InSPIRE

Innovation Summit for
Preparedness & Resilience

Spatial Craftsmanship

Learn to Build Custom Solutions in ArcGIS Pro

Adam Fackler, GIS Specialist

NAPSG Foundation & Missouri Task Force 1

NAPSG Foundation

InSPIRE

What does it mean to “Build a tool”

We’re not creating an entirely new geoprocessing tool,

We’re tying existing tools together to make things go faster!

But why?

- Standardize workflows in your organization
- Automate data processing
- Workflow sharing

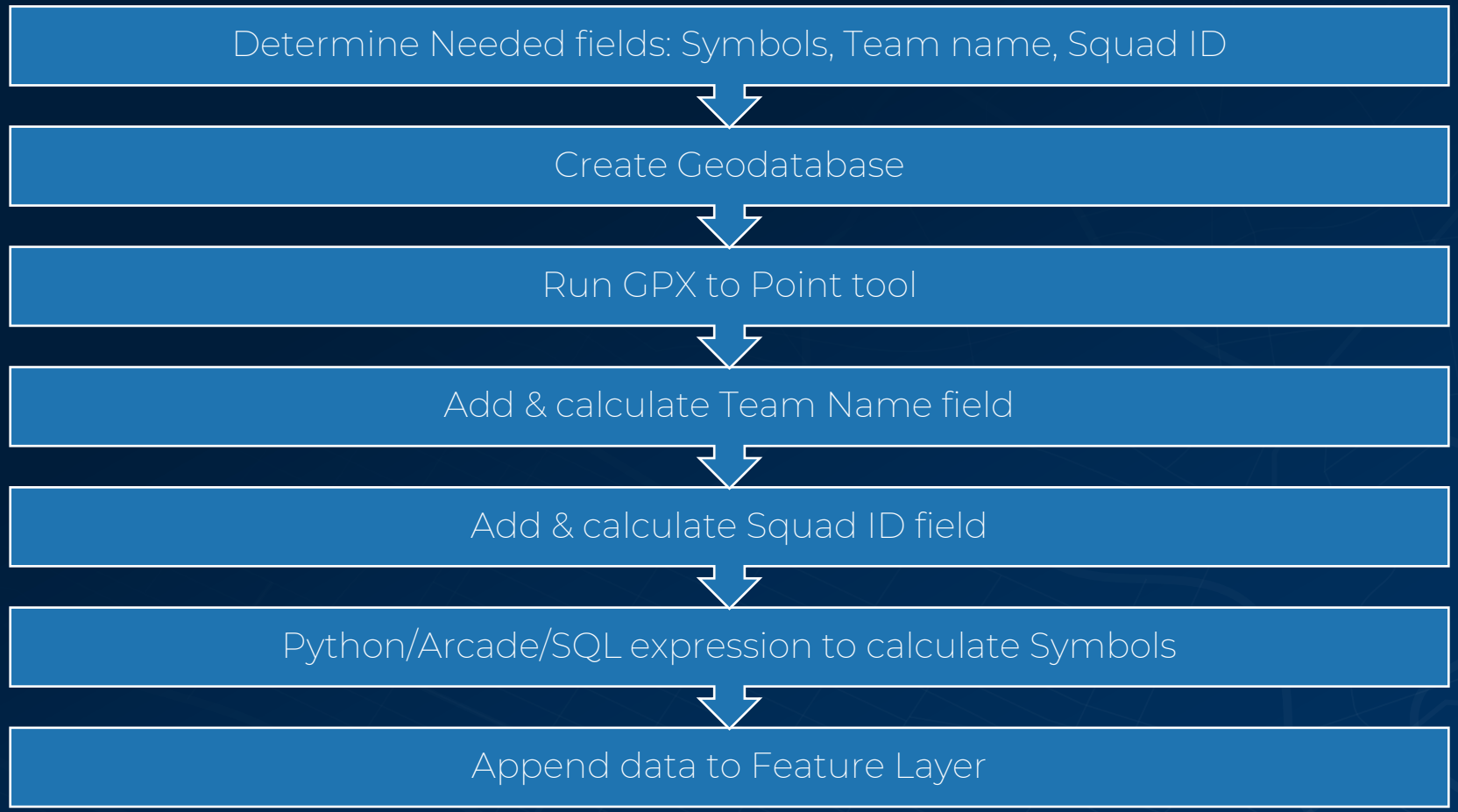
NAPSG Foundation

InSPIRE

Scenario

- Your worst nightmare has reared its ugly head for the first time since 2019… You are assigned to convert waypoints collected from the field using Garmin GPS units equipped with IronSights into something that SARCOP can read. You also are told that several teams are out collecting data with the same system and will be bringing you the data several times over the next few days

How do we usually do this?



NAPSG Foundation

InSPIRE

Ways to create these tools

Modelbuilder

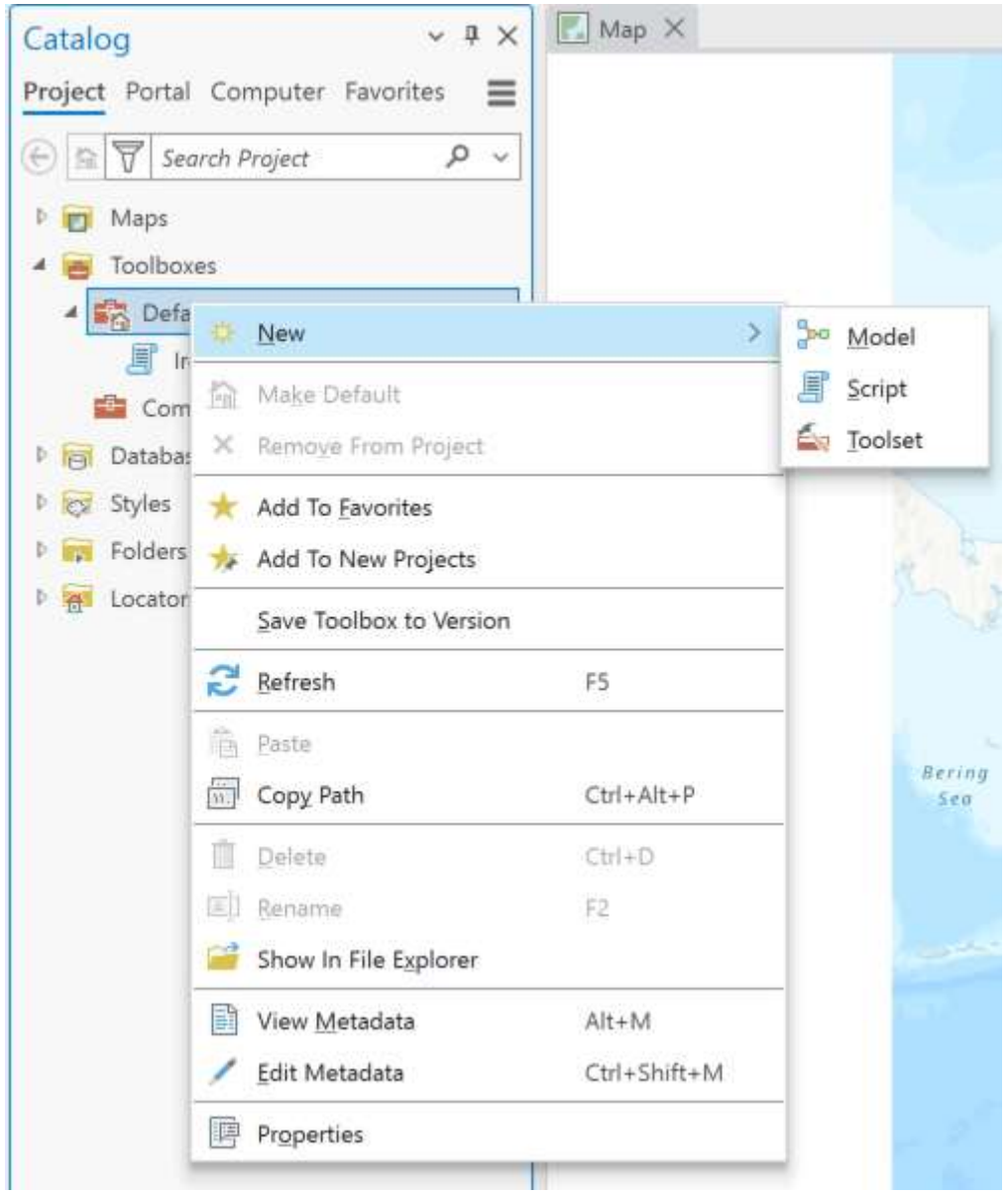
- Beginner friendly
- Diagram driven
- Great backend functionality, Front-end can be challenging
- Great for quickly tying tools together
- Handles intermediate data automatically

Custom Python Script Tool

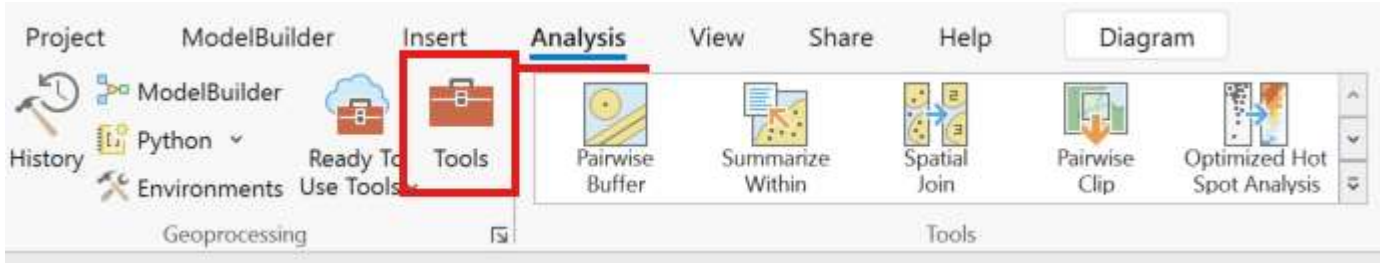
- Programming know-how required (Specifically ArcPy)
- Easier to scale up to include tasks beyond GIS
- Skills can be applied across ArcGIS suite

Modelbuilder

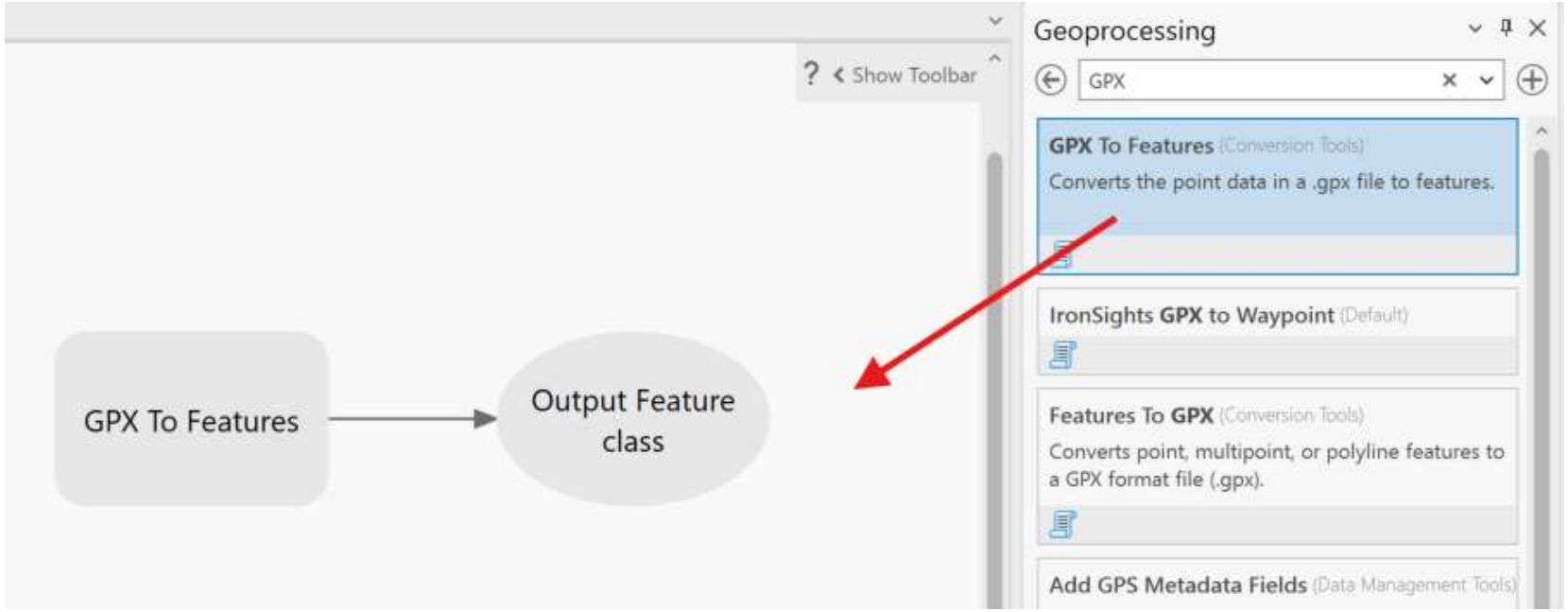
- Open an APRX and go into the catalog. Then open Toolboxes, then right-click on a toolbox and select “New Model” to create a new model
- The model should open by default. If not, right-click on the model and click Edit
- See the full documentation for Modelbuilder here: [Esri Documentation link]



- With the model open, go to the Analysis tab and open Tools



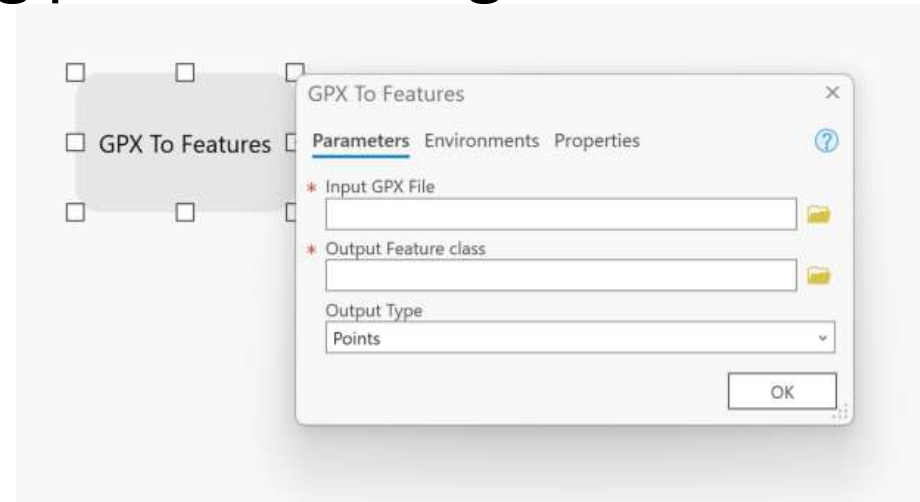
- Search for GPX to Features. Before opening the tool, click and hold on the tool, then drag it into the model. This will add the tool into your model



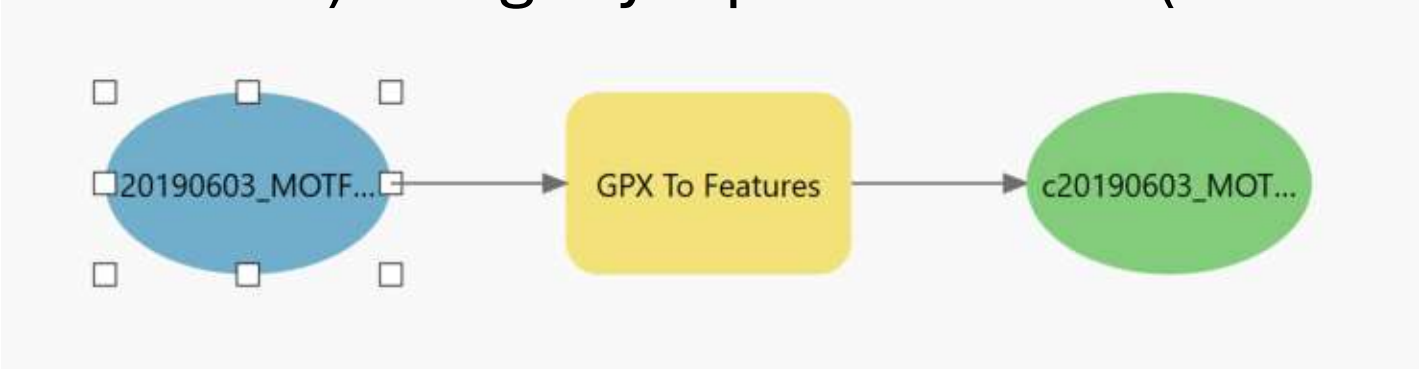
- To edit the tool's parameters, double-click on the GPX to Features box. This will look like the parameters you would see when you open the tool in the geoprocessing pane. Clicking OK does not run the tool in this case

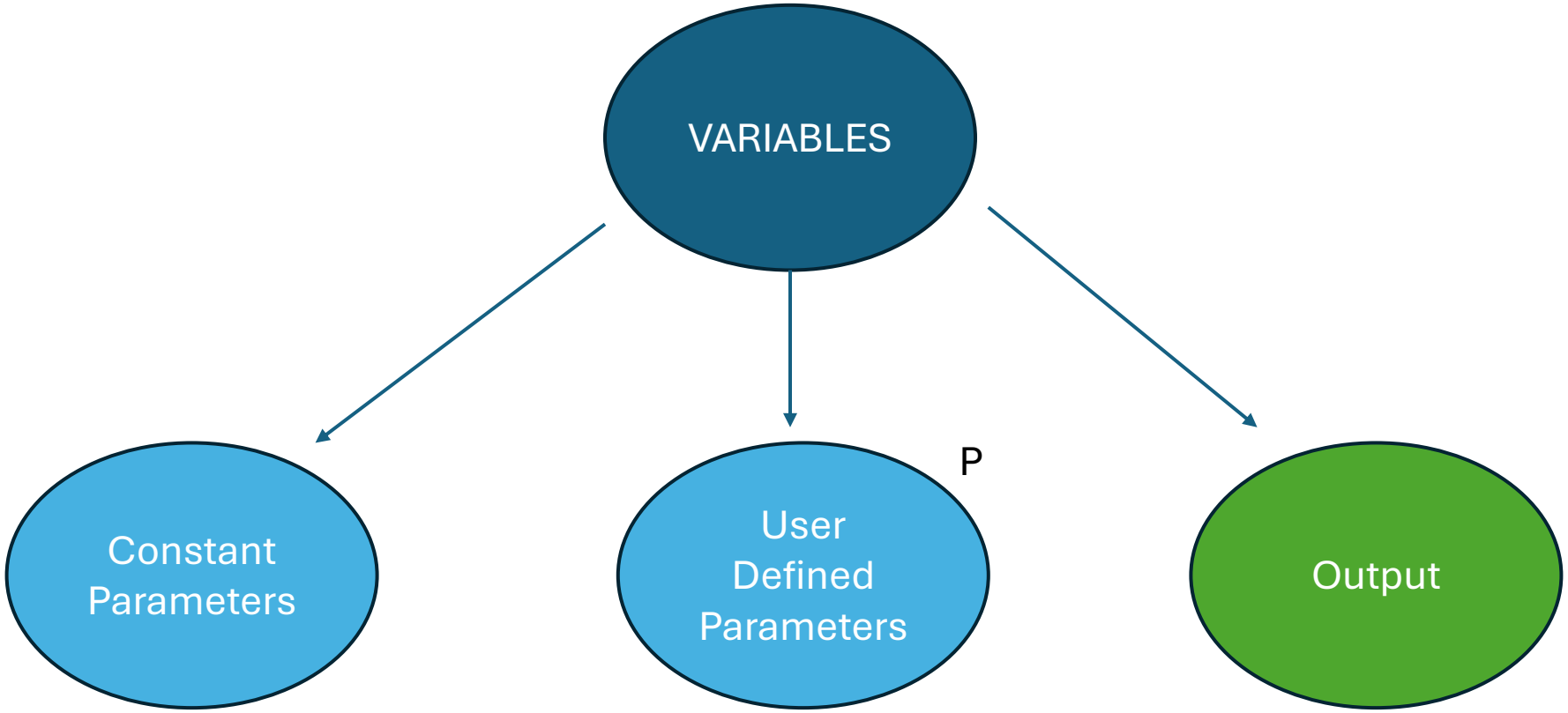
TOOLS

VARIABLES



- If you fill in the parameters, the tool will change color, signifying that the tool's parameters are met. Other branches will also appear (based on the tool) to signify input variables (such as the input GPX file)

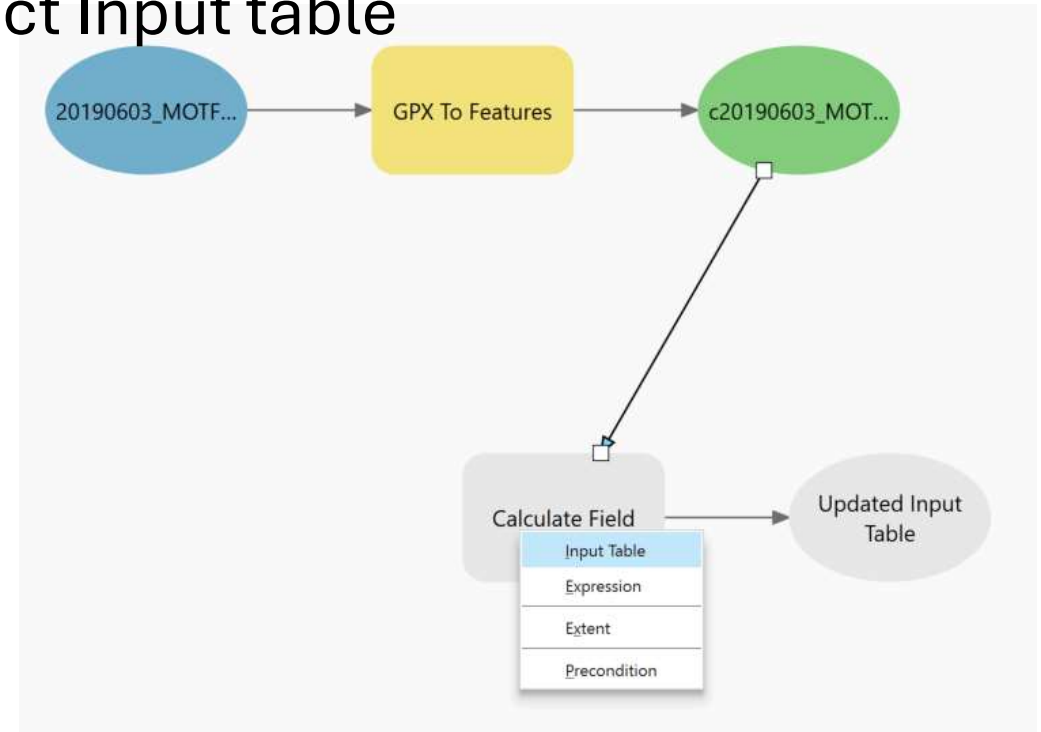




Variables can take three forms:

1. Constant parameters are defined parameters that only change in the Model Editor
2. User defined Parameters sport a P in the top right and will appear as a User Input parameter when the model is ran using the Geoprocessing pane
3. Output displays the output of the tool (such as a feature class)

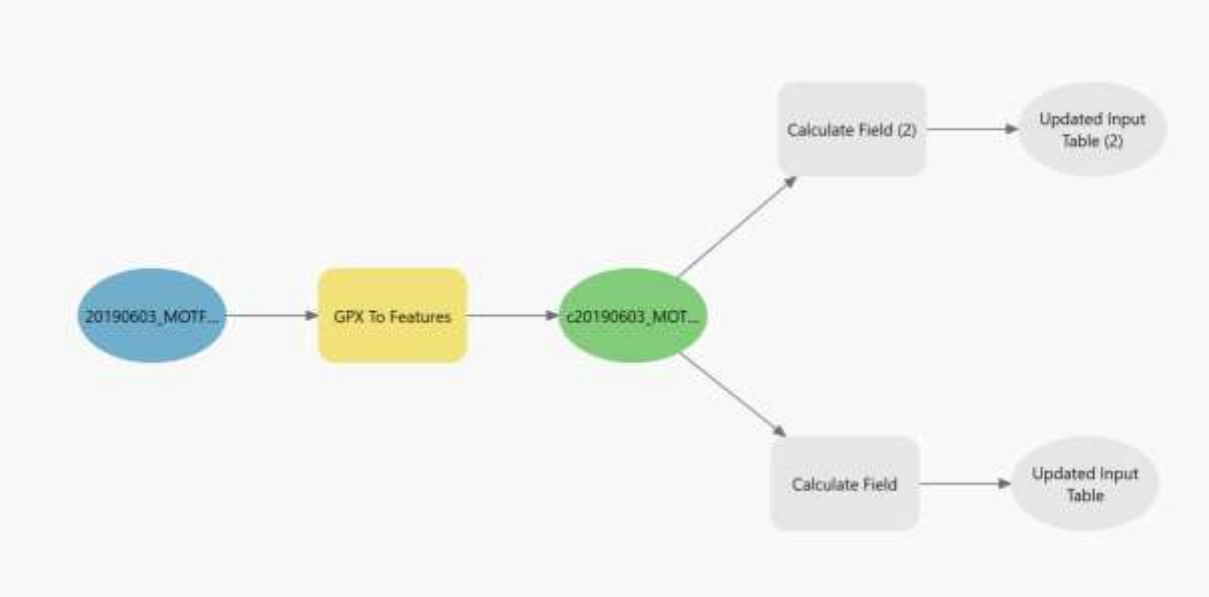
- Search for the tool “Field Calculate” and drag it into your model. Then set it up to calculate a new field called “Team_Name”
- When complete, click and hold on the output feature class of the GPX to Features (the green variable) and drag the arrow to the first Field calculate. Select Input table

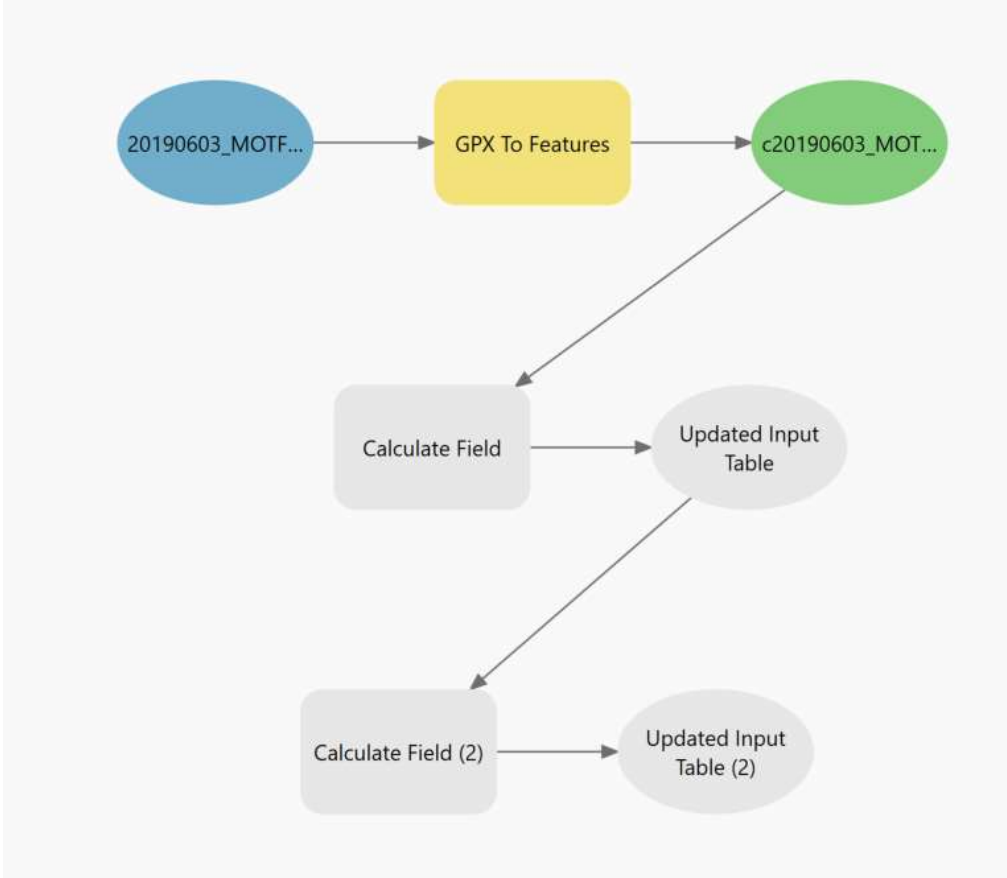


- Do this a second time using the Updated Input Table as the new feature class, connecting it to another field calculate in the same way. This connecting is how you can tie the tools together in Modelbuilder

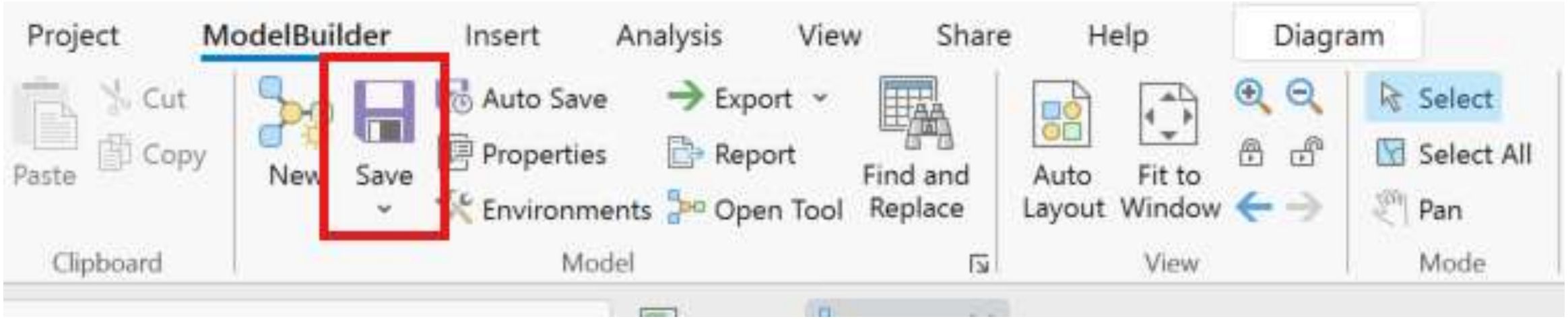
- Complete the same steps for a second field calculate tool, this time for Squad ID
- Connect it to the last tool’s output feature class. See the next slide for a diagram of what the tool should look like now

With the GPX to Features and two field calculates, your tool should look like one of the below. You can set up this step either way and the model will accomplish the same thing, but always keep in mind what steps are required for the tool and wire the model accordingly.

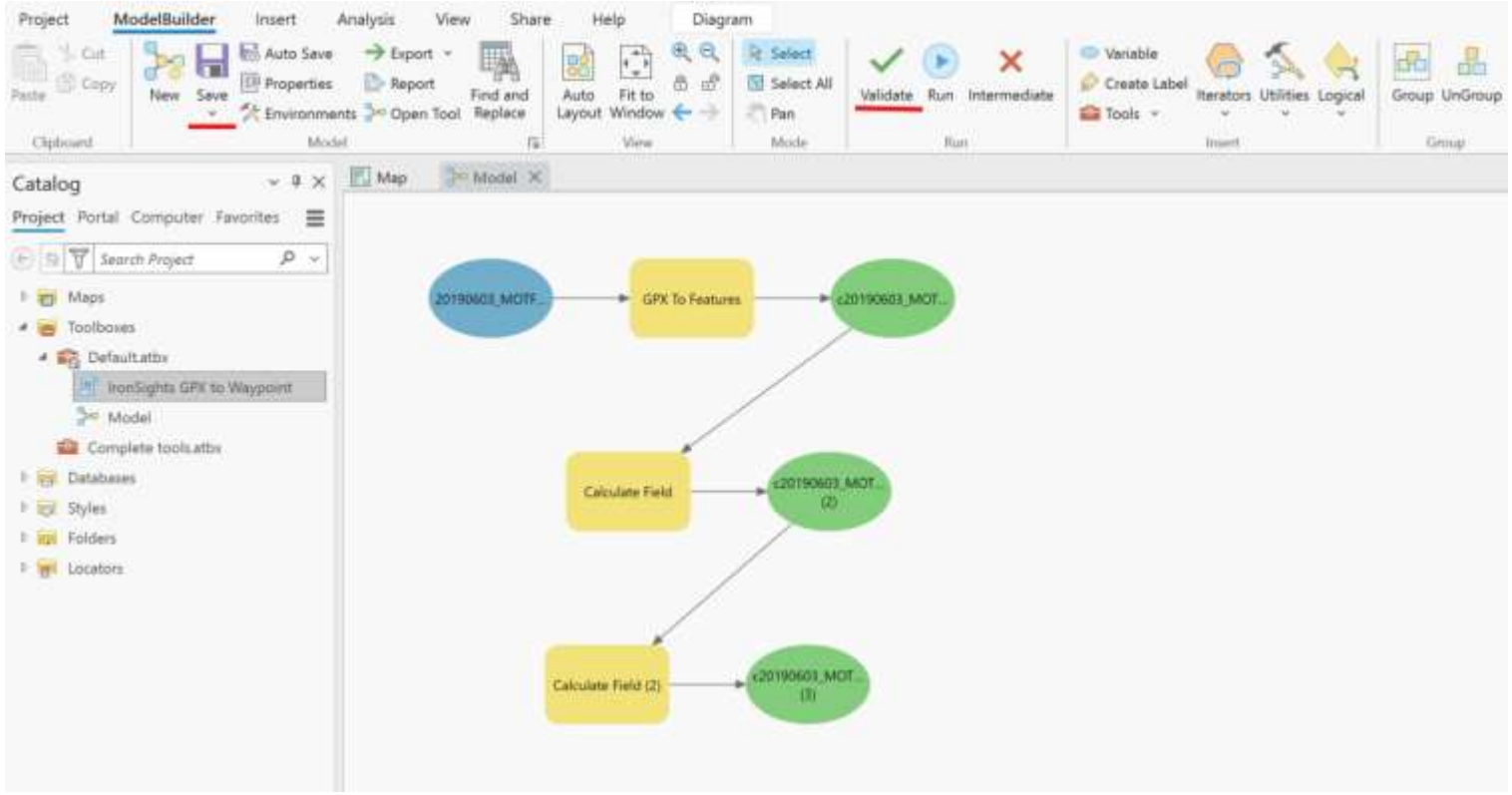




Remember to save!!



- Double-click on the field calculate tools and fill out tools to make one create and calculate a team name, the other a Squad ID (if not done already)
  - Field calculate lets you create and calculate a new field in one step
- When complete, the tool should be all colored in
- Save the tool and click Validate in the Modelbuilder tab

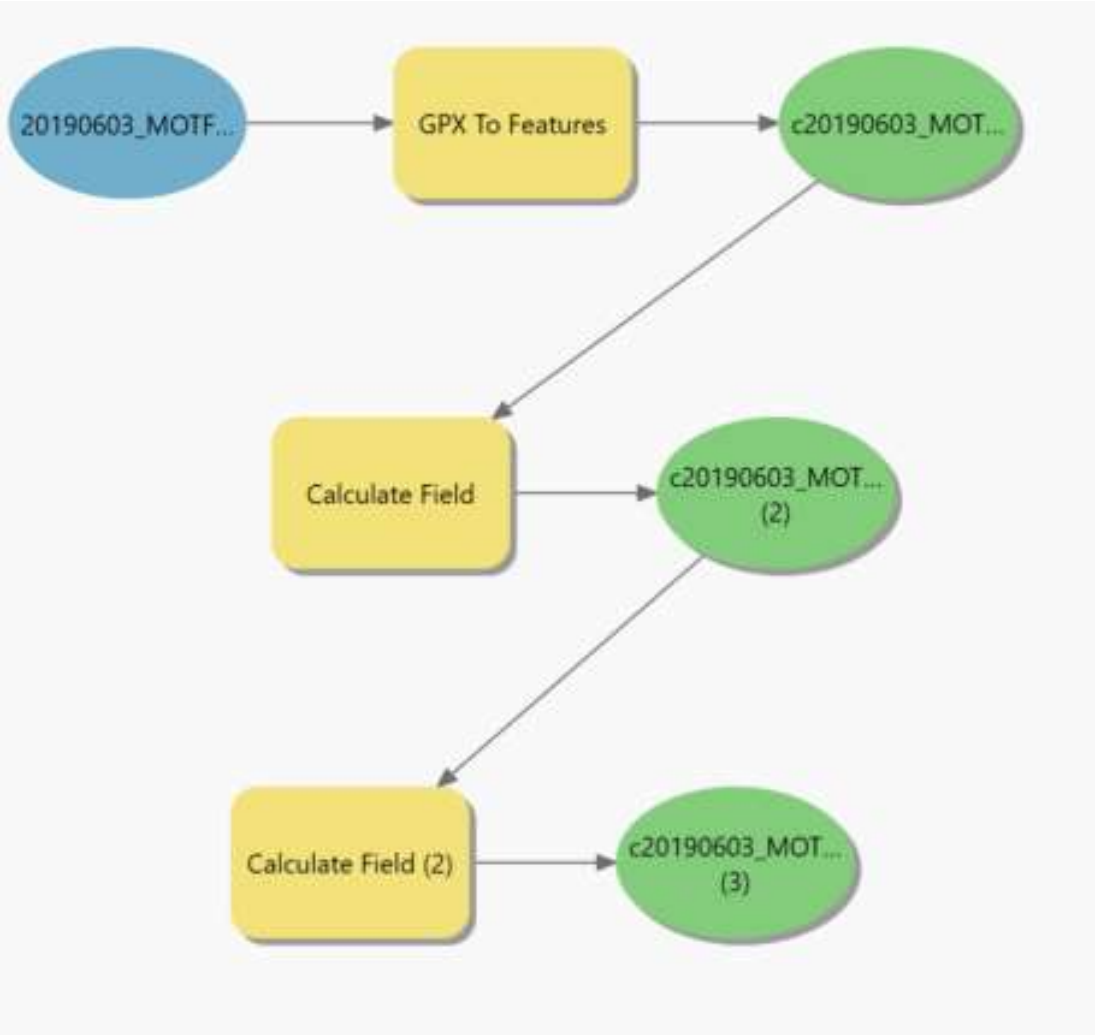


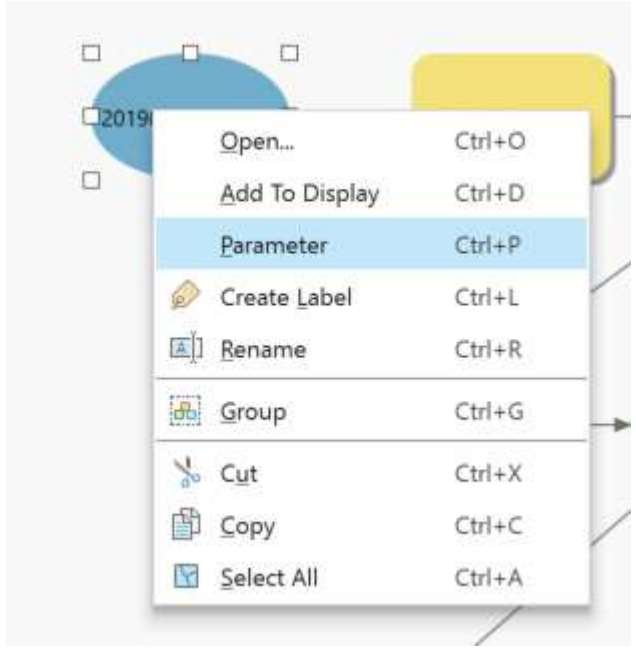
Don't worry if nothing happens when clicking Validate; this is to make sure everything is filled out. You do NOT need to click this to run the tool, but it's good practice to run this to make sure your tool doesn't throw any immediate errors.

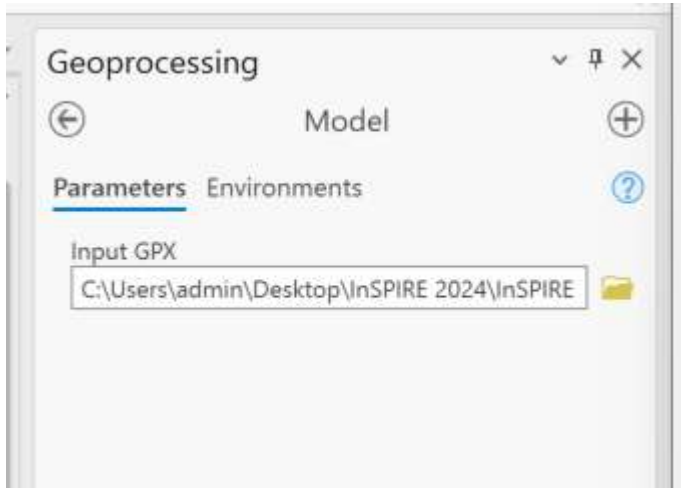
When Ready, click Run next to the Validate button and watch the tool work. When done, go to the default geodatabase for your data output

- The tool that is currently running will be highlighted in Red , and a shadow will appear behind the variable and tool when it is successfully run. This is an example of the tool when it is complete

- With all of this, the tool likely does not take any user inputs and will run with the user defined features. Next we will put in some parameters to change what files we run the tool with.

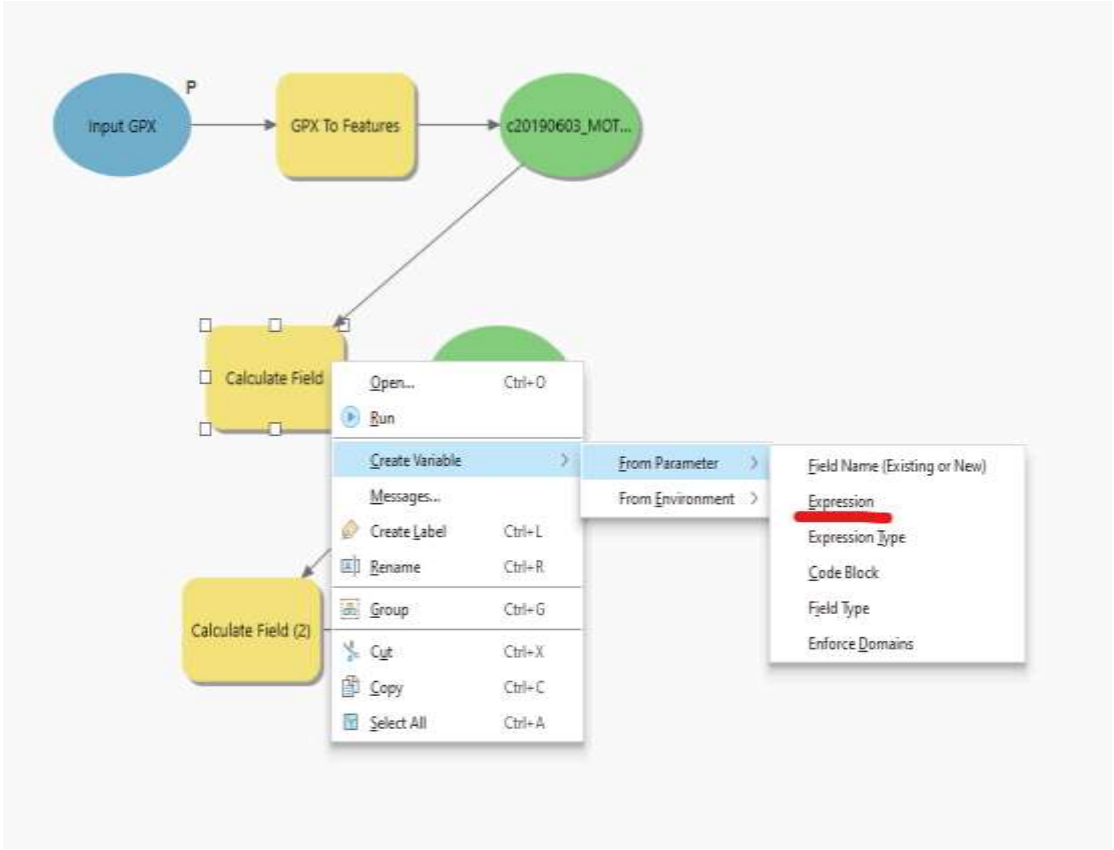


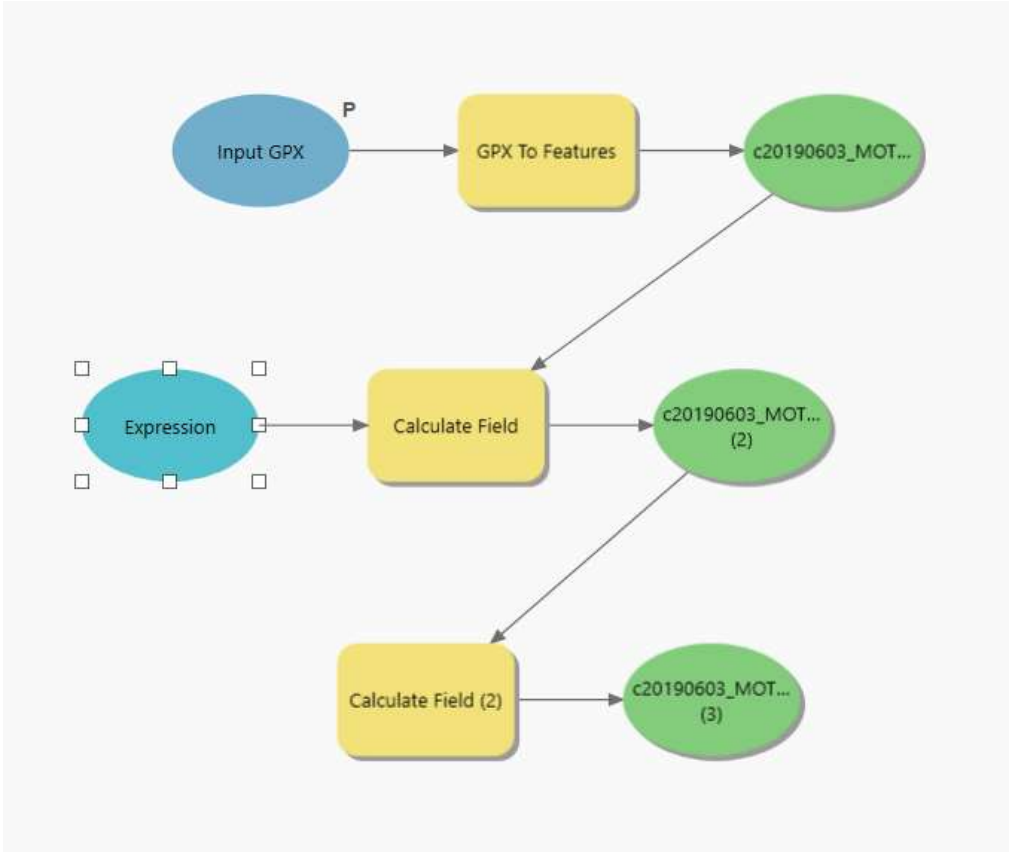




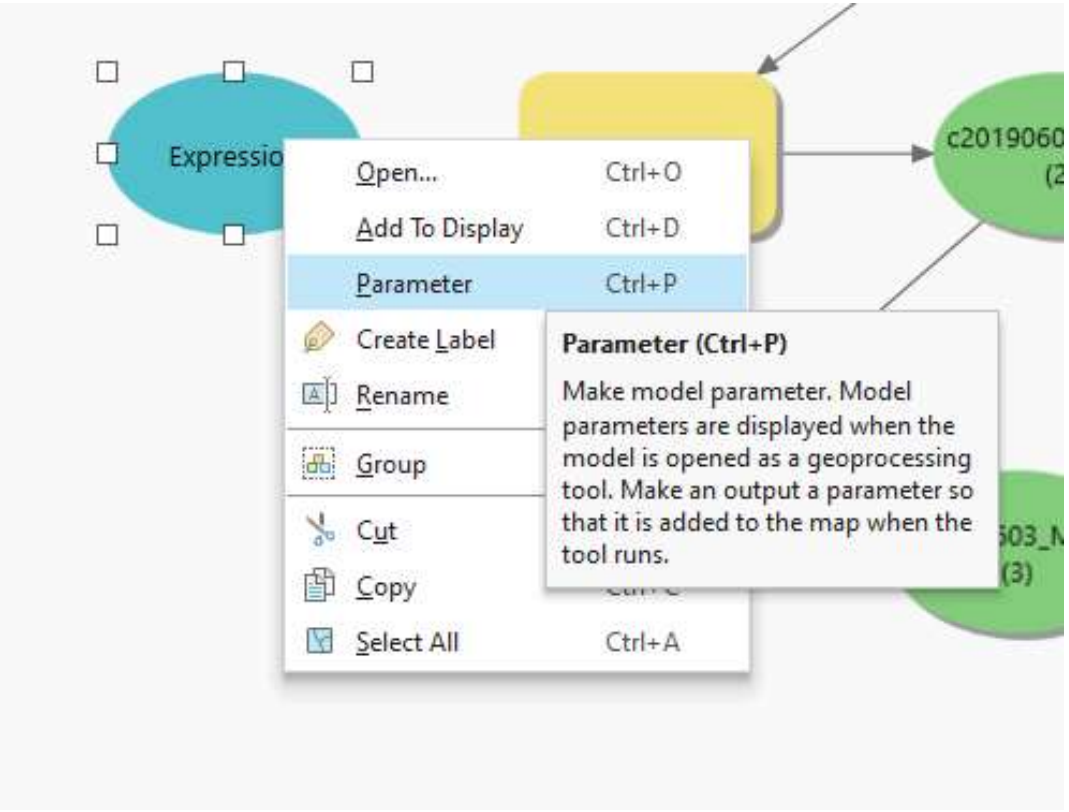
- Let's add in a user parameter now. Right-click on the input GPX variable and select Parameter
- A small "P" will appear in the top right of the bubble, signifying that this is a use input parameter
- After saving the model, double click on the model in the toolbox. This will open in the geoprocessing pane with all the user parameters you set (in this case so far just one – the input GPX)
- Right-click on the variable oval within your model and click Rename. Whatever the variable name is changed to will change the name of the parameter in the Geoprocessing pane

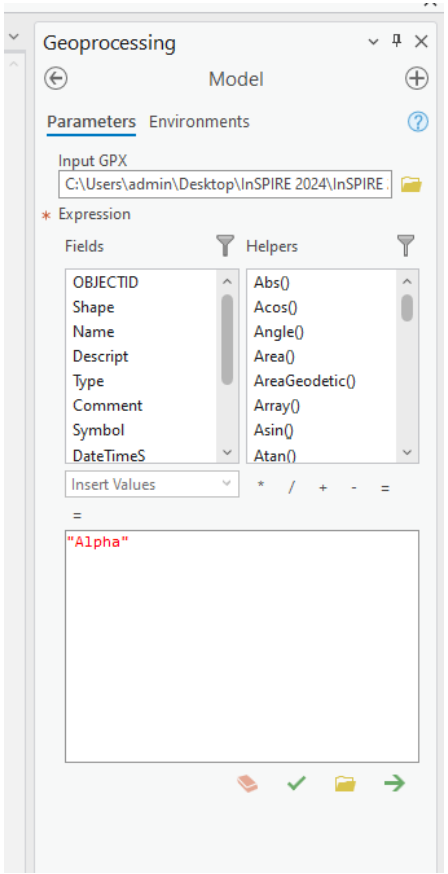
- You can do the same thing on any tool by right clicking on the tool and selecting Create Variable > From Parameter, then choosing what parameter you would like as the user input. In this case we will make the Expression a parameter so it can be changed when the tool is ran in the geoprocessing pane



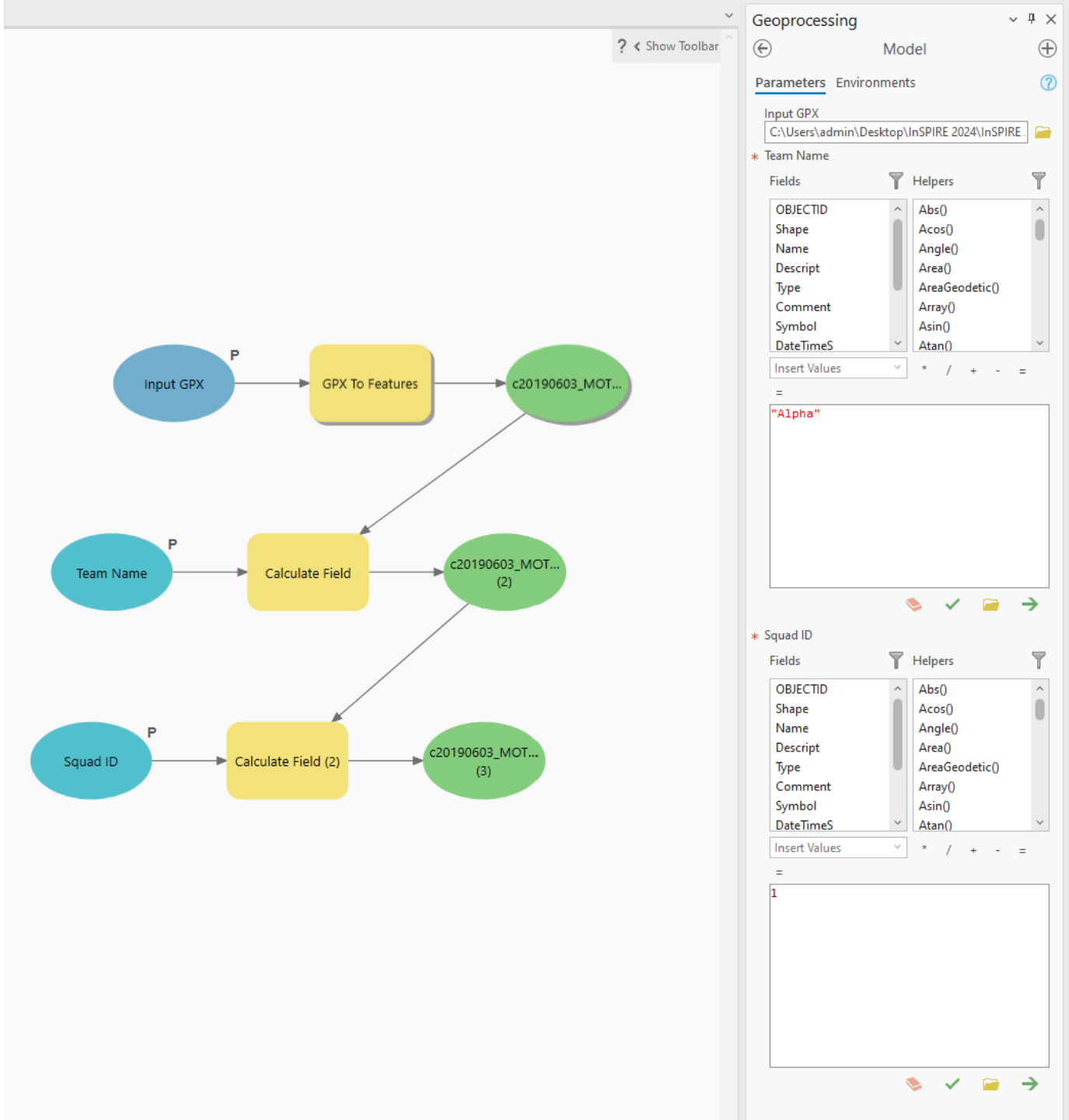


- Like before, right-click on the new Expression variable and choose Parameter. Then save the model and try opening it in the Geoprocessing pane once again
- This time you should notice that the whole field calculate expression appears. This is a limitation of Modelbuilder in that it is difficult to set up a simple text input

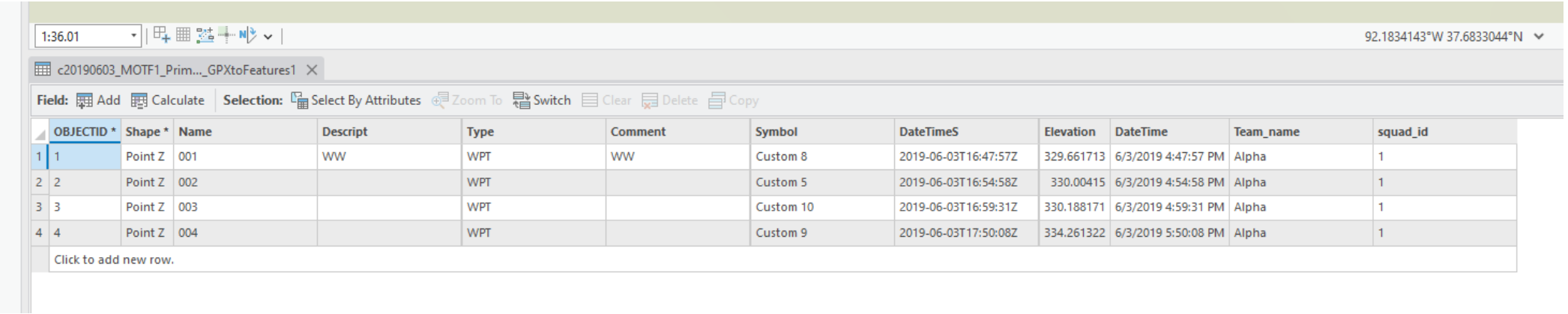




- Rename the first Expression to “Team Name,” then repeat the last few steps to do the same for Squad ID. When complete your model and geoprocessing pane UI should look like this
- Try running the tool with some provided test data, located in the InSPIRE 2024\InSPIRE 2024 Pro Project\example_data folder. Expored data should go to the Default geodatabase



- Examine the output data. You'll notice that the features have been created and the Team Name and Squad ID fields have been added to the attributes, calculated to whatever you put in



	OBJECTID *	Shape *	Name	Descript	Type	Comment	Symbol	DateTimeS	Elevation	DateTime	Team_name	squad_id
1	1	Point Z	001	WW	WPT	WW	Custom 8	2019-06-03T16:47:57Z	329.661713	6/3/2019 4:47:57 PM	Alpha	1
2	2	Point Z	002		WPT		Custom 5	2019-06-03T16:54:58Z	330.00415	6/3/2019 4:54:58 PM	Alpha	1
3	3	Point Z	003		WPT		Custom 10	2019-06-03T16:59:31Z	330.188171	6/3/2019 4:59:31 PM	Alpha	1
4	4	Point Z	004		WPT		Custom 9	2019-06-03T17:50:08Z	334.261322	6/3/2019 5:50:08 PM	Alpha	1
	Click to add new row.											

Choose the output name and location

- The last part this will cover is selecting the output location instead of going to the default. In the model, right-click on each current parameter and select Parameter once again. This will turn it off. Then select and re-enable parameters for each variable in the following order:

1. Input GPX
2. Output Feature Class
3. Team Name
4. Squad ID

- This will change the variable order in the geoprocessing UI

Next step ideas

- This tool covers a very basic GPX to features workflow. There are additional tasks that can be added on to make this production ready. Below are some ideas. Think about how you would tackle these problems and what tools you would use, then use what you learned to implement them into the model

- Tracklog processing (feature ‘type’ is TWPT instead of WPT)
- Symbol calculation (take the calculation from the Python section and implement it in the model)
- Append data into another database or Feature Layer

Python and Arcpy

Building a geoprocessing tool with Python

Python terminology & key foundations

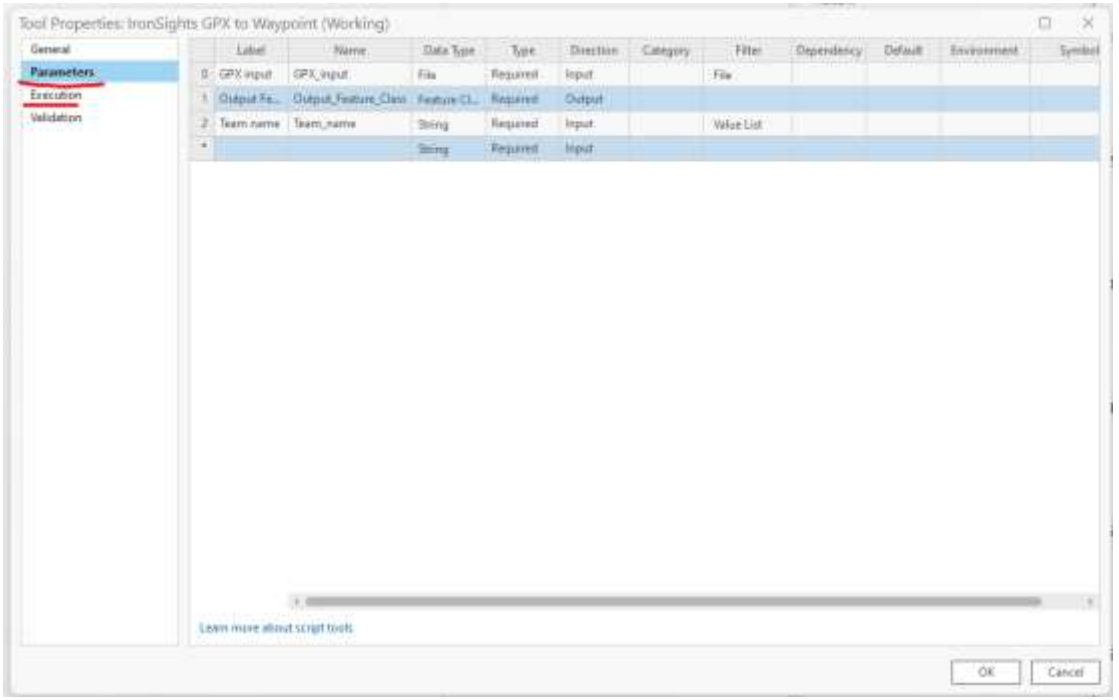
- Libraries – modules of commands that can be brought in and utilized in Python
  - “Import” and library name are usually found at the top of the script
  - Any time you see the library name, it means it’s calling a specific command
  - Library is the building, command is the book
  - Arcpy, OS, Pandas, NumPy, and TKinter are popular and appear in my example scripts

- Variables – Containers for storing data values, usually seen with an equal sign
  - Example: x = 1, x is the variable. Anywhere that X is accessed is where the number 1 is located
  - If the script was then print(x), the number 1 would be printed

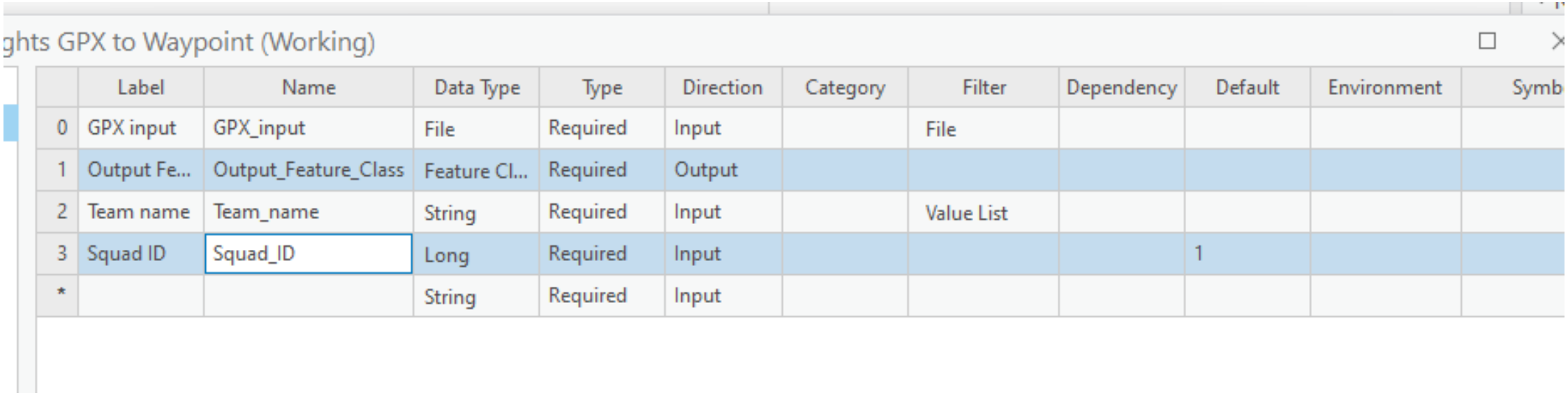
- Counting in programming starts at 0 instead of 1! This is important for parameters in ArcPy

- IDE – Program to edit code
  - For demo, I am using the built-in script editor

- As part of the template, I set up part of the script to begin. It is located in the default toolbox called “IronSights GPX to Waypoint”. This script already brings in the library needed, has a few parameters set up already, and has the symbol calculation embedded in the tool.
- Right-click on the script and click Properties. The “Parameters” and “Execution” sections are where we will be spending most of our time



- For this example, we will start with the parameters.
- Like in Modelbuilder, Parameters are the user inputs in the tool that appear in the geoprocessing pane. In a Python tool, we have a lot more control over what the parameters look like
- To add a parameter, click the empty row at the bottom of the table. Add a Squad ID field. Then set the Data Type to Long, and set the default to “1”



	Label	Name	Data Type	Type	Direction	Category	Filter	Dependency	Default	Environment	Symb
0	GPX input	GPX_input	File	Required	Input		File				
1	Output Fe...	Output_Feature_Class	Feature Cl...	Required	Output						
2	Team name	Team_name	String	Required	Input		Value List				
3	Squad ID	Squad_ID	Long	Required	Input				1		
*			String	Required	Input						

- Open the Execution tab. I already added some parts to help with the development.

- Take a second to read through the script as it is. Lines with several “#” signs are comments, or lines that are not read by the script, effectively working as notes within the script

- The script, and most other python scripts, are broken up into sections for easy reading and logical comprehension. They may not always be written out this way, but this is a good start. Writing comments through your script is always a good idea for both you and anyone who may be using your script in the future.
- The next slide offers more information about each broken out section

```
1  ######## Bring in the needed Python libraries
   import arcpy

2  ######## Input parameters from the Pro tool user interface
   gpx = arcpy.GetParamaterAsText(0)
   OutputFeatureClass = arcpy.GetParamaterAsText(1)
   team_name= arcpy.GetParamaterAsText(2)

3  ######## GPX to Features

   ######## Team name calculation
4  arcpy.management.CalculateField(
       in_table=OutputFeatureClass,
       field="team_name",
       expression=f'"{team_name}"',
       expression_type="PYTHON3",
       code_block="",
       field_type="TEXT",
       enforce_domains="NO_ENFORCE_DOMAINS"

5  ######### Squad ID calculation

6  ######### Symbol calculation
   arcpy.management.CalculateField(
               in_table=f"{OutputFeatureClass}",
               field="Symbol",
               expression="updatevalue(!Symbol!)",
               expression_type="PYTHON3",
               code_block="""def updatevalue(value):
               if value == "Custom 0":
                   return "unaffected"
               if value == "Custom 1":
                   return "minor"
```

1. This section is where libraries are imported. They are almost always imported in at the beginning of the script

2. These are the parameters being brought into the script. The next slide will cover how the variables work and brought into the script

3. Placeholder for the GPX to Features tool

4. Field calculate tool for Team name. We will dive closer into how the tools are brought in in a later slide

5. Placeholder for the Squad ID calculation

6. Pre-build calculation for the symbol field

7. (Off screen) Adds newly created feature class to map

User Parameters

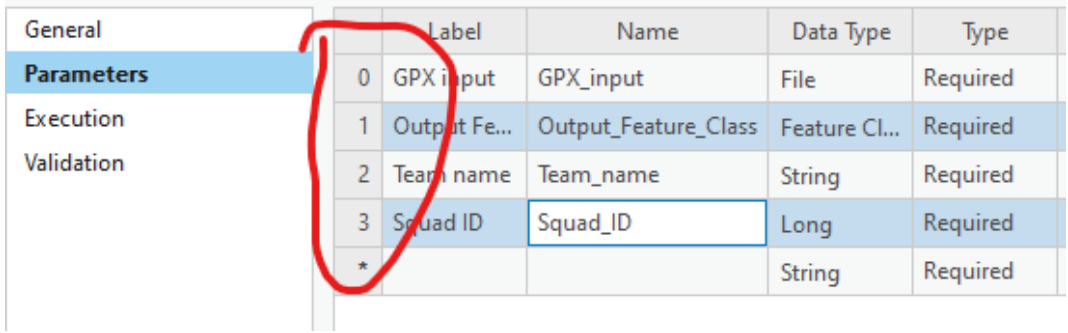
```
####### Input parameters from the Pro tool user interface
gpx = arcpy.GetParameterAsText(0)
OutputFeatureClass = arcpy.GetParameterAsText(1)
team_name= arcpy.GetParameterAsText(2)
```

- These are the user parameters being brought in from the UI within Pro into the script. These are examples of script variables

- Arcpy.GetParameterAsText() is the most common variable type you will see with parameters in custom tools. The number in the paratheses indicates what parameter this variable is pulling from (0 being the first variable in the Parameters tables, 1 being the second, etc). This will pull in the parameter as a string. Now, throughout the script, any time you see the variable, you should know that that attached user parameter is being placed in that line

- You can also use arcpy.GetParameter() if you are pulling a number or list

- Easiest way to get the parameter number is from the parameters tab. Whatever parameter you are pulling, look at the number on the left side



- Add a new parameter following the scheme in lines for Squad ID, then go back to the script and copy & paste the team name variable
- Modify the variable to say “Squad_ID”
- Your new parameter should be:

squad_id = arcpy.GetParameterAsText(3)

```
####### Input parameters from the Pro tool user interface
gpx = arcpy.GetParameterAsText(0)
OutputFeatureClass = arcpy.GetParameterAsText(1)
team_name= arcpy.GetParameterAsText(2)
squad_id = arcpy.GetParameterAsText(3)
```

Tools

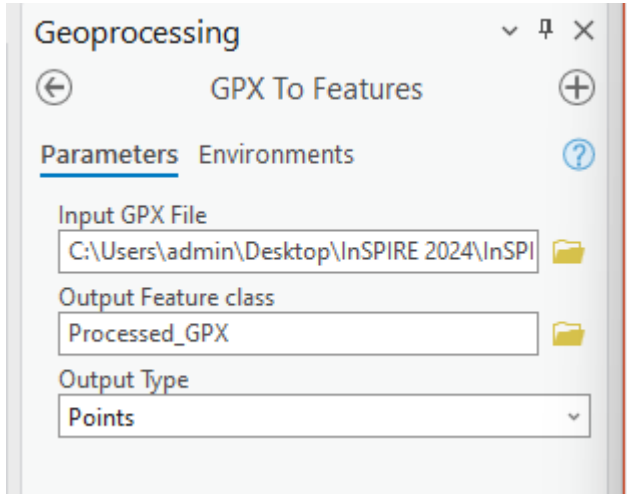
- Tools are brought in from Pro using the ArcPy library. Looking over the tool, you can see that the variables are laid out similarly to how they would be if you opened the tool as is in the geoprocessing pane

```
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="team_name",
    expression=f'"{team_name}"',
    expression_type="PYTHON3",
    code_block="",
    field_type="TEXT",
    enforce_domains="NO_ENFORCE_DOMAINS"
```

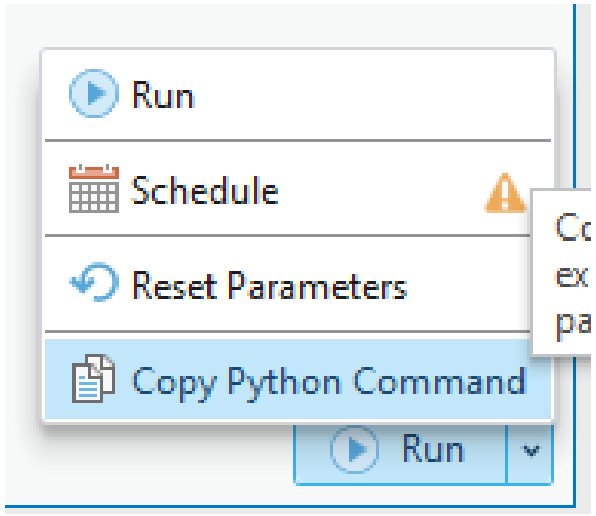
- Notice the input table is “OutputFeatureClass”. This is the parameter pulled in from line 6 OutputFeatureClass = arcpy.GetParamaterAsText(1)
- Team_name is also pulled in, but MUST be brought in through the format process (see slides 43 and 44)

Adding new tools

- You might think that adding new tools into the script is challenging, but Pro allows us to do it very easily. Click OK in the tool properties and open the GeoProcessing pane
- Search for GPX to Features and fill out the tool UI as you usually would (You can find some example data in InSPIRE 2024 Pro Project\example_data folder)



- Before you hit Run, click the dropdown arrow
- Then select Copy Python Command



- Return to the Execution tab of your script (toolbox – right click on the script – Properties – Execution)
- Click under the GPX to Features comment and paste it into the Python command. This copies the python command plus the input parameters. To get the script to work for us, we will need to do some editing

```
####### Bring in the needed Python libraries
import arcpy

####### Input parameters from the Pro tool user interface
gpx = arcpy.GetParameterAsText(0)
OutputFeatureClass = arcpy.GetParameterAsText(1)
team_name= arcpy.GetParameterAsText(2)

####### GPX to Features
arcpy.conversion.GPXtoFeatures(
    Input_GPX_File=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\example_data\20190603_MOTF1_PrimarySearch_Winters_.gpx",
    Output_Feature_class=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\Default.gdb\c20190603_MOTF1_PrimarySearch_Winters__GPXtoFeatures2",
    Output_Type="POINTS"
)
####### Team name calculation
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
```

- In the GPX to Features command, make the following changes:

1. Input_GPX_File line: remove everything **between the equal sign and the first comma**

```
####### GPX to Features
arcpy.conversion.GPXtoFeatures(
    Input_GPX_File=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\example_data\20190603_MOTF1_PrimarySearch_Winters_.gpx",
    Output_Feature_class=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\Default.gdb\Processed_GPX",
    Output_Type="POINTS"
)
```

```
####### GPX to Features
arcpy.conversion.GPXtoFeatures(
    Input_GPX_File=,
    Output_Feature_class=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\Default.gdb\Processed_GPX",
    Output_Type="POINTS"
)
```

- Between the equal sign and the first comma, place “gpx”
- Do the same for the next line called Output_Feature_class, changing it to “OutputFeatureClass”

- The command should look like this when done:

```
####### GPX to Features
arcpy.conversion.GPXtoFeatures(
    Input_GPX_File=gpx,
    Output_Feature_class=OutputFeatureClass,
    Output_Type="POINTS"
)
```

- What we did was utilize the variables set earlier in the script and place them into the command's parameters. This same thing can be done throughout the script

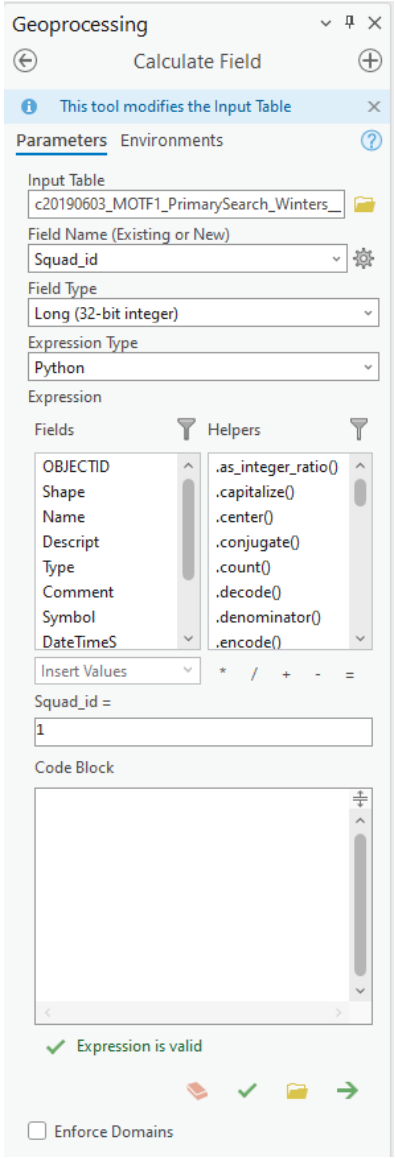
```
####### Bring in the needed Python libraries
import arcpy

####### Input parameters from the Pro tool user interface
gpx = arcpy.GetParameterAsText(0)
OutputFeatureClass = arcpy.GetParameterAsText(1)
team_name= arcpy.GetParameterAsText(2)
squad_id = arcpy.GetParameterAsText(3)

####### GPX to Features
arcpy.conversion.GPXtoFeatures(
    Input_GPX_File=gpx,
    Output_Feature_class=OutputFeatureClass,
    Output_Type="POINTS"
)
```

Running your new tool

- Click OK when done editing. Then double-click on the script in the toolbox to open it in the geoprocessing pane.
- Fill out the parameters of the tool using a sample GPX file. The rest of the parameters can be left as the default. Then click Run. When finished, find that new feature class in your default GDB and open the Attribute table
  - You may need to refresh the GDB
- The features were created, the Symbol field was changed to match the SARCOP symbol code, and Team Name was calculated

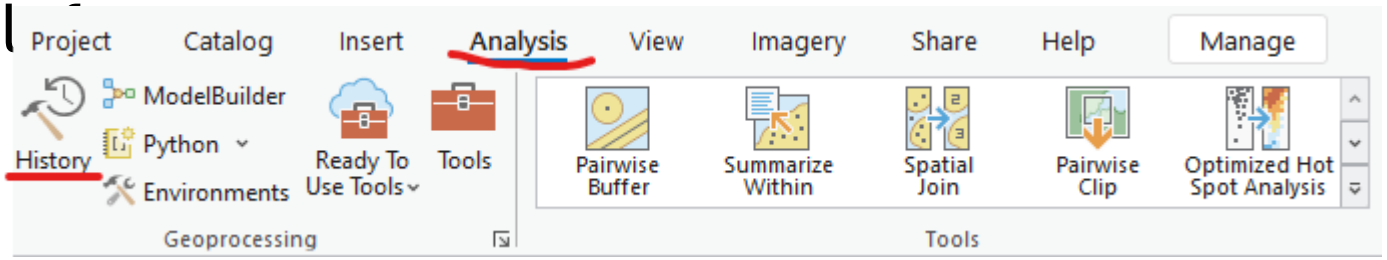
Another way to add commands

- There are a few ways a tool command can be added to the script. The first way we already covered. The second way is through the geoprocessing history
- First, head back to the geoprocessing pane and search for, then run a field calculation for Squad ID using the newly created feature class
- This time, click run and let the tool finish

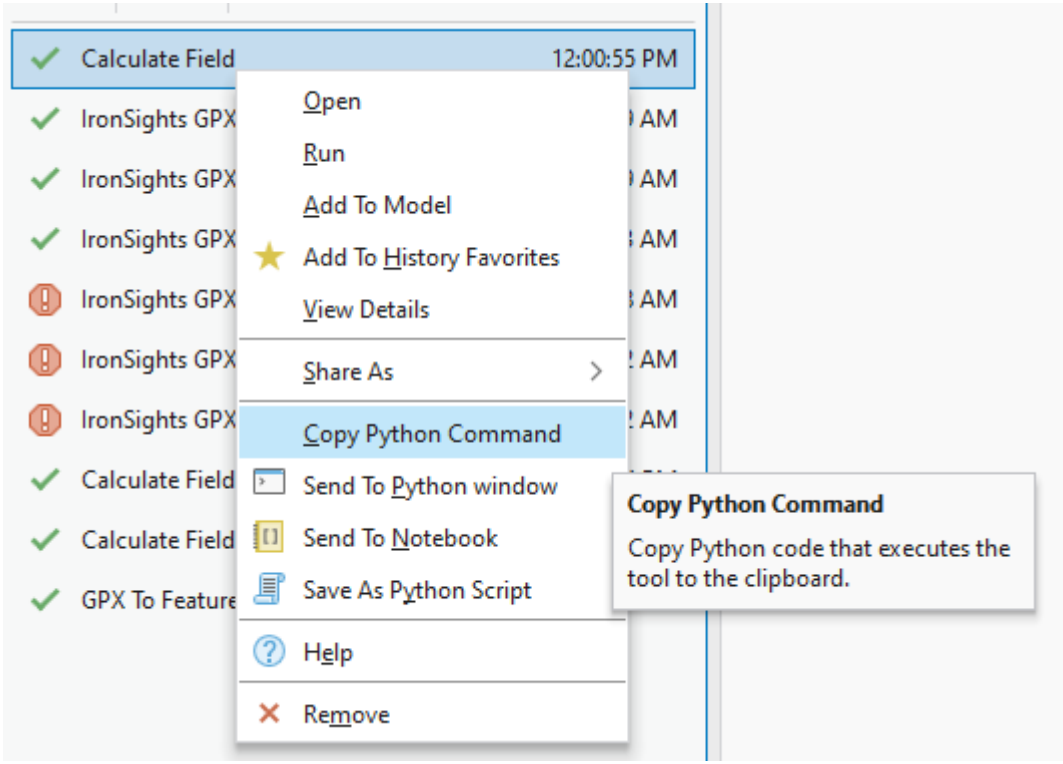


Another way to add commands

- Go to Analysis and History on the far



- This opens the history of what tools you have ran in this project. Select and right-click on the latest Calculate Field, then chose the Copy Python Command option



- Like before, open the script's Execution page. Find the Squad ID comment and paste in the python command

```
    input_GPX_file=gpx,
    Output_Feature_class=OutputFeatureClass,
    Output_Type="POINTS"
)
####### Team name calculation
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="team_name",
    expression=f'"{team_name}"',
    expression_type="PYTHON3",
    code_block="",
    field_type="TEXT",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
######### Squad ID calculation
arcpy.management.CalculateField(
    in_table=r"C:\Users\admin\Desktop\InSPIRE 2024\InSPIRE 2024 Pro Project\Default.gdb\c20190603_MOTF1_PrimarySearch_Winters__GPXtoPoint11",
    field="Squad_id",
    expression="1",
    expression_type="PYTHON3",
    code_block="",
    field_type="LONG",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
######### Symbol calculation
arcpy.management.CalculateField(
            in_table=f"{OutputFeatureClass}",
            field="Symbol",
            expression="updatevalue(!Symbol!)",
            expression_type="PYTHON3",
            code_block="""def updatevalue(value):
            if value == "Custom 0":
                return "unaffected"
```

- Change the in table and expression fields to “OutputFeatureClass” and “squad_id” respectively

```
,
######### Squad ID calculation
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="Squad_id",
    expression=squad_id,
    expression_type="PYTHON3",
    code_block="",
    field_type="LONG",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
```

- Click OK and try running your tool once again
- Make sure to keep the commas in the command. Removing these will make the script not work!
- In this case because we are working with an integer, we do not have to have the nested quotes and F string formatting in the experssion

Calculate field - strings

- You may notice the expression is different between the Squad ID and Team Name. This is because Team name uses a string, while Squad ID uses an integer
- Like in regular Field Calculate, you must have the entry in quotes to place in a string value. The same rule applies here, meaning the expression must be in two sets of quotes

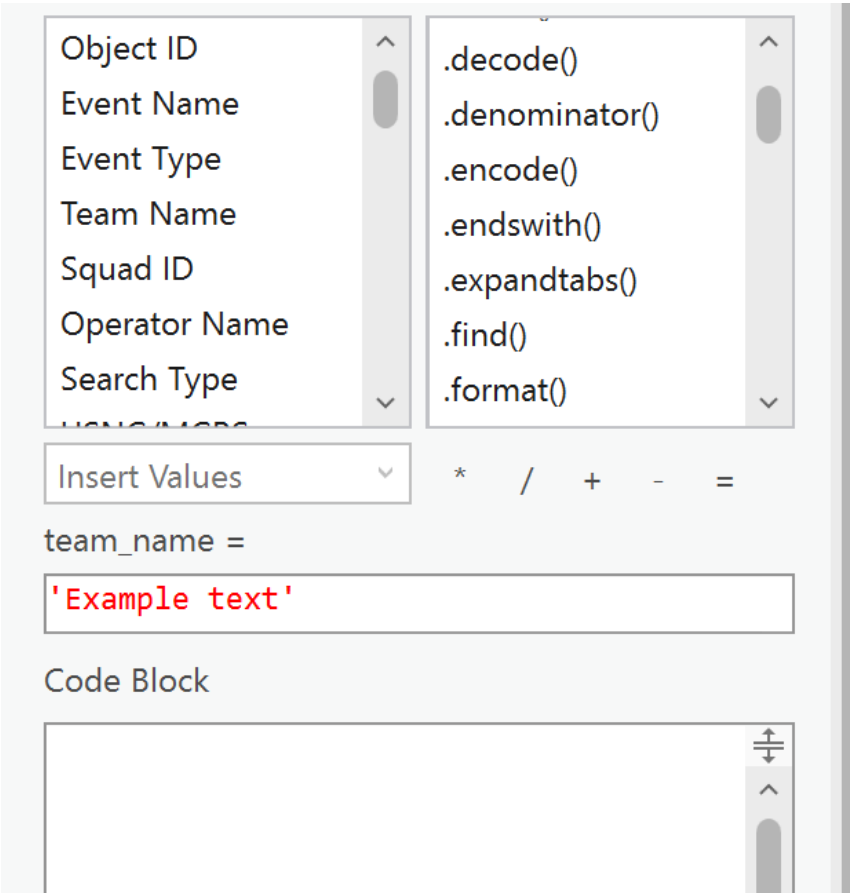
```
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="team_name",
    expression="'Example Text'",
    expression_type="PYTHON3",
    code_block="",
    field_type="TEXT",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
```

expression="'Example Text'",

Calculate field - strings

- Just like in the regular calculate field, a string must be in quotes in the expression

```
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="team_name",
    expression="'Example Text'",
    expression_type="PYTHON3",
    code_block="",
    field_type="TEXT",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
```



Format strings

- In our script, we also use the “format string” (f string in this case) function in Python to plug in our variable
- Format strings allow us to insert a variable into an existing string without breaking the string. Placing a “f” before the string and {} where we want the variable to go
- Example:

```
X = “World”
print(f“Hello {x}”)
```

Output:

Hello World

- More about F strings here https://www.geeksforgeeks.org/formatted-string-literals-f-strings-python/#

Format strings

- In our script, Team Name is set up as a F string so that it can intake whatever string is chosen, therefore the command looks like:

```
######## Team name calculation
arcpy.management.CalculateField(
    in_table=OutputFeatureClass,
    field="team_name",
    expression=f"'{team_name}'",
    expression_type="PYTHON3",
    code_block="",
    field_type="TEXT",
    enforce_domains="NO_ENFORCE_DOMAINS"
)
```

```
expression=f"'{team_name}'",
```

- A similar setup is used when you have an expression block in a calculate field command. Notice how the function is set into several sets of quotes

- The best way to work with a calculate field that requires a code block is to set it up in the Calculate Field tool in the geoprocessing pane, then remove the whole command and replace it any time you need to make changes to it

```
######### Symbol calculation
arcpy.management.CalculateField(
            in_table=f"{OutputFeatureClass}",
            field="Symbol",
            expression="updatevalue(!Symbol!)",
            expression_type="PYTHON3",
            code_block="""def updatevalue(value):
            if value == "Custom 0":
                return "unaffected"
            if value == "Custom 1":
                return "minor"
            if value == "Custom 2":
                return "major"
            if value == "Custom 3":
                return "destroyed"
```

Raw String

- These look like:

Path = r’C\users\admin\Documents\file.txt’

- Distinguished by the letter “r” in front of the string, similar to the F string
- Ignore Escape Character Sequences
- Reads the string “as is”
- Without it, file paths would be read incorrectly since they use forward slashes as they are commonly used for text formatting
- Print(r”C:\Users\admin\Documents”) will print correctly.
- Print(”C:\Users\admin\Documents”) will throw an error

Automatically add data to the map

```
project = arcpy.mp.ArcGISProject("Current")
m = project.activeMap
m.addDataFromPath(OutputFeatureClass)
```

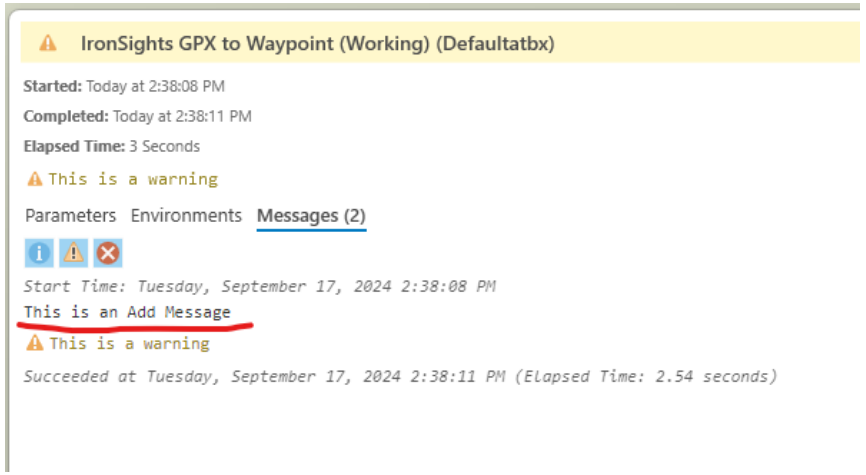
- Add these three lines at the bottom of your script to add the data to your current map
- These lines are available at the bottom of the provided script

Messages & Errors

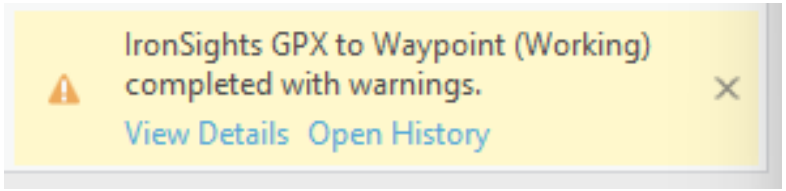
- Sometimes it is appropriate to add in messages to help you understand where your script is at, add messages to help debug, or cause the script to fail on purpose with a more helpful error message.
- Arcpy.AddMessage() – Adds a general message in the geoprocessing pane while the tool is running
- Arcpy.AddWarning() – Makes the tool throw a yellow message when tool is finished running
- Arcpy.AddError() – Makes tool run a red message when tool is finished
- Arcpy. SetProgressorLabel() – Adds a label in the processor to tell the user what step the tool is at

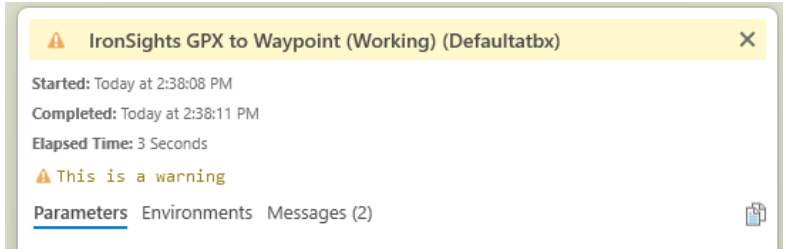
Messages & Errors

- Arcpy.AddMessage()

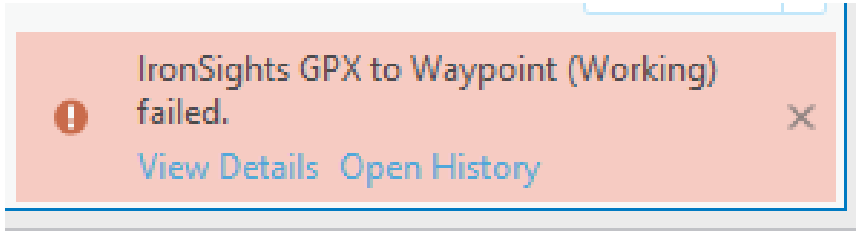


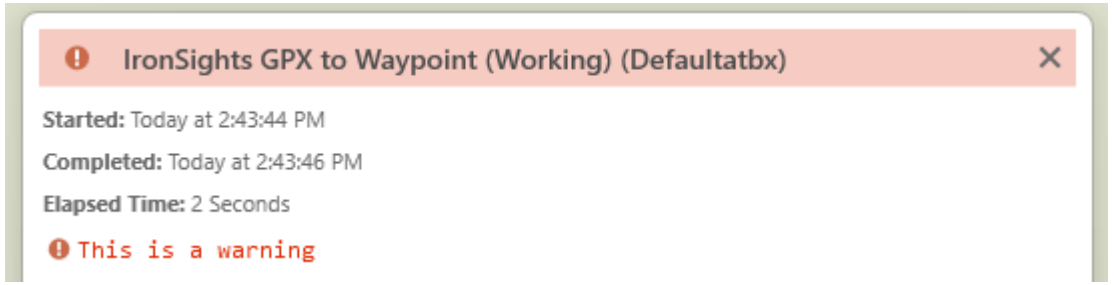
Arcpy.AddWarning()



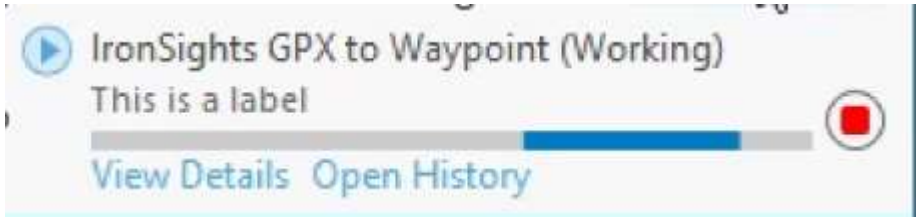


- Arcpy.AddError()





- Arcpy.SetProgressorLabel()



Resources

- W3 schools – python tutorials and samples
- Python courses by freeCodeCamp.org online and on YouTube
- Advanced Python Scripting for ArcGIS Pro by Paul A. Zandbergen – Awesome book I used frequently

THANKS!

afackler@publicsafetygis.org

napsgfoundation.org

@napsgfoundation