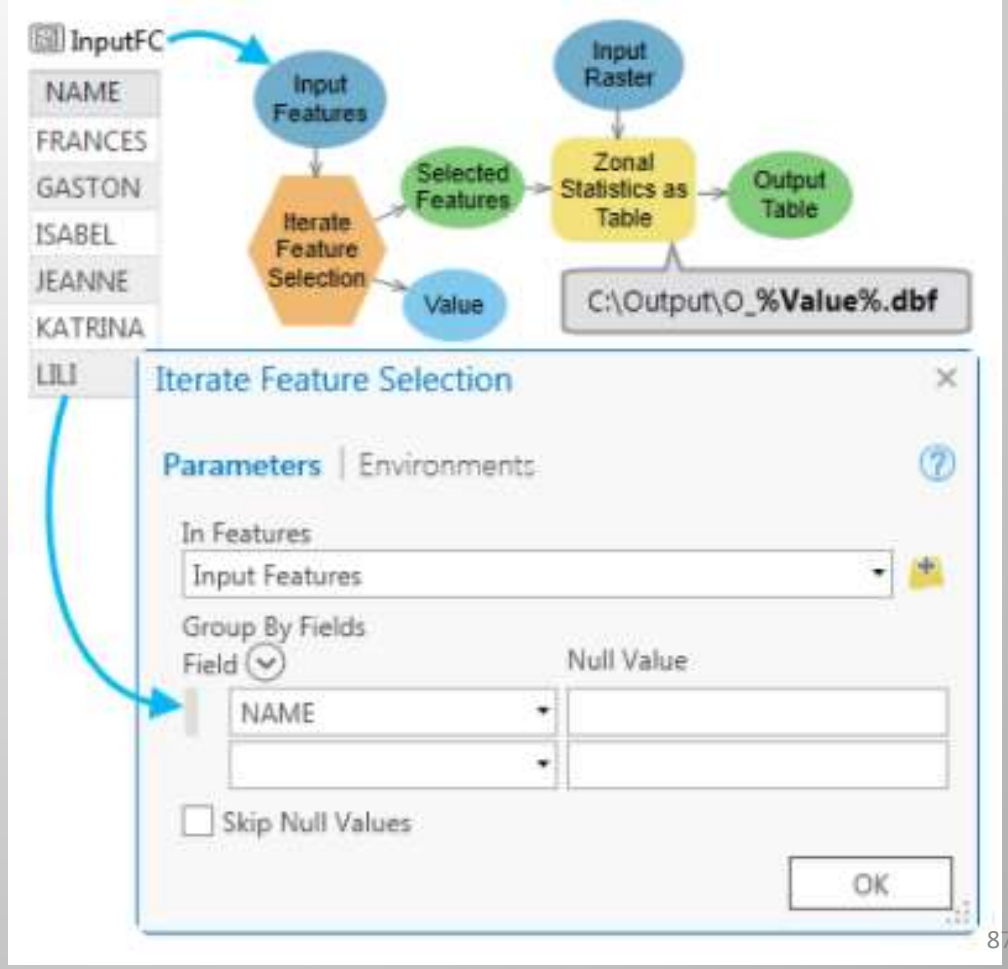
# TYPES OF ITERATORS (FOR):

- Loops or iterates over a set of values from one value to another based on a given interval.
- For example, a buffer with a for value is like saying for value (500 interval) buffer from 500 to 2000.



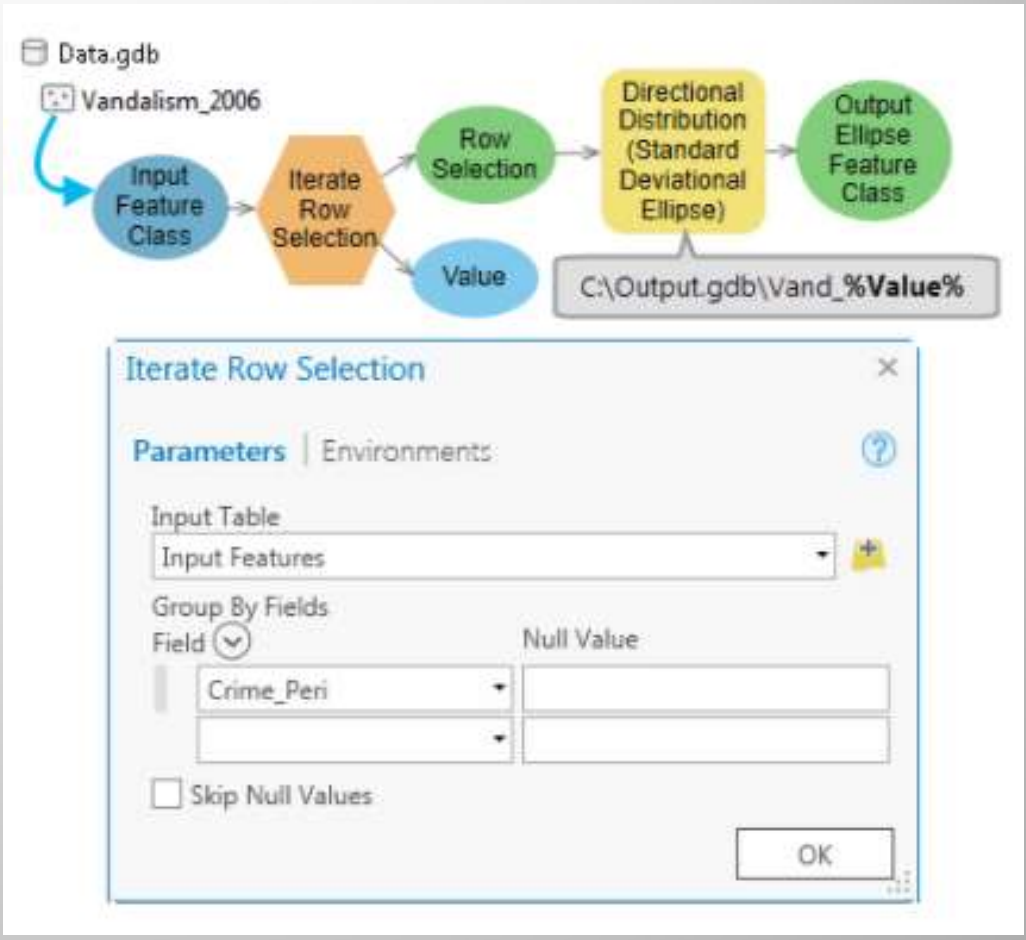
# TYPES OF ITERATORS (ITERATE FEATURE SELECTION):

- By iterating feature selection you can run geoprocessing tools on selected features one at a time
- For example counties iterate each county and buffer by 1 mile.



# TYPES OF ITERATORS (ITERATE ROW SELECTION):

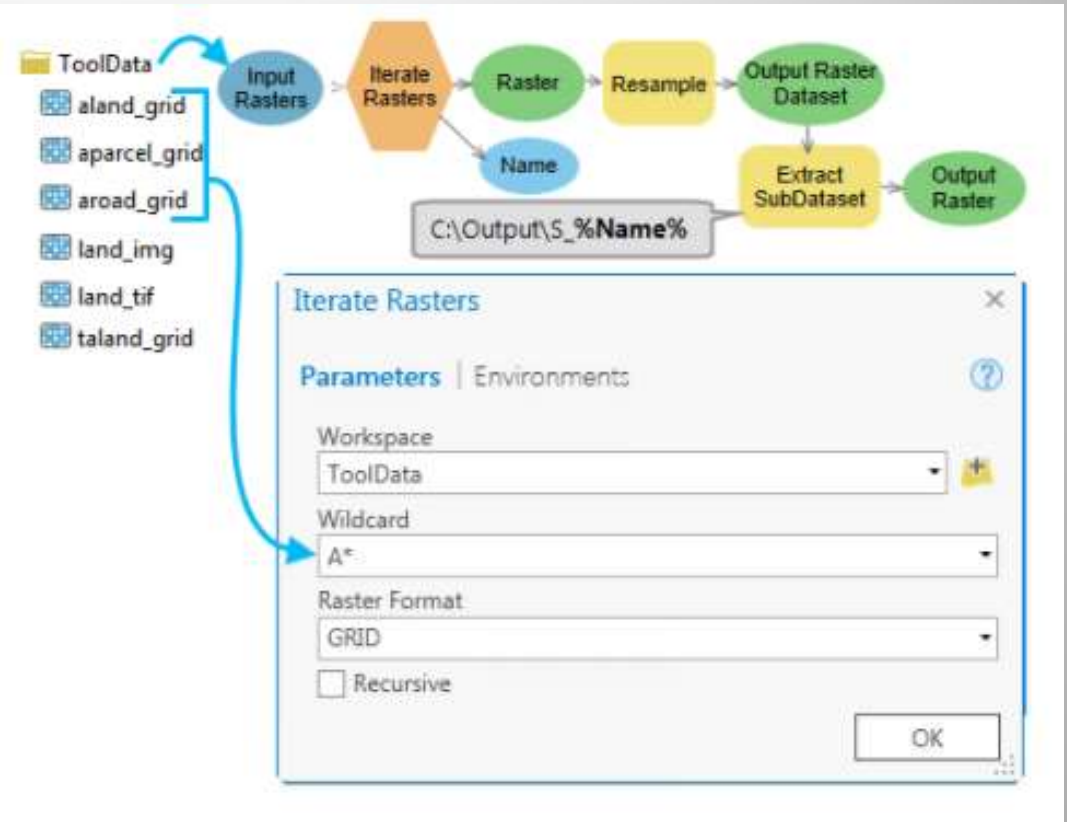
- Similar to Iterate features, but can do with tabular information
- Will discuss later but similar to using a search cursor in Python and ArcPy



# TYPES OF ITERATORS (ITERATE RASTERS):

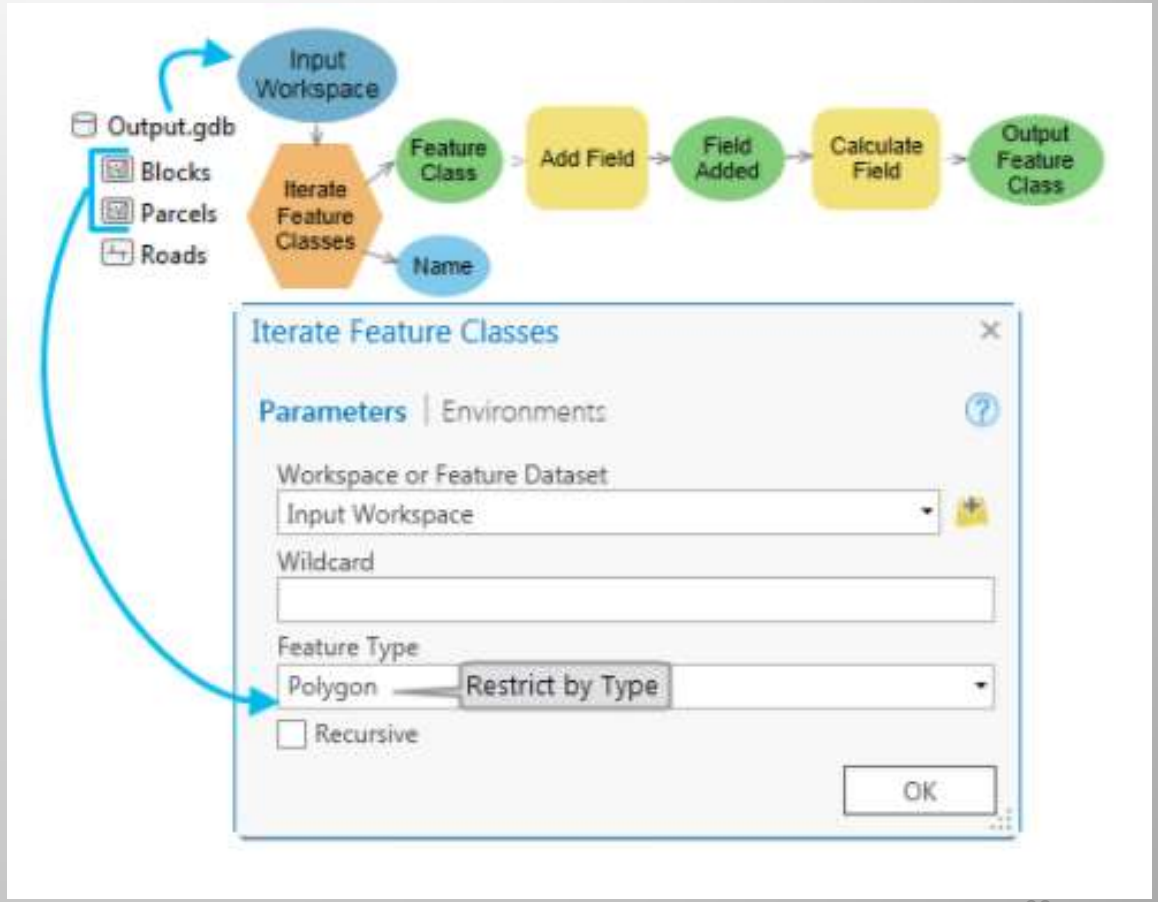
- Iterates over rasters in a given workspace and performs geoprocessing operations on each raster.

- This is like a for loop in Python. You'll learn how to do this soon.



# TYPES OF ITERATORS (ITERATE FEATURE CLASSES):

- Iterates over feature classes in a given workspace.
- Can provide wildcards, for example run only on feature classes that end with buffer (*buffer)
- Can limit it to only loop through workspace based on feature type (i.e. point, line or polygon



Iterate through and select each polygon

Selected County

Selected County Name

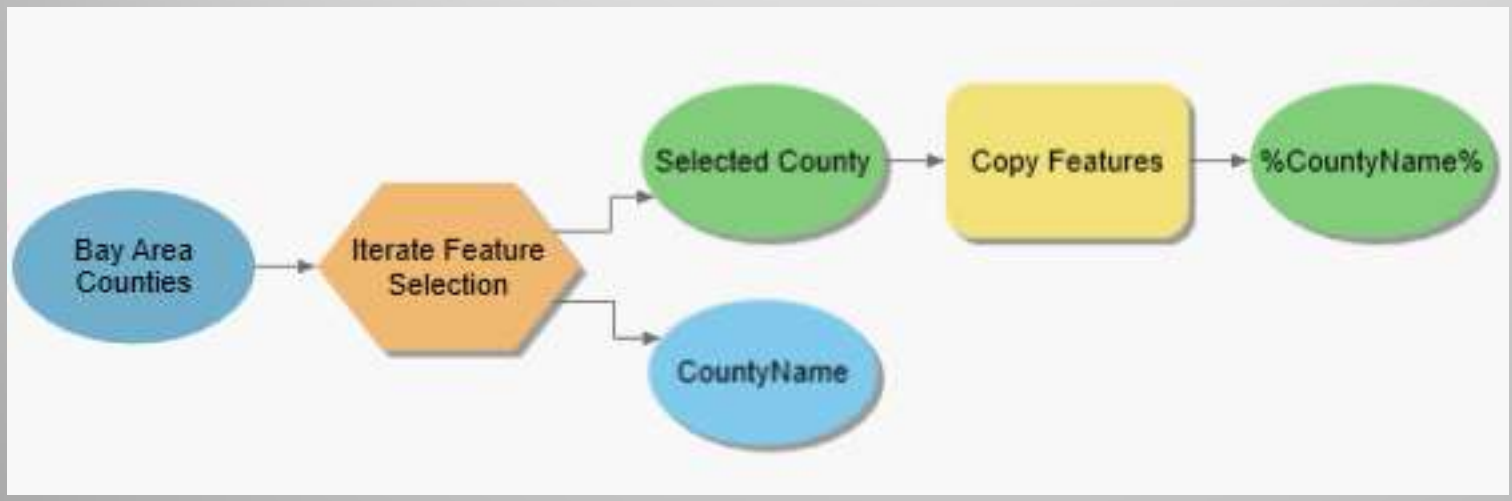
Copy Selected State to new feature class

Alameda

Marin

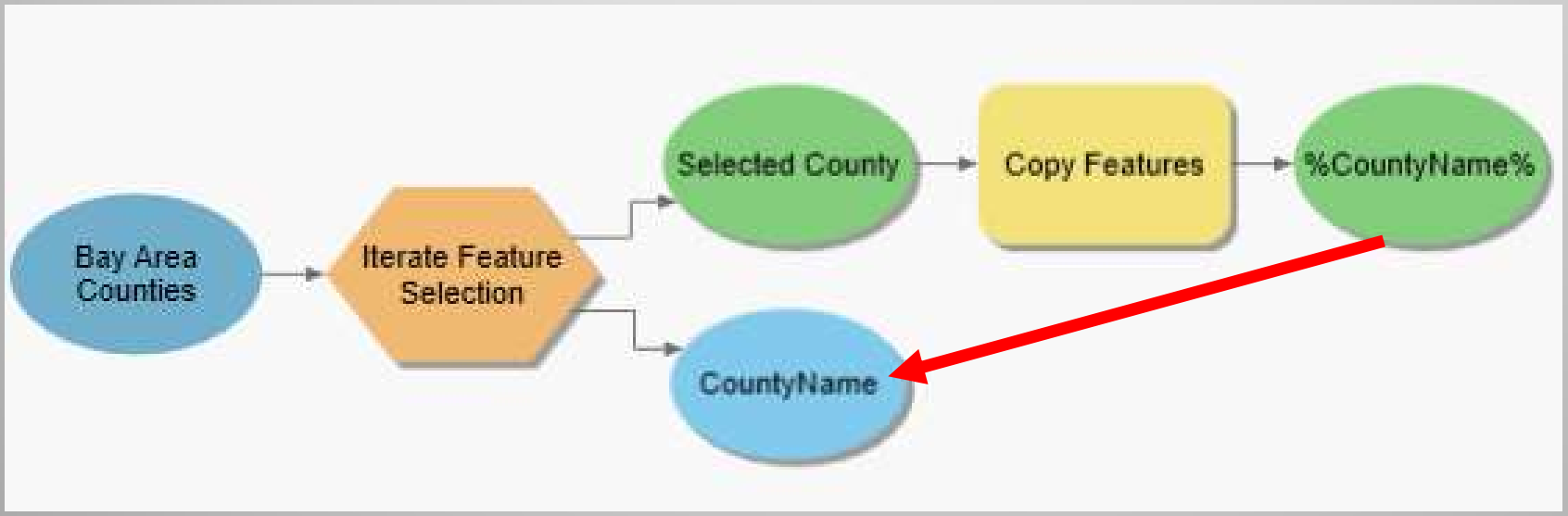
Contra Costa

Solano



# IN-LINE SUBSTITUTIONS

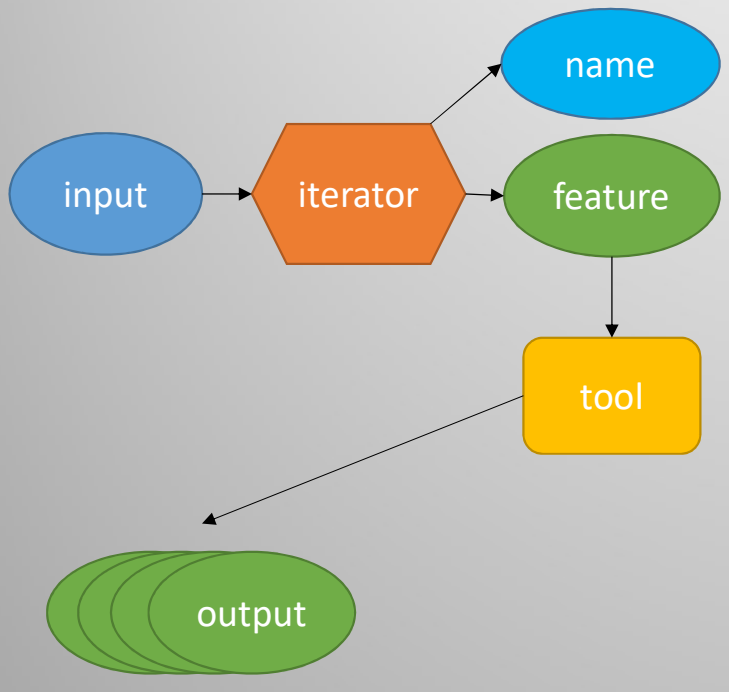
- Allow you to derive names from other elements in the model automatically
- %variable_name% (avoid spaces)



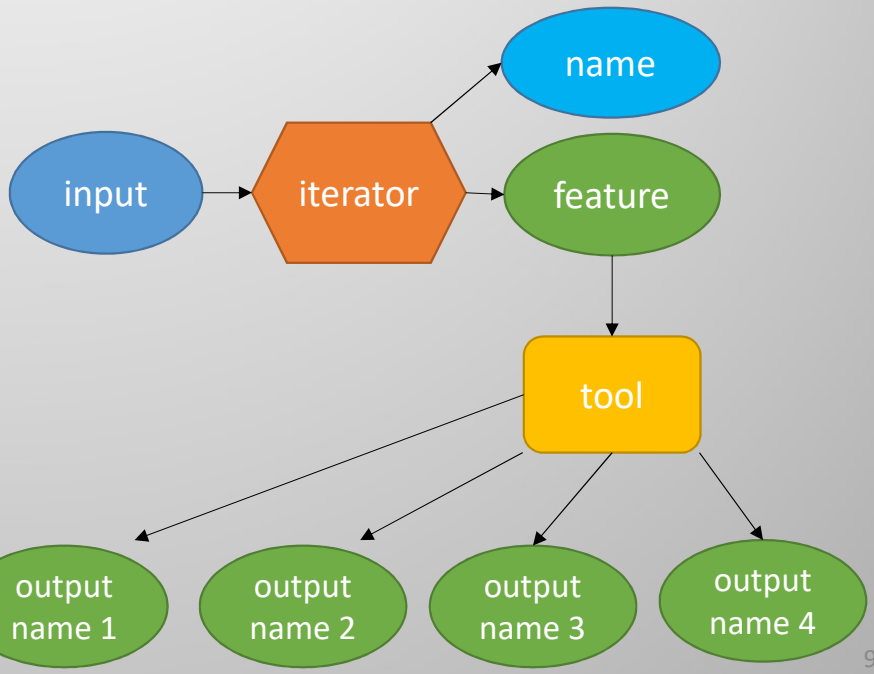
# IN-LINE SUBSTITUTIONS

- Are important to avoid overwriting data!

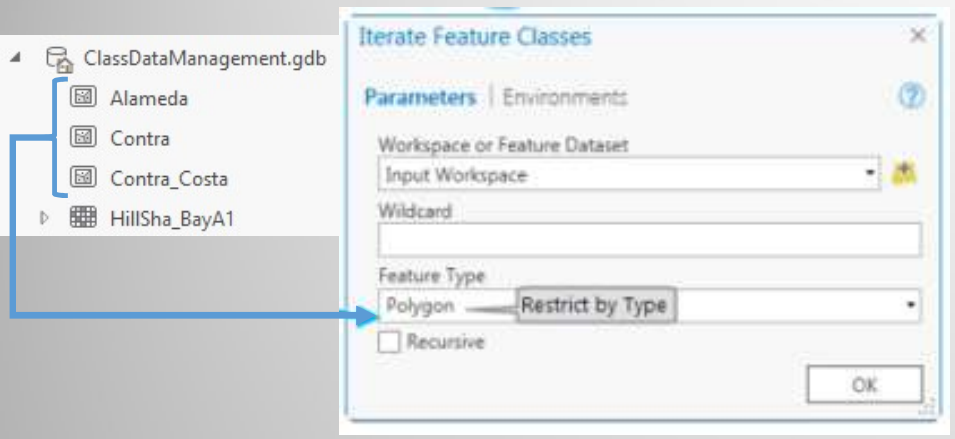
**Without in-line substitutions**

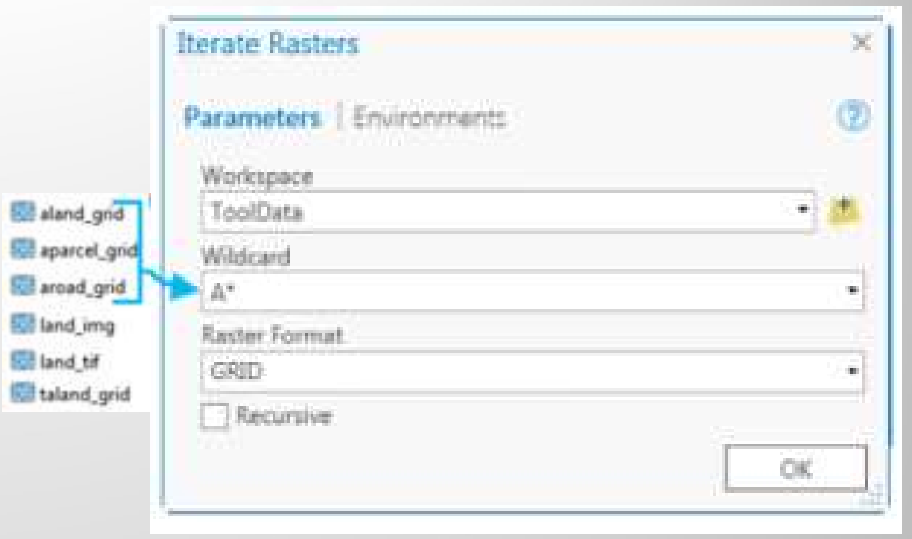



**With in-line substitutions**



# Wildcards and types




- Wildcards return certain items based on the characters in their name

  A* → only items that start with "A"

  *clipped →only items that end with "clipped"

- Type return only items of a specific file type

  JPG returns only files with a .jpeg extension

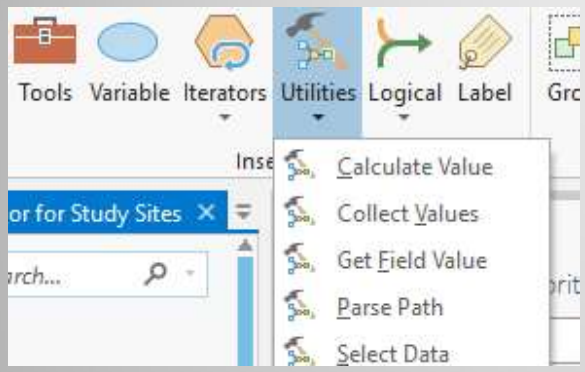
  polygons returns only files have a polygon data type

# Limitations

- You can only have 1 iterator / model
- Inputs must be of the same type (rasters, shapefiles, etc.)
- To use multiple iterators, you must nest models

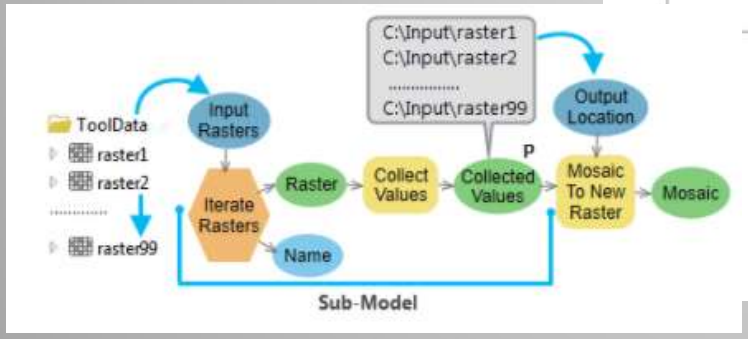
# Model Only Tools: useful for more complex models

## -Collect Values

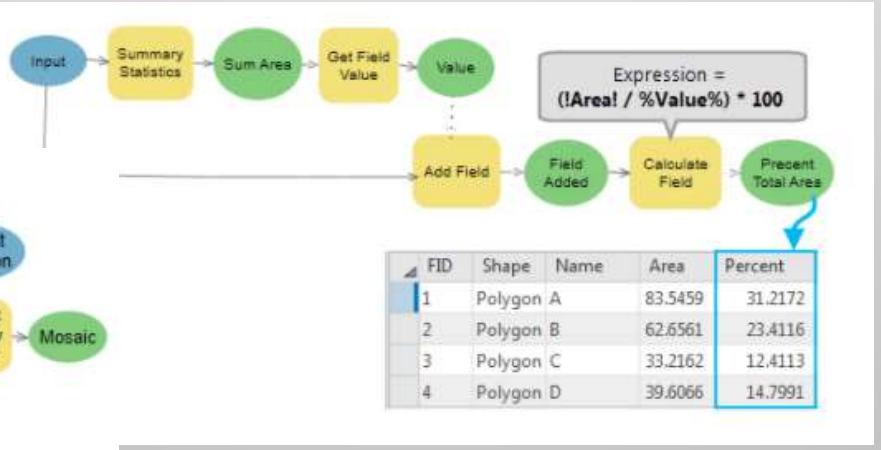




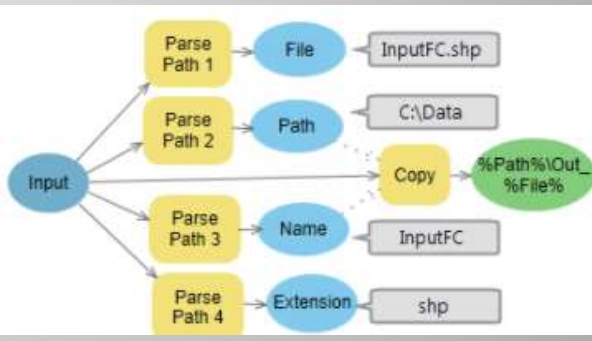
## -Calculate Values




## -Get Field Values

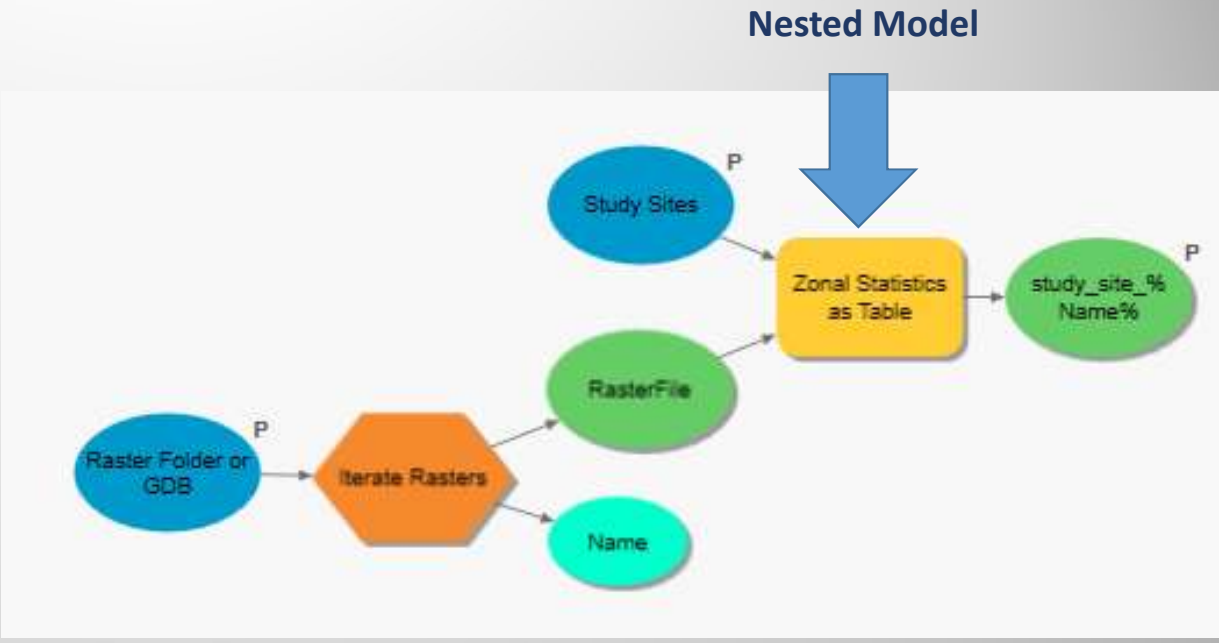



## -Parse path




- SFPD_Incidents.shp
- C:\MB_Data\SFPD_Incidents.shp
- .shp
- SFPD_Incidents

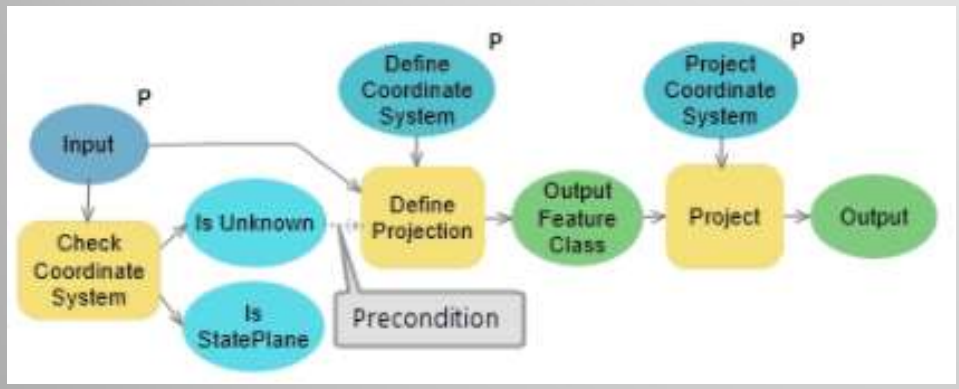
# Nesting models




- Drag and drop an existing model into a new model
- The model being added **must have parameters**

# Branching (if-then-else)

- More efficient—geoprocessing is only carried out if it's appropriate
- But you need to write a script…..




```
import arcpy

# Get the data path and describe its coordinate system
#
data = arcpy.GetParameterAsText(0)
prj = arcpy.Describe(data).spatialReference.name.lower()

# Check if indata is in StatePlane, has no PRJ, or something other than StatePlane
#     Parameter index 1 refers to the "Is Unknown" variable
#     Parameter index 2 refers to the "Is StatePlane" variable
#
if prj.find("_stateplane_") > -1:
    # Set the "Is Unknown" parameter to FALSE, and the "Is StatePlane" parameter to TRUE
    #
    arcpy.SetParameter(1, False)
    arcpy.SetParameter(2, True)
    arcpy.AddMessage("Excellent. Coordinate system is StatePlane.")
elif prj == "unknown":
    # Set the "Is Unknown" parameter to TRUE, and the "Is StatePlane" parameter to FALSE
    #
    arcpy.SetParameter(1, True)
    arcpy.SetParameter(2, False)
    arcpy.AddMessage("To continue you must project your data!")
else:
    # Set the "Is Unknown" parameter to FALSE, and the "Is StatePlane" parameter to FALSE
    #
    arcpy.SetParameter(1, False)
    arcpy.SetParameter(2, False)
    arcpy.AddMessage("Nice try. The coordinate system is neither StatePlane nor unknown.")
```

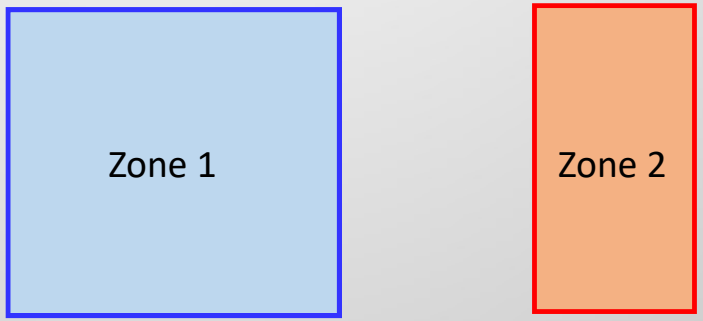
# Exercise 4

- Use 3D Analyst Geoprocessing Tools
- Create a model with a raster iterator to calculate zonal statistics from three rasters

# Zonal Statistics

- Calculates statistics about the raster cells in a particular zone

| | | | |
|---|---|---|---|
| **1** | **1** | **3** | **3** |
| **1** | **1** | **2** | **7** |
| **3** | **5** | **1** | **0** |
| **2** | **6** | **4** | **0** |



# Zonal Statistics

- Calculates statistics about the raster cells in a particular zone

| | | | |
|---|---|---|---|
| 1 | 1 | 3 | 3 |
| 1 | 1 | 2 | 7 |
| 3 | 5 | 1 | 0 |
| 2 | 6 | 4 | 0 |

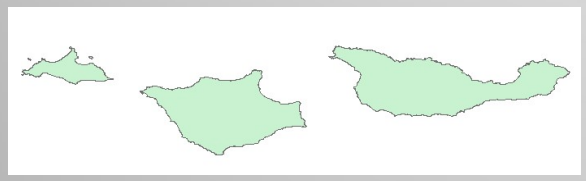
mean = sum of values / # cells

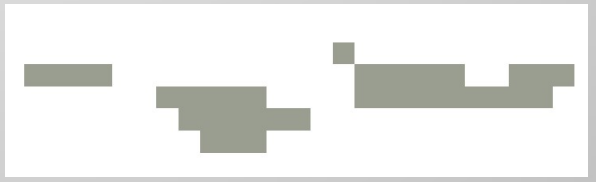
Zone 1 mean = (1+1+1+1)/4 =1

Zone 2 mean = (7+3) / 2 = 5

# Zonal Statistics with polygons

- Polygon is internally converted in to a raster
- Temporary raster then used to create the zones
- Be aware of the pixel size of the raster you wish to summarize and make sure your polygons are sufficiently large





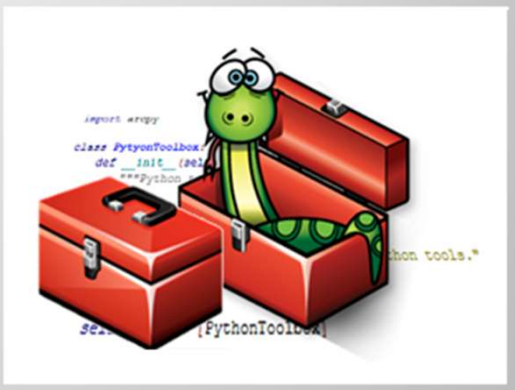
# Exercise 4 Questions

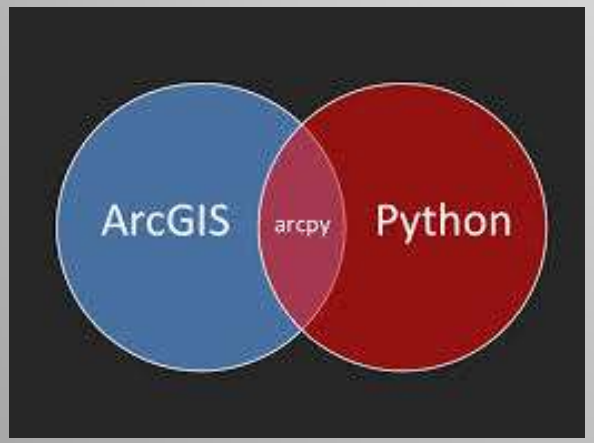
1. Why is it better to user a Raster Iterator than drag and drop the Zonal Statistics as Table tool a bunch of times?

2. Could we add another iterator to the model? [hint, try it out and see what happens]

# DAY 2: Python and Geoprocessing








# AGENDA

**DAY 1**

- Review of ArcGIS Pro
- Geoprocessing Framework
- Tools, Models and Scripts
- Introduction to Model Builder
- Model Parameters and Model Tools
- Advanced Functions in Model Builder

**DAY 2**

- Introduction to Python
- ArcPy and scripting in ArcGIS Pro
- Lists, conditions and loops
- Creating a script tool
- Introduction to the new ArcGIS API for Python

# WHAT IS PYTHON?

1. Object Oriented Programming Language
2. Multi-purpose (Web, Desktop, GUI, Data, Scripting, etc)
3. Open source
4. Interpreted
5. Easier for beginners
6. Focus is on readability and productivity
7. Comes installed with the ArcGIS Platform (ArcGIS Pro & ArcMap)

# ESRI AND PYTHON

1. Scripting language of choice beginning with ArcMap 9.0 (early 2000s)
2. Primary python library for ArcGIS Desktop Apps is called “arcpy”
3. ESRI fully embraces it for :
    - Data Analysis
    - Data Conversion
    - Data Management
    - Map Automation
4. Now have begun developing a Python API for Web GIS

# PYTHON FOR ARCGIS PRO

1. Traditionally ArcGIS Desktop has used Python 2.7, beginning with Pro ESRI moves to Python 3.x
2. Implemented a new method of Python version management called Python Package Management uses an open source manager called “Conda”
3. Interpreted
4. Focus is on readability and productivity
5. Comes installed with the ArcGIS Platform (ArcGIS Pro & ArcMap)

# Section 5

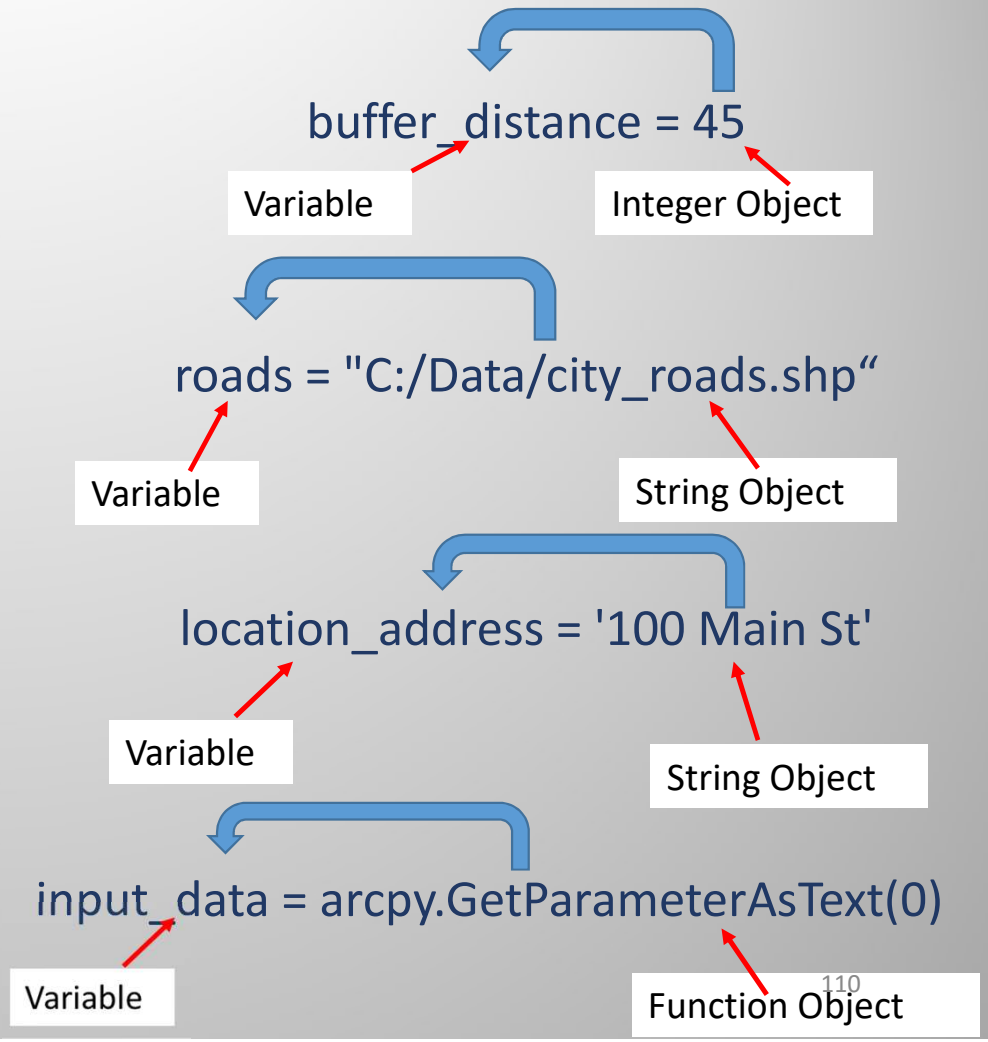
- Python introduction
- A crash course through the basics of Python syntax
- How to write and execute scripts

# PYTHON VARIABLES

- **Variables** are data containers for all data types
- Data is assigned to variables using the equals operator ( = )
- In Python variables can be reassigned within the same script

```
>>> num = 3
3
>>> num = 6
6
```

- Local vs. Global variables (local is by function global is the entire script)



# PYTHON VARIABLE RULES

- Variables must be discrete and have no spaces (use underscores to space for readability)
- Variable must not start with a number or a symbol (except an underscore).

```
>>> 1demo = "a string"
SyntaxError: invalid syntax
>>> &demo = "a test"
SyntaxError: invalid syntax
>>>
```

- Variables can start with an underscore (and sometimes a double underscore is used for “private” functions).

```
>>> _demo = "this works“
```

- Good practice to make them explanatory

# Why use variables?

- Easier to interpret than literal values
- Avoids retyping long strings
- Reusable

Example:

With variables:

```
ca_schools = "D:/MB_Data/Lab1/Cal_Public_Schools.shp"
ba_counties ="D:/MB_Data/Lab1/ba_counties.shp"
schools_clipped = "D:/MB_Data/Lab1/Schools_Clip_Script.shp"
arcpy.Clip_analysis(ca_schools,ba_counties,schools_clipped)
```

Without variables:

```
arcpy.Clip_analysis("D:/MB_Data/Lab1/Cal_Public_Schools.shp","D:/MB_Data/Lab1/ba_counties.shp","D:/MB_Data/Lab1/Schools_Clip_Script.shp")
```

# PYTHON DATA TYPES

- Number (integer and float) -- immutable
- String -- immutable
- List -- mutable
- Tuple -- immutable
- Dictionary -- mutable

- Strings, lists and tuples are sequences
- **Strings, numbers and tuples are *immutable***
- **List and dictionaries are *mutable***

# PYTHON DATA TYPES

- Strings
  - quote = 'This string contains a quote: "Here is the quote" ‘
  - Single quote and double quote both work (must be consistent), for quoting inside strings use double and single
- Integers
  - >>> 4/2

  **2**
- Floats
  - >>>5/2

  **2.5**
- Lists
  - >>>weekend = [‘Saturday’ , ’Sunday’]
  - >>>weekend[0]

  ‘Saturday’
- Tuples
  - >>> atuple = ('e','d','k')
  - >>> atuple[0]

  'e‘
- Dictionary
  - >>> new_dictionary = {}
  - >>> new_dictionary ['key'] = 'value'
  - >>> new_dictionary

  {'key': 'value'}
  - >>> adictionary = {'key':'value'}
  - >>> adictionary ['key']

  'value‘

  >>>ages_dictionary = {‘bob’:44, ‘susan’:35, ‘Kelly’ : 40}

  >>>ages_dictionary[‘bob’]

  44

# PYTHON CONCEPTS: KEYWORDS

- There are a number of keywords built into Python that should be avoided when naming variables.

| | | | | |
|---|---|---|---|---|
| • max | • tuple | • while | • elif | • class |
| • min | • def | • in | • or | • try |
| • sum | • del | • as | • None | • with |
| • return | • from | • if | • True | • except |
| • list | • not | • else | • False | • break |

# PYTHON CONCEPTS: ZERO-BASED-INDEXING

- Python indexing and counting always begins with 0 instead of 1.
- Can also be accessed using negative numbers, -1 for the last member of the dataset.
- Used a lot for lists, but work for other data types too
- layer_List = [“roads”, “streams”, ”soil”]

**Indexes**

| 0 | 1 | 2 |
|---|---|---|

```
>>>layer_List[0]
“roads”
>>>layer_List[2]
“soil”
>>>layer_List[-1]
“soil”
```

**Slicing Indexes**

```
>>>layer_List[:2]
[‘roads’, ’streams’]
>>>layer_List[1:]
[‘streams’, ‘soil’]
>>>layer_List[1:2]
[‘streams’]
```

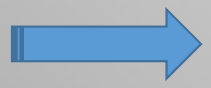
*Up to position 2, but not including*

*Beginning with and including position 1*

*Beginning with and including position 1 and up to position 2, but not including position 2*

# PYTHON CONCEPTS: INDENTATION

- Indentation matters in Python.
- Tabs vs Spaces, should be consistent when writing code. Don't mix tabs and spaces!
- One indentation is generally 4 spaces or 1 tab.
- Must indent on a line following a colon ( : )
  - *>>>for item in items:*

Indent  *print("I indented")*

# PYTHON CONCEPTS: STRING FORMATTING

- Python's str.format() method of the string class allows you to do variable substitutions and value formatting. This lets you concatenate elements together within a string through positional formatting.
- A good example in GIS for string for formatting is when using SQL statements.

```
>>> list_names = ['bob', 'susan', 'kevin', 'kelly']
>>> for name in list_names:
...     print("Let me introduce {0}".format(name))
...
Let me introduce bob
Let me introduce susan
Let me introduce kevin
Let me introduce kelly
>>>
```

# PYTHON CONCEPTS: COMMENTS

- In python using “#” will allow you to create a comment for a single line of code
- Multi-line comments enclosed in 3 single quotes on both start and end lines of comment
- Also useful to comment out lines when debugging
- A good practice to write comments for code readability and sharing

Single-line Comment

```
4 #define variables
5 ca_schools = "D:/MB_Data/Lab1/Cal_Public_Schools.shp"
6 ba_counties ="D:/MB_Data/Lab1/ba_counties.shp"
7 schools_clipped = "D:/MB_Data/Lab1/Schools_Clip_Script.shp"
```

Multi-line Comment

```
19 '''
20 Multi line
21 comments
22 are done
23 this way
24 '''
```

# PYTHON CONCEPTS: BACKSLASH (\) ESCAPE

- When defining a file location in windows you can't use the typical path from explorer
  "C:\MyFolder\MyData\rivers.shp "

- Python interprets the "\" as special escape character.

- There are 3 options to resolve this in Python:
  1. **"C:/MyFolder/MyData/rivers.shp "** use a forward slash
  2. **"C:\\MyFolder\\MyData\\rivers.shp"** use double black slashes
  3. **r "C:\MyFolder\MyData\rivers.shp "** put an **r** in front, this is a raw string ***

# PYTHON CONCEPTS: OPERATORS

- **Modulo ( % )**
  - produces the remainder value of a division value.

```
>>> 6 % 5
1
```

- **Addition ( + )**
  - adds integers and floating values together. Can also be used in string formatting to “add” two strings together.

```
>>> 4+5
9
```

- **Subtraction ( - )**
  - is straight forward and relies on the dash symbol used in mathematics. Cannot be used
- **Division ( / )**
  - Division uses the forward slash symbol:

```
>>> 2/5
2.5
```

- **Exponent (** )**
  - Performs exponential (power) calculation on operators and assign value to the left operand

```
>>> 3**2
9
```

- **Multiplication ( * )**
  - Multiplying is used in both mathematical operations and for string operations.

```
>>> 2*12
24
```

- **Equals ( == )**
  - the double equals sign is employed to evaluate if two values are equal. The single equals sign is used for assignment the two can be confusing.

```
>>> 5 == 2
False
```

- **Does not equal ( != )**
  - an exclamation point and an equals sign ( != ) to determine if two values are not equal:

```
>>> 5 != 2
True
```

- **Less than or equals ( < ) or ( <= ) and Greater than or equals ( > ) or ( >= )**

```
>>> 5 >= 2
True
>>> 5 <= 2
False
```

# PYTHON CONCEPTS: LOOPS

- Two types of loops: **for loops** and **while loops**
- A geospatial example would be searching through a set of parcels to buffer them individually, to then find all of the parcel's neighbors using the intersect or touches methods.

**For loops** are used to perform an action on each member of an ordered set, or iterable, or a specified subset of the ordered set.

```
In [1]: aList = [23, 'a', 3, 'hi']

In [2]: for aVal in aList:
   ...:     print(aVal*2)
   ...:
46
aa
6
hihi
```

**While loops** are used to create a continuous action that will only stop if an error is encountered or if a condition is met. For instance, the example below will loop for ten loops before it stops:

```
In [5]: x = 0

In [6]: while x < 10:
   ...:     print(x)
   ...:     x = x + 1
   ...:
0
1
2
3
4
5
6
7
8
9
```

# PYTHON CONCEPTS: CONDITIONALS

Conditionals are used to sort data in loops to identify specific pieces of data that should be operated on. There are three main types of conditionals used in loops: **If, Elif,** and **Else**.

**If** conditional, by using "if" to identify a situation, the code can differentiate within the members of the dataset being evaluated.

```
In [9]: data = [2,5,7,9]

In [10]: for num in data:
    ...:     if num == 7:
    ...:         print(num**2)
    ...:
49
```

**Elif** conditional, is shorthand for "else if" and it is used to create a second (or Nth - there is no limit on the number of elifs allowed) conditional.

```
In [12]: data = [2,5,7,9]

In [13]: for num in data:
    ...:     if num == 7:
    ...:         print(num**2)
    ...:     elif num > 8:
    ...:         print(num, num**2)
    ...:
49
9 81
```

**Else** conditional, allows the programmer to perform an action on all members of the dataset being evaluated that do not match other conditions listed.

```
In [14]: data = [2,5,7,9]

In [15]: for num in data:
    ...:     if num == 7:
    ...:         print(num**2)
    ...:     elif num > 8:
    ...:         print(num, num**2)
    ...:     else:
    ...:         print("This is the condition")
    ...:
This is the condition
This is the condition
49
9 81
```

# PYTHON CONCEPTS: FUNCTIONS

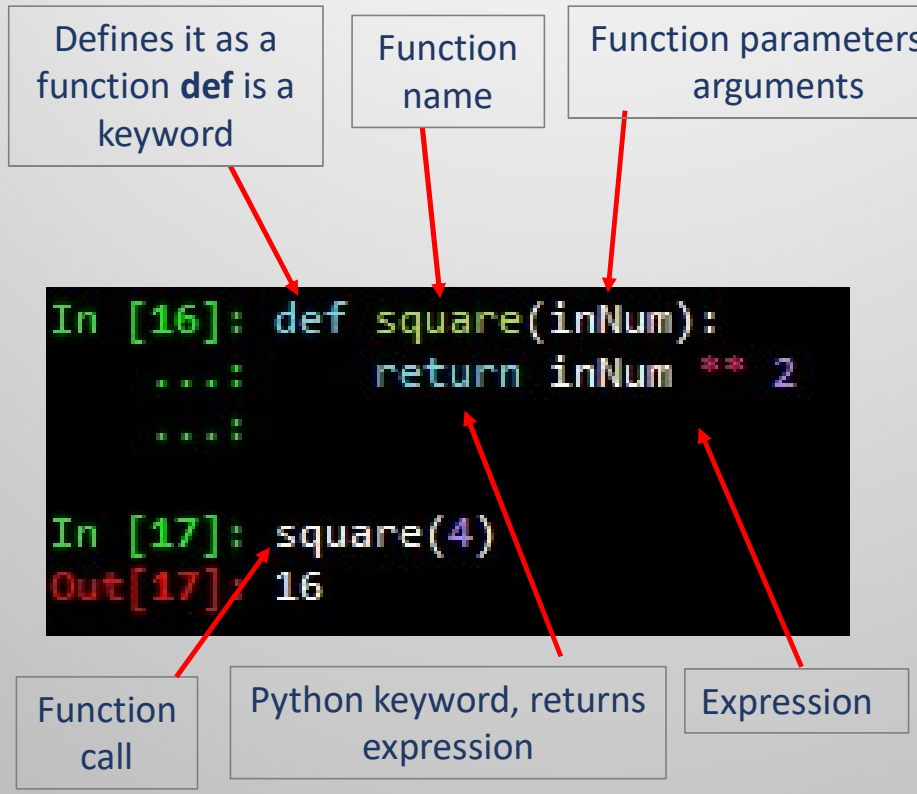
- Functions are defined in scripts ( and modules, or groups of scripts combined for a common objective) using the keyword **def**.
- Makes for efficient easily reproducible code
- They are blocks of code that usually perform one or many related tasks that can be used more than once.
- Functions are usually at the beginning of scripts.
- A common example is one we've already used, **print()** function.

# PYTHON CONCEPTS: FUNCTIONS Continued…

- Using a function is referred to as calling a function
- Additional functions can be accessed using modules

*<function>(<arguments>)*

```
>> pow(2,3)
8
```



# PYTHON CONCEPTS: METHODS

- A method is a function that is closely coupled to some object

**<object>.<method>(<arguments>)**

```
>> > location = “San Francisco State University”
>>> location.count(“a”)
3
```

- In this above example count is the **method**. A lot of Python’s data types have methods

# PYTHON CONCEPTS: MODULES & IMPORT

- **Module** or **Library** is external code brought into your script to use as "tools" or "methods". Makes writing code much cleaner and easier.
- To access **Modules** we use "**import**" statements
- Import statements are usually at the very beginning of your script
- Two ways to import Modules into your script:

```
In [26]: import math

In [27]: math.pi
Out[27]: 3.141592653589793
```

```
In [30]: from math import sqrt

In [31]: sqrt(16)
Out[31]: 4.0
```

# Exercise 5

- Get familiar with basics of Python syntax
- Using an IDE, spyder and running code in that environment
- Editing an existing script and executing it

# Exercise 5 Questions

1. What data type are all of the variables in script 5-1.py, integer, float, or string?

2. What's the difference between the interpreter and editor windows?

3. Why should you use an integrated development environment?

# Section 6

- Python in ArcGIS Pro and ArcPy
- Tips for writing a good script
- Using tools in the ArcPy module and how to get their syntax
- Debugging errors

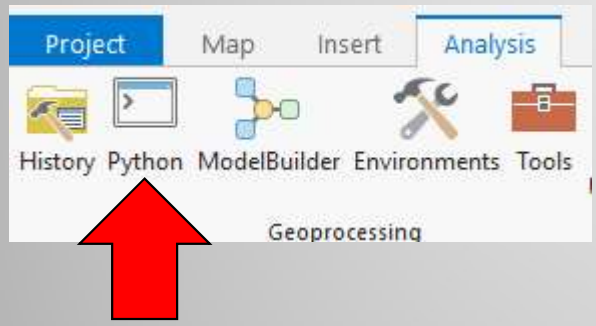
# ARCPY

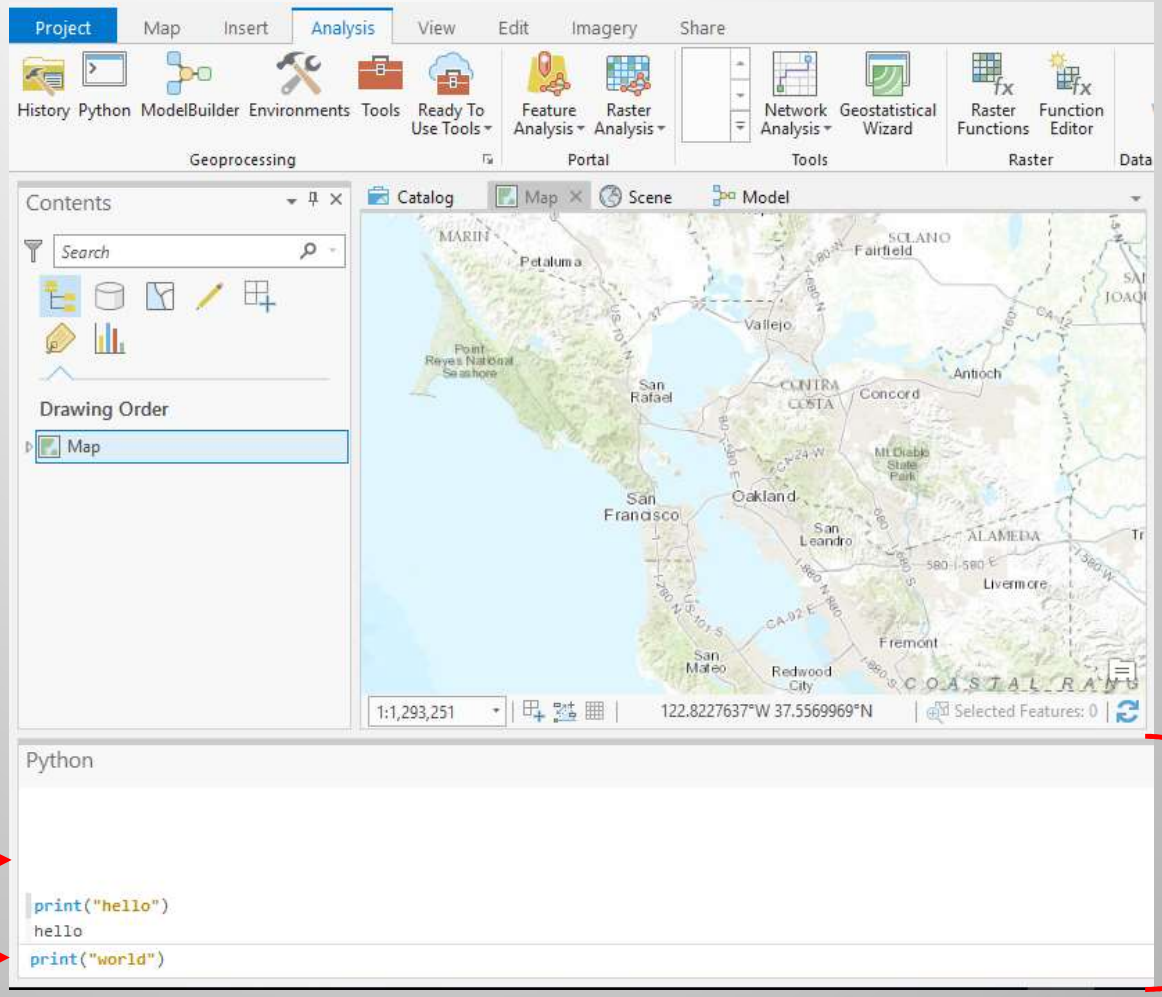
- The primary desktop site package from ESRI
- *Import arcpy*
- Used to access ArcGIS Pro Desktop Geoprocessing tools
- Syntax usually follows this standard:
  arcpy.*geoprocessingtool*(*parameter1,parameter2,parameter3*)

```
#import modules
import arcpy

#define variables
ca_schools = "D:/MB_Data/Lab1/Cal_Public_Schools.shp"
ba_counties ="D:/MB_Data/Lab1/ba_counties.shp"
schools_clipped = "D:/MB_Data/Lab1/Schools_Clip_Script.shp"

#delete the file if it already exists
if arcpy.Exists(schools_clipped):
    arcpy.Delete_management(schools_clipped)

#clip the the data
#syntax is arcpy.Clip_analysis(input features, clip features, output data)
arcpy.Clip_analysis(ca_schools,ba_counties,schools_clipped)

print("Script complete")
```

# PYTHON IN ArcGIS PRO

- Analysis Tab -> Python Window




Transcript

Python Prompt

Python Window

# PYTHON WINDOW IN ArcGIS PRO

- Works with current map document
- Interactive interpreter:
  - Executes code directly line-by-line
- Good for testing short code
- Code can be saved
- No error checking / debugging

# PYTHON IN ArcGIS PRO

- Python 3.x as opposed to 2.7 for ArcGIS Desktop
- Python environment is located here:

*C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3*

- Python for ArcGIS Pro is installed in a virtual environment created using the Python package manager called **Conda**
- The environment for Python in ArcGIS Pro is named "*arcgispro-py3*"

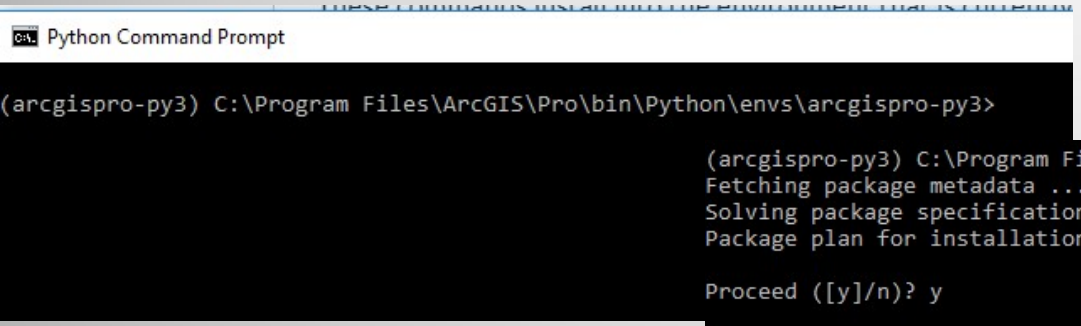

(Python Package Management)

- Python package manager, allows you to have multiple versions of Python installed.
- Helps solve the problem of having two versions of Python for ESRI Desktop platform. ArcGIS Desktop (Python 2.7) vs. ArcGIS Pro (Python 3.6)
- Easy to install Modules: "conda install *modulename*"
  - *Example: conda install spyder*
- Can create separate environments so you don't corrupt your ArcGIS Pro Python environment.

# CONDA Continued

## (Python Package Management)

Start Menu -> ArcGIS -> **Python Command Prompt**




***conda create –n envname***

```
(arcgispro-py3) C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3>conda create -n daraexample
Fetching package metadata .............
Solving package specifications:
Package plan for installation in environment C:\Program Files\ArcGIS\Pro\bin\Python\envs\daraexample:

Proceed ([y]/n)? y

#
# To activate this environment, use:
# > activate daraexample
#
# To deactivate this environment, use:
# > deactivate daraexample
#
# * for power-users using bash, you must source
#
```

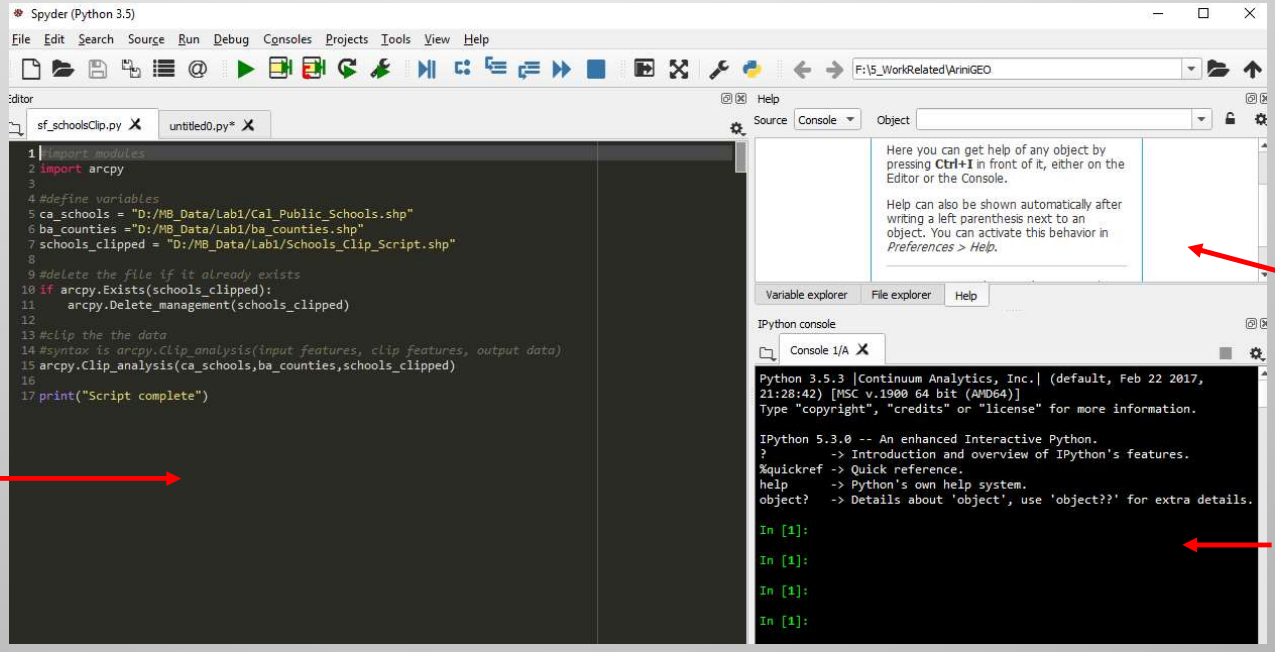
Note** to use these tools you must be an administrator or “run as administrator” on the computer. We will not be doing this in labs today, but feel free to try this at home

***activate envname***

```
(arcgispro-py3) C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3>activate daraexample

(daraexample) C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3>
```

# INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

- An advanced text editor with powerful functionality
- Tools for debugging and writing complex code
- Spyder




Code Editor Pane

Variable/File explorer/Help

IPython Console

# PLAN YOUR WORKFLOW

- Document who, what, why, when, etc.
- Create a pseudo-code outline to be filled-in
  - Import needed modules
  - Set environment settings
  - Define variables
  - Describe the steps and tools you will use

```
# -*- coding: utf-8 -*-
"""
Created on Sun Sep  3 16:40:55 2017

@author: Dara
"""

#import Modules


#set environment settings


#assign input variables


#assign output variables


#for each feature class in the geodatabase clip to the study area


#export to a new folder
```

# INCLUDE COMMENTS

- Anything following a hashtag (#) symbol will not be executed
- For multiple lines, use “”” to begin and end
- Helps other users to understand your thought process, and you to remember your thought process!

```
13 #nothing on this line will run during the script after the '#'
```

```
 7 """
 8 Using the 3 quotes for
 9 Multi-line comments
10 saves you from putting
11 a # before every line
12 """
```

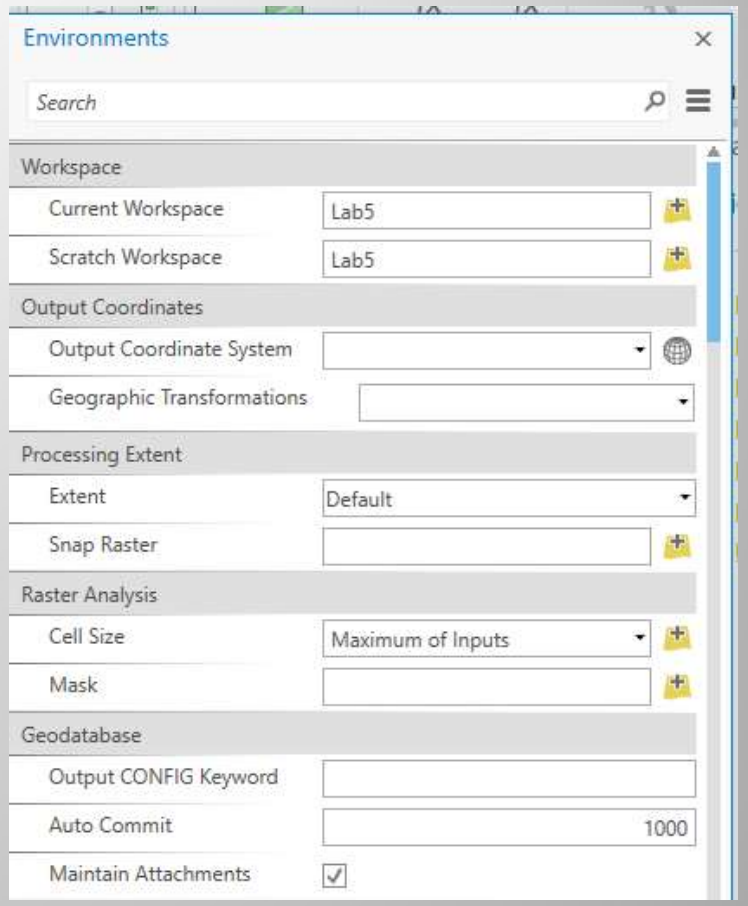
# ENVIROMENT SETTINGS

- Set the workspace

```
arcpy.env.scratchWorkspace =
r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data
\Lab2\TransitSchools.gdb"
```

- Set the overwrite settings to True

```
arcpy.env.overwiteOutput = True
```



# SYNTAX?

```
import arcpy, math
from functions_useful import find_key

arcpy.env.workspace = r"Z:\projects\current\water\Potable_Water\gisdata\derivates\Deliverable\Updated\Script\PotableWater2011.gdb\WaterDistribution"

def CalculaAzimuth(linea):
    if (hasattr(linea,'type') and linea.type == 'polyline'):
        xf = linea.firstPoint.X
        yf = linea.firstPoint.Y
        xl = linea.lastPoint.X
        yl = linea.lastPoint.Y
        dX = xl - xf
        dY = yl - yf
        PI = math.pi
        Azimuth = 0 #Default case, dX = 0 and dY >= 0
        if dX > 0:
            Azimuth = 90 - math.atan( dY / dX ) * 180 / PI
        elif dX < 0:
            Azimuth = 270 - math.atan( dY / dX )* 180 / PI
        elif dY < 0:
            Azimuth = 180
        return Azimuth
    else:
        return False

reducer = "wFitting"

lineFC = "Main_Single"

arcpy.MakeFeatureLayer_management(lineFC, 'mains_lyr')
main = 'mains_lyr'

arcpy.MakeFeatureLayer_management(reducer, 'reducer_lyr')
reduc = 'reducer_lyr'

#sql = "OBJECTID = 5772"
#sql = "OBJECTID > 33900 AND OBJECTID < 6526"
sql = "FITTINGTYPE = 'Reducer' OR FITTINGTYPE = 'Reducing Cross' OR FITTINGTYPE = 'Reducing Tee'"

with arcpy.da.SearchCursor(reducer, ['OID@', 'SHAPE@XY', 'SHAPE@', 'Dir', 'SHAPE@X', 'SHAPE@Y'], sql) as cursor:
    for row in cursor:
        objectID = row[0]
        xy = row[1]
        point = row[2]
```

# SYNTAX IN TOOL HELP

http://pro.arcgis.com/en/pro-app/tool-reference/analysis/clip.htm

## Syntax

```
Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})
```

| Parameter | Explanation | Data Type |
|---|---|---|
| in_features | The features to be clipped. | Feature Layer |
| clip_features | The features used to clip the input features. | Feature Layer |
| out_feature_class | The feature class to be created. | Feature Class |
| cluster_tolerance (Optional) | The minimum distance separating all feature coordinates as well as the distance a coordinate can move in X or Y (or both). Set the value to be higher for data with less coordinate accuracy and lower for data with extremely high accuracy. | Linear unit |

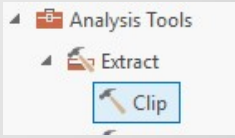
## Code sample

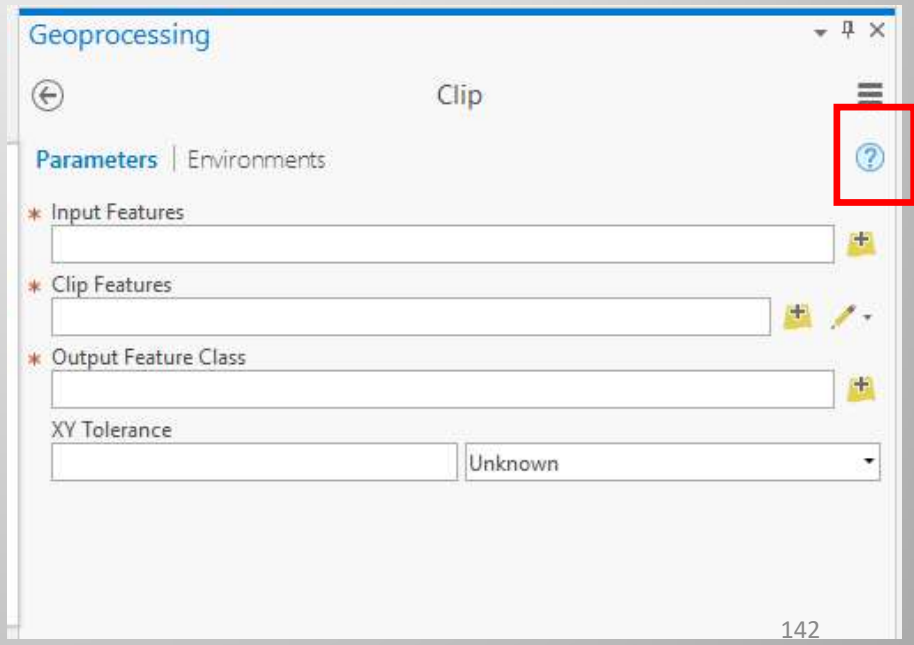
Clip example (Python window)

The following Python window script demonstrates how to use the Clip function in immediate mode.

```
import arcpy
from arcpy import env

env.workspace = "C:/data"
arcpy.Clip_analysis("majorrds.shp", "study_quads.shp", "C:/output/studyaroa.shp")
```

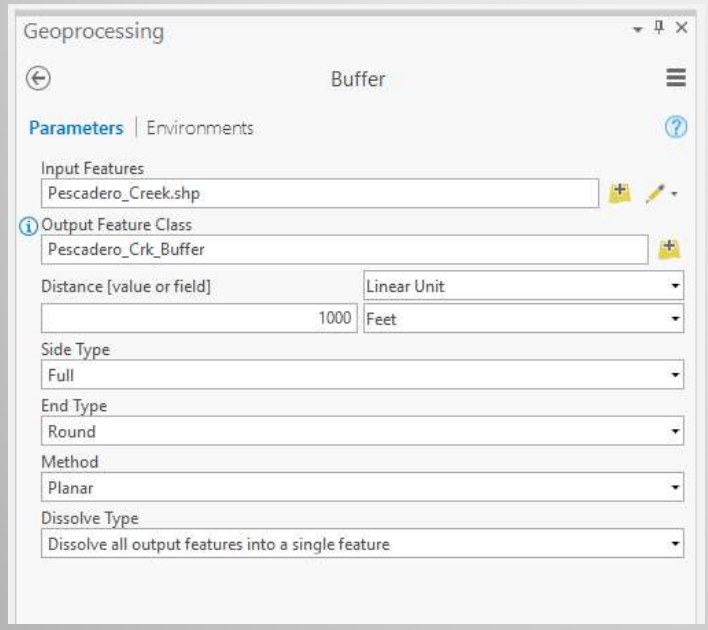
Online Tool Help all provides Python Syntax



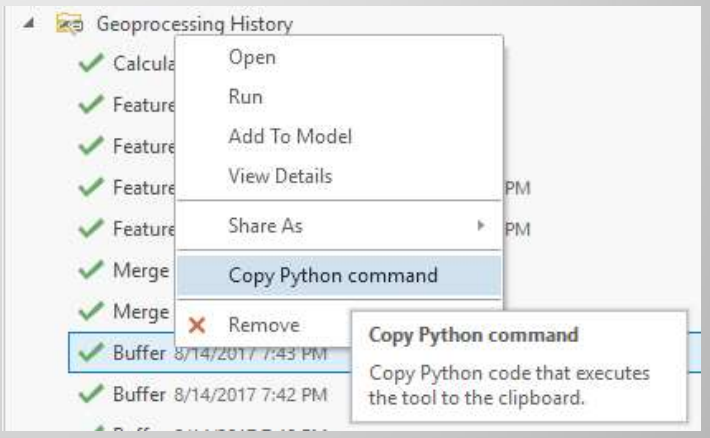



# USING GEOPROCESSING RESULTS FOR SNIPPET

Run a tool in ArcGIS Pro




Open Catalog Pane >>>
Geoprocessing History



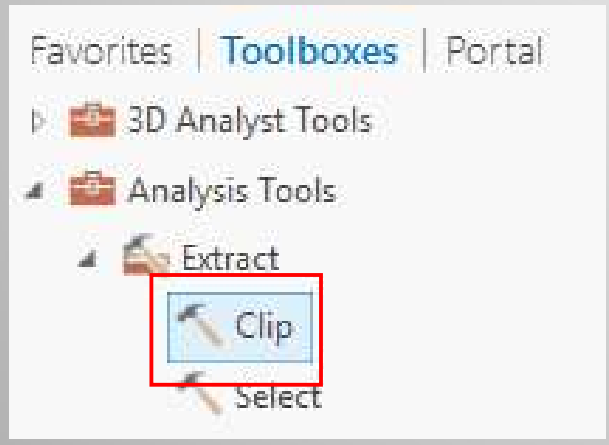

Python Snippet:

```
arcpy.analysis.Buffer("Pescadero_Creek",
r"C:\Users\Dara\Documents\ArcGIS\Projects\ClassDemo.gdb\Pescadero_Creek_Buffer", "1000 Feet",
"FULL", "ROUND", "ALL", None, "PLANAR")
```

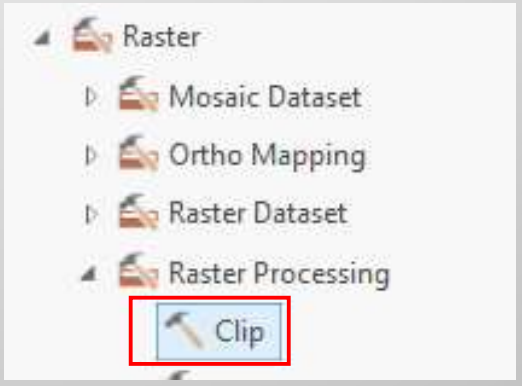
arcpy.Clip_analysis (inputData, clipFeatures, outputData)

Module function parameters for the function

tool name_tool box alias



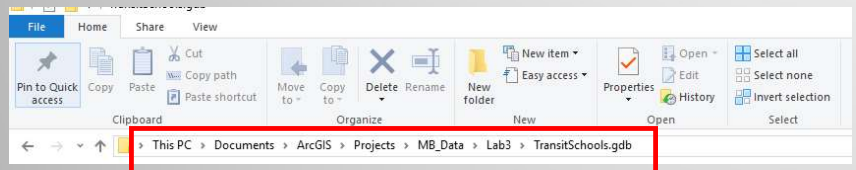

*arcpy.Clip_analysis (in_features, clip_features, out_feature_class, {cluster_tolerance})*




*arcpy.Clip_management (in_raster, rectangle, out_raster, {in_template_dataset}, {nodata_value}, {clipping_geometry}, {maintain_clipping_extent})*

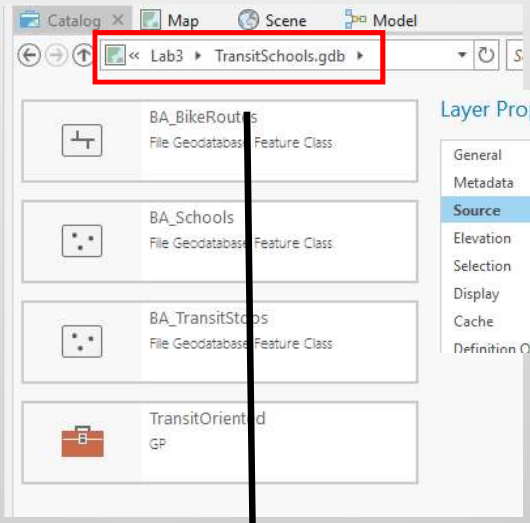
# Accessing data without a GUI:

## Open windows explorer:



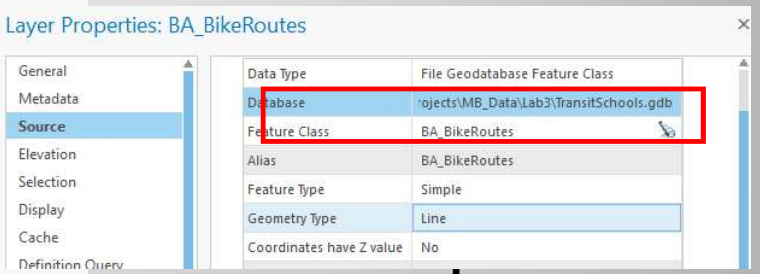

C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab3\TransitSchools.gdb

## Open Catalog View:




C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab3\TransitSchools.gdb

## Layer Properties Source tab:




C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab3\TransitSchools.gdb

# DEBUGGING

- Process of finding and fixing the problems or bugs in a program

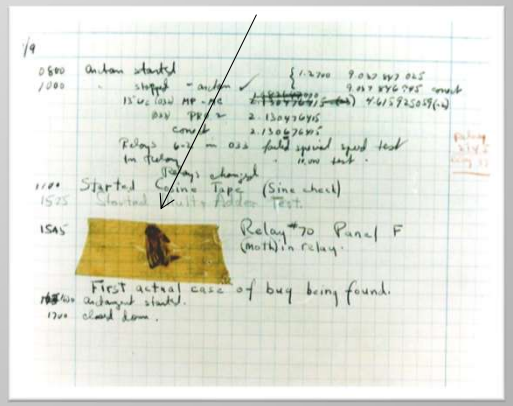
1. Visually inspect your program
2. Include print statements

```
12 print "Script sucessfully ran clip tool"
```

3. Selectively comment out lines of code

4. Use an IDE debugger

An early bug…



# HANDLING ERRORS

1. Syntax Errors [script won't run]
   - Problem with the grammar of the script

   Ex: missing brackets or quotes, lack of indentation

2. Exceptions or logic errors
   - Problem with the logic of the script

   Ex: reference to a misspelled variable name, non-existent data source, schema-lock etc.

## ICCE:

-Indentation

-Capitalizations

-Closures
[] "" ()

-Evaluations
(==)

# *Debugging Demo*

# Exercise 6

- Create a geoprocessing script from scratch in Spyder

# Exercise 6 Questions

1. What does this do? When should you not use this?

```
arcpy.env.overwriteOutput = True
```

2. What does this do?

```
import arcpy
```

3. Where did you run into the most problems debugging your scripts?

# Section 7

- Creating lists for geoprocessing
- Automating geoprocessing via for-loops
- If-then-else

# DATA TYPE: LISTS

- Allow you to store associated collection
- Are changeable (mutable)
- Uses square brackets [] to contain the list
- Uses commas to separate items

```
buffer_distances = [25,50,75]
input_data = [“C:\\Data\\river.shp”, “C:\\Data\\swamp.shp”]
herList = [ myList, yourList]
```

# LISTS IN GEOPROCESSING

- Very useful when automating tasks
- Data container
- The same tool sequence can be run on each thing in the list

```
arcpy.env.workspace = "C:\\Data"
feature_classes = ["streams.shp","counties.shp","roads.shp"]

fclist = arcpy.ListFeatureClasses()
print(fclist)
["streams.shp","counties.shp","roads.shp"]
```

- You can use wildcards and types to constrain list

```
rasters = arcpy. ListFeatureClasses("s*", "point")
```

# For Loops

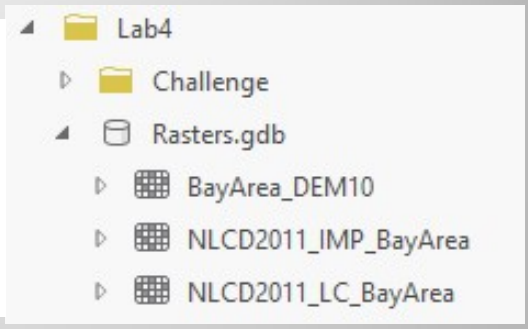
- For each item in a list:
  carry out a task on that item

```
arcpy.env.workspace = "C:\\Data"
rasters = arcpy.ListRasters() ← List

for file in rasters:
    arcpy.BuildPyramids_management (file)    } ← For loop
```

- note the indentation after the colon(:)

## Goal: To build pyramids for every raster in a geodatabase

```
3 import arcpy
4 arcpy.env.workspace = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab4\Rasters.gdb"
5
6 arcpy.BuildPyramids_management("BayArea_DEM10")
7 arcpy.BuildPyramids_management("NLCD2011_IMP_BayArea")
8 arcpy.BuildPyramids_management("BNLCD2011_LC_BayArea")
```




## What happens if you get additional rasters?

```
3 import arcpy
4 arcpy.env.workspace = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab4\Rasters.gdb"
5
6 rasters = arcpy.ListRasters()
7 print(rasters)
8 for item in rasters:
9     arcpy.BuildPyramids_management(item)
```

# *Lists and For-loops demonstration*

# If-Then-Else

- Controls whether an action happens or not

**If condition is true:**

**carry out action**

**else:**

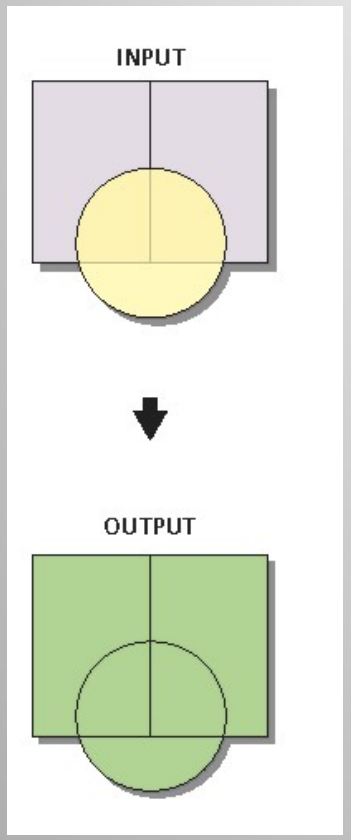
**carry out different action**

```
3 import arcpy
4 input_data = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab1\Cal_Public_Schools.shp"
5
6 #if the folder already exists copy the data
7
8 if arcpy.Exists(r"C:\demo"):
9     arcpy.CopyFeatures_management(input_data,r"C:\demo\schools")
10    print("Demo folder exists, transferred data")
11
12 #otherwise create the folder and copy the data
13 else:
14    arcpy.CreateFolder_management("C:\\","demo")
15    arcpy.CopyFeatures_management(input_data,r"C:\demo\schools")
16    print("Created demo folder, transferred data")
17
```

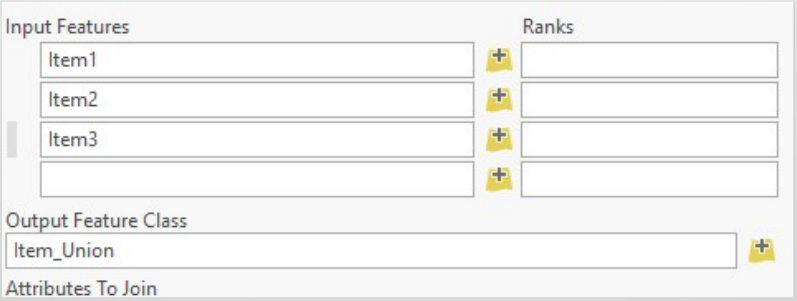
# Exercise 7

- Build off of the script you created in Lab 6
- Create a list and use it as an input into an Arc tool
- Write a for-loop

# Union




= [ “item1”, “item2”, “item3” ]



# Exercise 7 Questions

1. Tools in the arcpy module often have optional parameters. Suppose you have a tool with three optional parameters, and you only want to enter a value for the third of these parameters. What's one way you could you go about doing this?

2. Why include print statements in your scripts?

3. What are some advantages of using lists?

# Section 8

- Cursors in ArcPy
  - Search Cursors
  - Update Cursors
  - Insert Cursors

# WHAT ARE CURSORS?

- A database technology term for accessing a set of records in a table.
- Cursors can be used to iterate over a set of records or insert new records into a table.
- Records in the table are generally referred to as rows.
- In ArcGIS there 3 types of cursors: search, insert and update.

# SEARCH CURSORS

- A search cursor is used to retrieve rows and access data
- Methods supported:
  - next – Retrieves the next row
  - reset – Resets the cursor to its starting position

```
3  import arcpy
4
5  counties = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab1\ba_counties.shp"
6
7  with arcpy.da.SearchCursor(counties,["County", "Acres"]) as cursor:
8      for row in cursor:
9          county_name = row[0]
10         county_acres = row[1]
11         print("County name is {0} and the total acres is {1}".format(county_name, county_acres))
```

# INSERT CURSORS

- An insert cursor is used to insert new rows and create new records or data.
- Methods supported:
  - insertRow – inserts a row into the table
  - next – Retrieves the next row object
- With statement not supported must use *del* statement

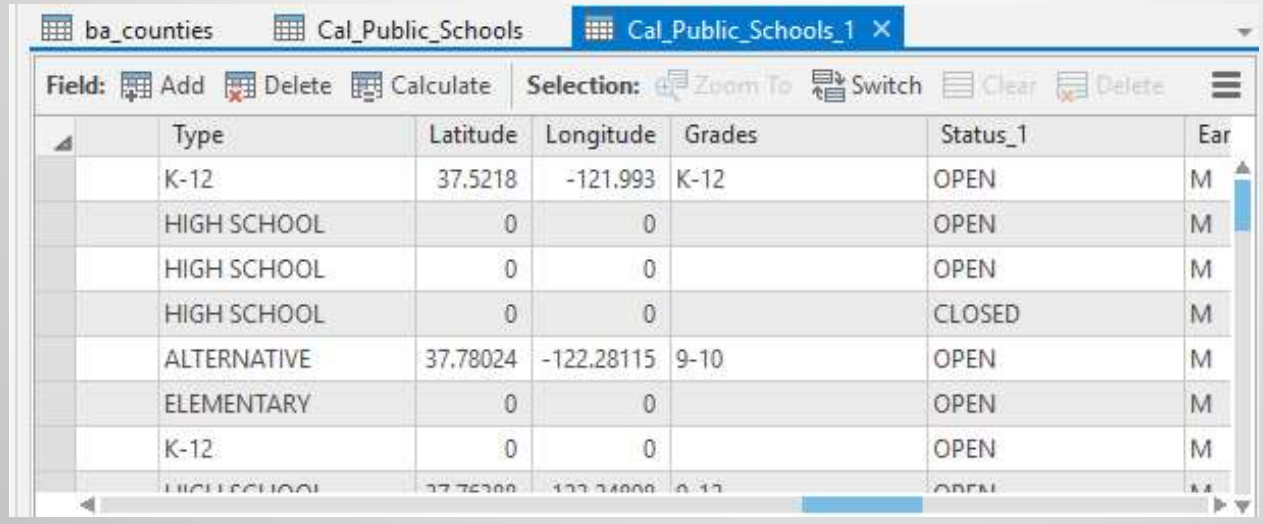
```
schools = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab1\Cal_Public_Schools_1.shp"

# A list of values that will be used to construct new rows
row_values = [('Example High', (-122.15,37.73)),
              ('Example Elementary', (-122.33,37.44))]

# Open an InsertCursor
cursor = arcpy.da.InsertCursor(schools, ['Name', 'SHAPE@XY'])

# Insert new rows that include the county name and a x,y coordinate
#  pair that represents the county center
for row in row_values:
    cursor.insertRow(row)

# Delete cursor object
del cursor
```

# UPDATE CURSORS

- An update cursor is used to update information or data in rows and/or delete rows.
- Methods supported:
  - deleteRow – Removes the row from the table
  - next – Retrieves the next row object
  - reset – Resets the cursor to its starting position
  - updateRow – Updates the current row

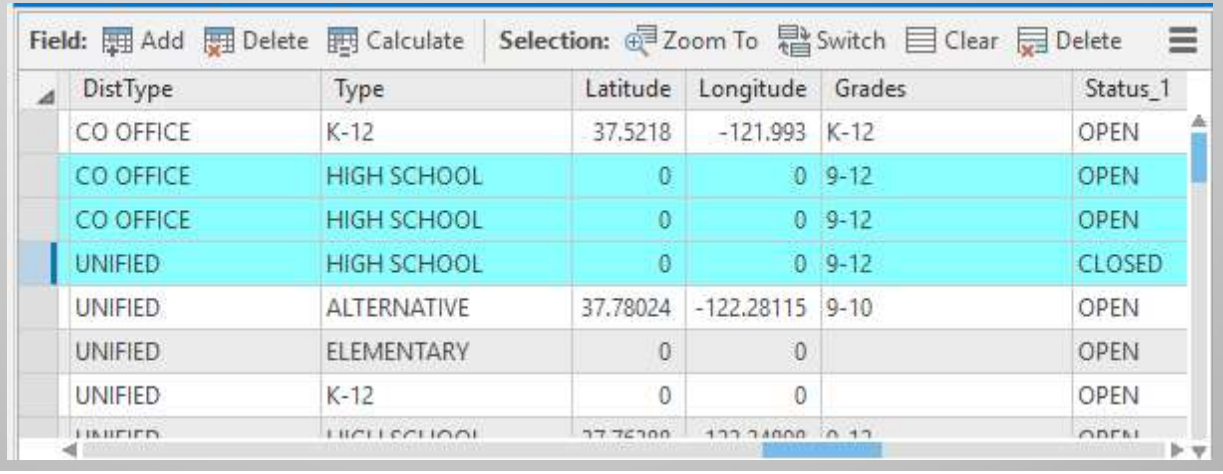
```
3  import arcpy
4
5  schools = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab1\Cal_Public_Schools_1.shp"
6
7  # Create update cursor for feature class
8  with arcpy.da.UpdateCursor(schools, ["Type","Grades"]) as cursor:
9      for row in cursor:
10         if row[0] == "HIGH SCHOOL":
11             row[1] = "9-12"
12
13         cursor.updateRow(row)
14
```

# UPDATE CURSORS




| Type | Latitude | Longitude | Grades | Status_1 | Ear |
|---|---|---|---|---|---|
| K-12 | 37.5218 | -121.993 | K-12 | OPEN | M |
| HIGH SCHOOL | 0 | 0 | | OPEN | M |
| HIGH SCHOOL | 0 | 0 | | OPEN | M |
| HIGH SCHOOL | 0 | 0 | | CLOSED | M |
| ALTERNATIVE | 37.78024 | -122.28115 | 9-10 | OPEN | M |
| ELEMENTARY | 0 | 0 | | OPEN | M |
| K-12 | 0 | 0 | | OPEN | M |

BEFORE




| DistType | Type | Latitude | Longitude | Grades | Status_1 |
|---|---|---|---|---|---|
| CO OFFICE | K-12 | 37.5218 | -121.993 | K-12 | OPEN |
| CO OFFICE | HIGH SCHOOL | 0 | 0 | 9-12 | OPEN |
| CO OFFICE | HIGH SCHOOL | 0 | 0 | 9-12 | OPEN |
| UNIFIED | HIGH SCHOOL | 0 | 0 | 9-12 | CLOSED |
| UNIFIED | ALTERNATIVE | 37.78024 | -122.28115 | 9-10 | OPEN |
| UNIFIED | ELEMENTARY | 0 | 0 | | OPEN |
| UNIFIED | K-12 | 0 | 0 | | OPEN |

AFTER

# DATA ACCESS MODULE (.da)

## arcpy.da

- Python module for working with data.
- Improves cursor support (including faster performance)
- Good practice to always use the data access module when using cursors

| | |
|---|---|
| Domain | The Domain object contains properties that describe an attribute domain. |
| Editor | The Editor class allows use of edit sessions and operations to manage database transactions. |
| InsertCursor | **`InsertCursor`** establishes a write cursor on a feature class or table. **`InsertCursor`** can be used to add new rows. |
| Replica | The Replica object contains properties that describe a replica. |
| SearchCursor | **`SearchCursor`** establishes read-only access to the records returned from a feature class or table. |
| UpdateCursor | **`UpdateCursor`** establishes read-write access to records returned from a feature class or table. |
| Version | The Version object contains properties that describe a version. |

# USING SQL IN PYTHON

- Can use Structured Query Language (SQL) in cursors to run on specific rows.
- This is a very useful application of string formatting covered earlier – *sql = "Type = '{0}'".format(some_var)*

```
3 import arcpy
4
5
6 schools = r"C:\Users\Dara\Documents\ArcGIS\Projects\MB_Data\Lab1\Cal_Public_Schools.shp"
7
8 sql = "Zip = 94501"
9
10 with arcpy.da.SearchCursor(schools,["Name", "City"], sql) as cursor:
11     for row in cursor:
12         school_name = row[0]
13         school_city = row[1]
14         print("School name is {0} and school city is {1}".format(school_name, school_city))
```

# ArcPy TOKENS

- Geometry tokens can also be used as shortcuts in place of accessing full geometry objects.
- If you only need specific properties of the geometry, use tokens to provide shortcuts to access geometry properties.
- For examplle, **SHAPE@XY** returns a tuple of x,y coordinates that represent the feature's centroid.

| Token | Explanation |
|---|---|
| **SHAPE@** | A geometry object for the feature. |
| **SHAPE@XY** | A tuple of the feature's centroid x,y coordinates. |
| **SHAPE@TRUECENTROID** | A tuple of the feature's true centroid x,y coordinates. |
| **SHAPE@X** | A double of the feature's x-coordinate. |
| **SHAPE@Y** | A double of the feature's y-coordinate. |
| **SHAPE@Z** | A double of the feature's z-coordinate. |
| **SHAPE@M** | A double of the feature's m-value. |
| **SHAPE@JSON** | The esri JSON string representing the geometry. |
| **SHAPE@WKB** | The well-known binary (WKB) representation for OGC geometry. It provides a portable representation of a geometry value as a contiguous stream of bytes. |
| **SHAPE@WKT** | The well-known text (WKT) representation for OGC geometry. It provides a portable representation of a geometry value as a text string. |
| **SHAPE@AREA** | A double of the feature's area. |
| **SHAPE@LENGTH** | A double of the feature's length. |

# Exercise 8

- Working with search cursors and insert cursors

# Exercise 8 Questions

1) Why did we use a “del” statement in this script?

2) Do you see how we used string formatting in the select by layer tool?

3) What is the difference between an insert and update cursor?

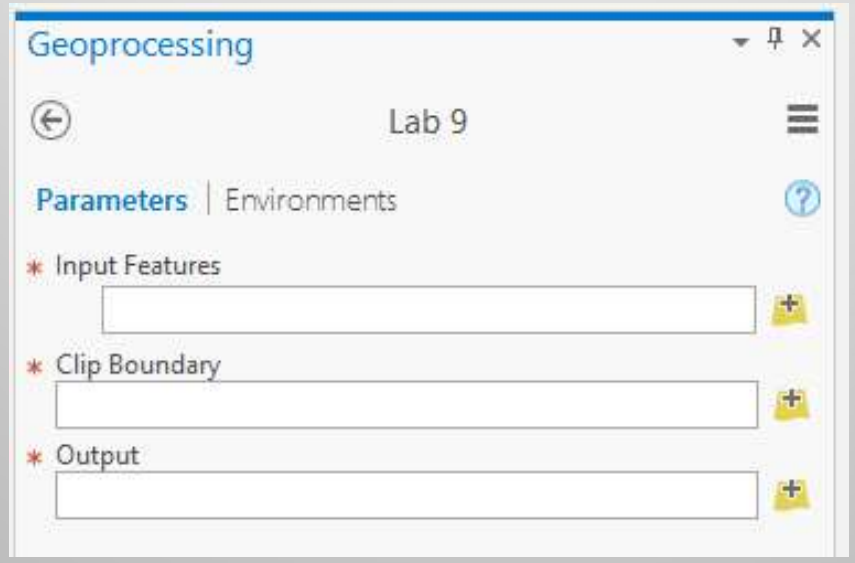
4) Why do use the “MakeFeatureLayer_management” tool in this exercise?

# Section 9

- Making a python script into a script tool

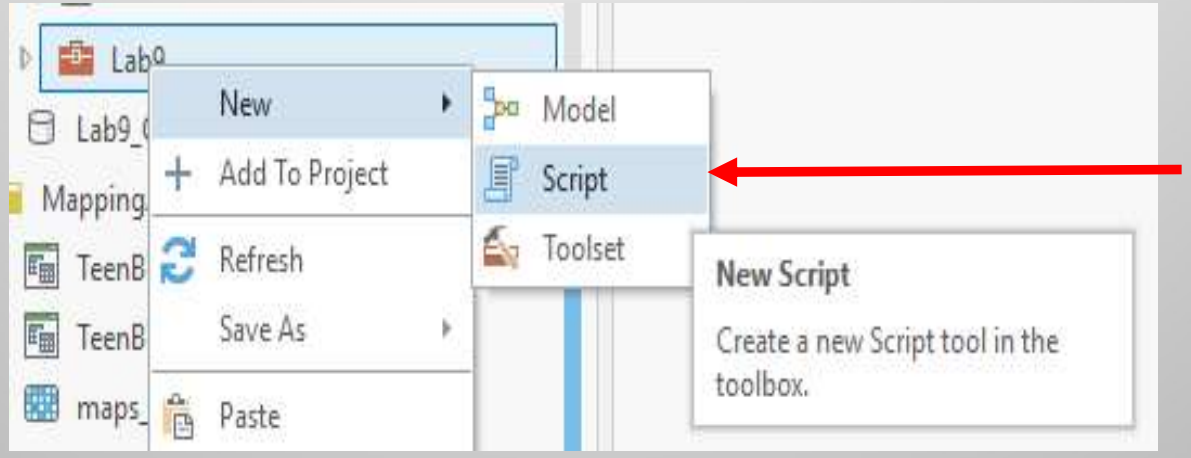
# WHY A SCRIPT TOOL?

- Easier for a non-python user
- You can restrict what inputs a user can specify
- Allows you to navigate the folder system to get a path name

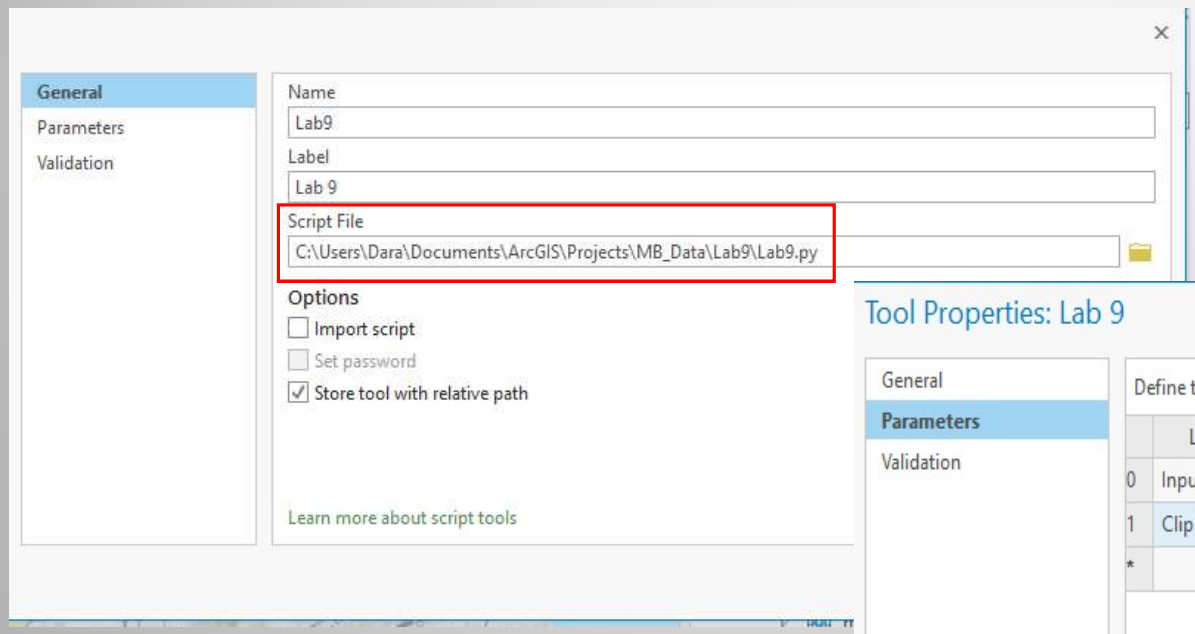


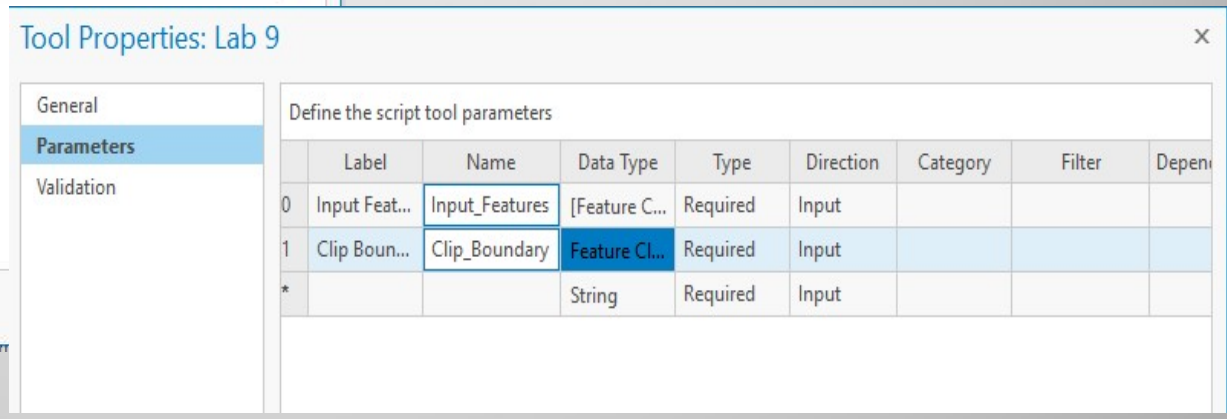
# CREATING A SCRIPT TOOL

- Script tools are stored in a custom toolbox (like models)
- Right-click **New>>Script**



# Script Tool Set Up Wizard



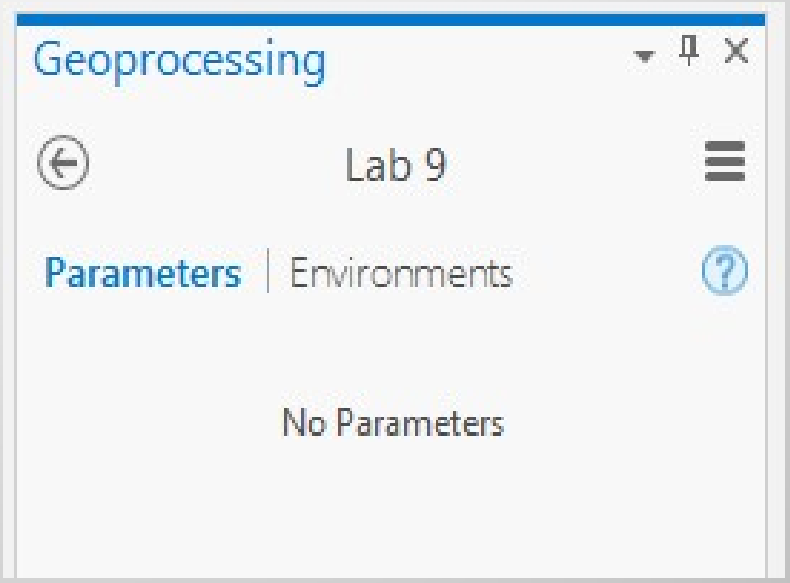




1. Provide descriptive information and define the script to run

2 If there are parameters, create their display name and specify their data type

# A script tool without parameters:

- Must edit the script itself to make any changes to inputs/output

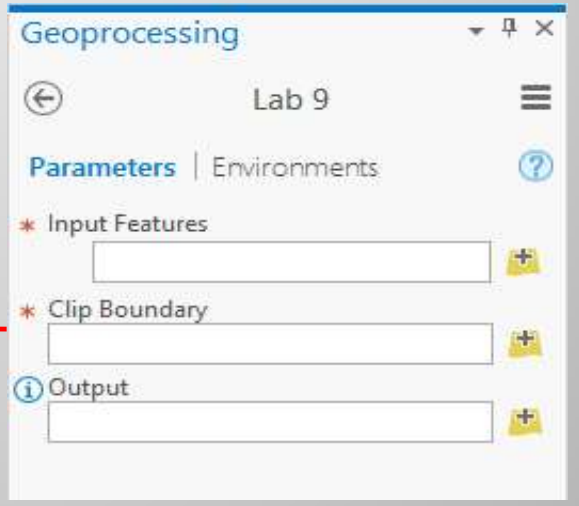


# PREPARING YOUR SCRIPT

- The python script should have places for user-defined parameters to go
  arcpy.GetParameterAsText(index)

- **Order matters** when setting up the parameters in the script tool properties

```
12 input_features = str.split(arcpy.GetParameterAsText(0), ";")
13
14 clip_boundary = arcpy.GetParameterAsText(1)
15
16 output = arcpy.GetParameterAsText(2)
```



# SHARING A SCRIPT TOOL

- Must share the custom tool box that contains the script tool, **and** the script
- Script tool only points to a python script—nothing is saved in the script tool itself

# PYTHON TOOLBOXES

- Geoprocessing toolboxes created entirely in python
- Self-contained in a .pyt file
- More efficient to maintain and update

```
import arcpy

class Toolbox(object):
    def __init__(self):
        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."
```

Names the toolbox

```
class CalculateSinuosity(object):
    def __init__(self):
        self.label       = "Calculate Sinuosity"
        self.description = "Sinuosity measures the amount that a river meanders"

    def getParameterInfo(self):
        #Define parameter definitions

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

        in_features.filter.list = ["Polyline"]

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

        sinuosity_field.value = "sinuosity"

        # Derived Output Features parameter
        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
```

Defines tool names and parameters

```
    def execute(self, parameters, messages):
        inFeatures  = parameters[0].valueAsText
        fieldName   = parameters[1].valueAsText

        if fieldName in ["#", "", None]:
            fieldName = "sinuosity"

        arcpy.AddField_management(inFeatures, fieldName, 'DOUBLE')

        expression = '''
import math
def getSinuosity(shape):
    length = shape.length
    d = math.sqrt((shape.firstPoint.X - shape.lastPoint.X) ** 2 +
                  (shape.firstPoint.Y - shape.lastPoint.Y) ** 2)
    return d/length
'''

        arcpy.CalculateField_management(inFeatures,
                                        fieldName,
                                        'getSinuosity(!shape!)',
                                        'PYTHON_9.3',
                                        expression)
```

Carries out the geoprocessing tasks

# Exercise 9

- Create a script tool 

# Exercise 9 Questions:

1. Why does order matter when setting up parameters in the script tool parameters tab?

2. If we already have a python script, which can be run outside of Arc, why would we want to make a script tool?

3. What does `arcpy.GeParameterAsText()` do?

# Section 10

- ArcGIS API for Python
- Python for Web GIS
- Quick Introduction to Jupyter Notebooks

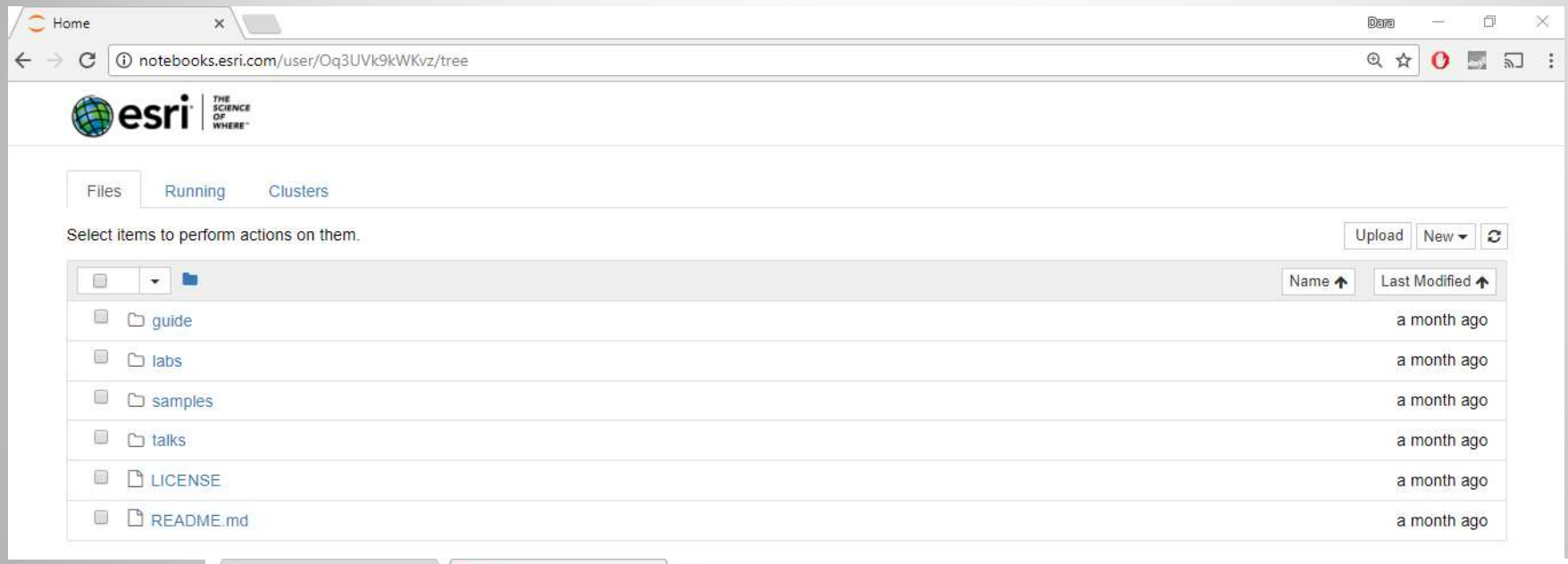
# ArcGIS API for Python

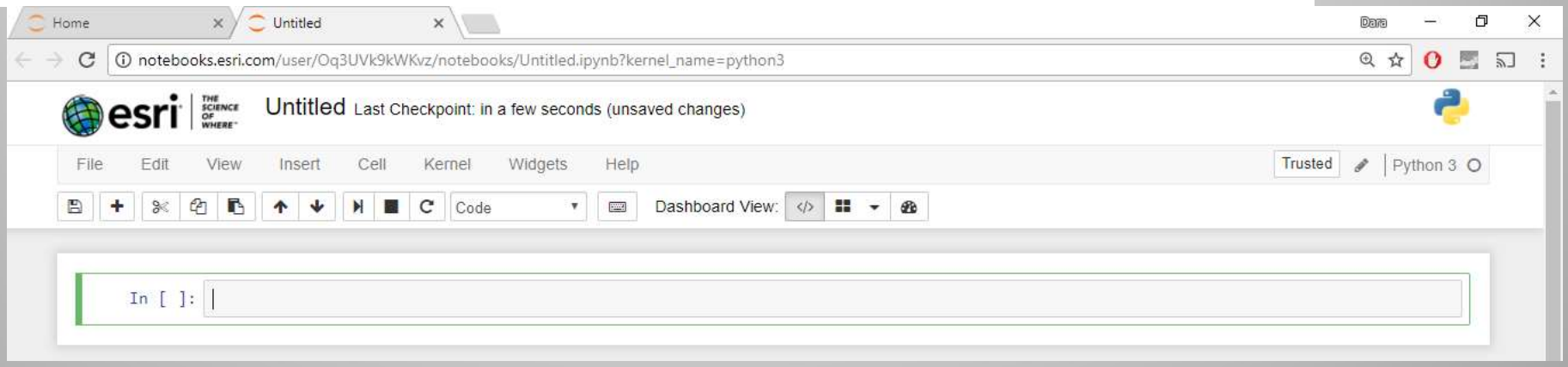
- Completely independent from ArcPy
- Separate install in your conda environment
- This is the beginning of using Python for web GIS
- These include:
  - ArcGIS Server
  - ArcGIS Online
  - ArcGIS for Portal
- Released 2016 still in the early phases of developement

# JUPYTER NOTEBOOKS

- Jupyter notebooks are iPython Notebooks, browser based console
- Makes it easier to share code and visualize outputs
- To install Jupyter “conda install jupyter”
- Execute “jupyter notebook”
- This will open a local host server where you can create python notebooks
- To experiment without installing and running Jupyter you can use this: http://notebooks.esri.com

# JUPYTER NOTEBOOKS



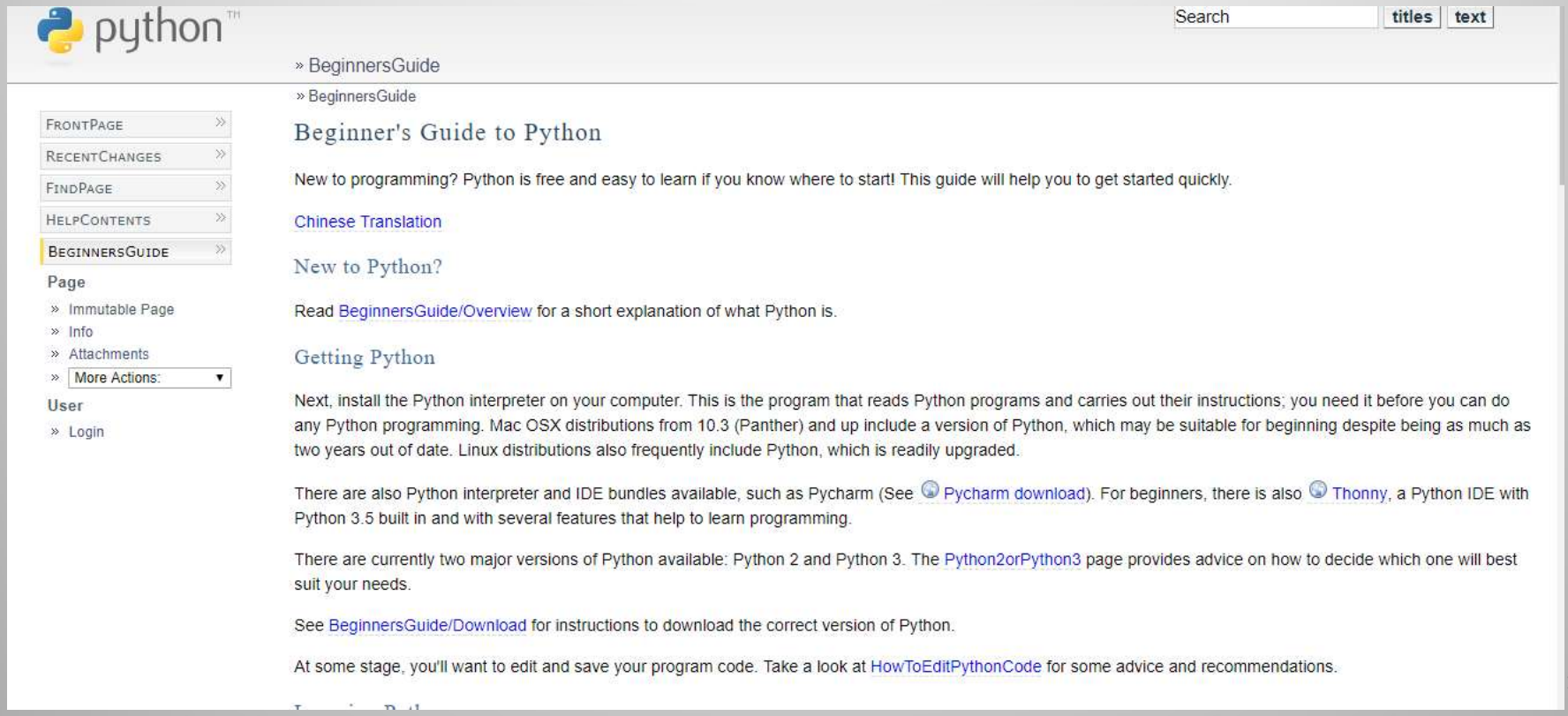



# EXERCISE 10

- Short introduction to the ArcGIS API for Python

# PYTHON DOCUMENTATION

https://wiki.python.org/moin/BeginnersGuide



# PYTHON ONLINE RESOURCES TUTORIALS

http://newcoder.io/ New Coder


https://learnpythonthehardway.org/

http://automatetheboringstuff.com/

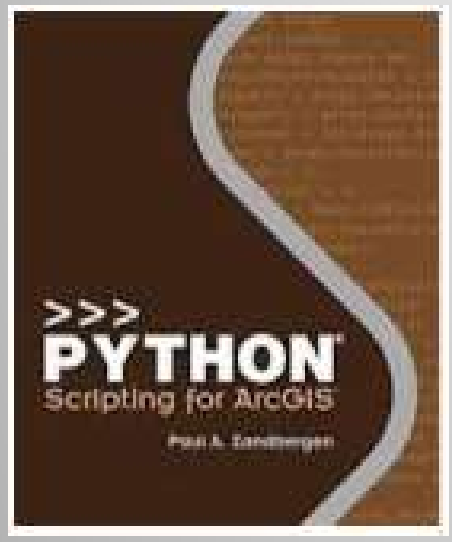

https://www.programiz.com/python-programming

**Learn Python Programming**

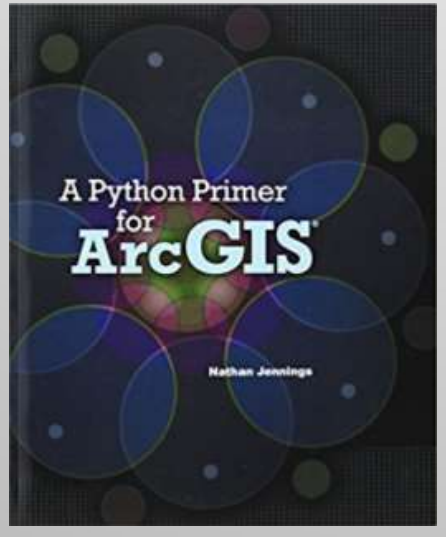
The Definitive Guide

# PYTHON BOOKS

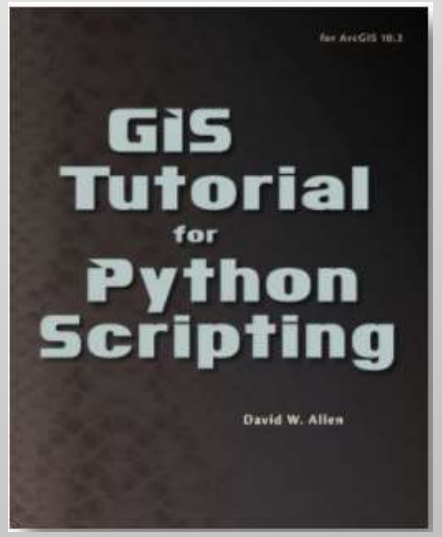
**Python Scripting for ArcGIS** by Paul A. Zandbergen




**A Python Primer for ArcGIS®**
by Nathan Jennings




**GIS Tutorial for Python Scripting**
by David W. Allen




**ArcPy and ArcGIS - Second Edition** by Silas Toms, Dara O'Beirne