Building a conflation tool with Python

- Started with streams and wetlands as they are the most common features and have more variables regarding how features are drawn from their representative points
- Basic logic:
  - How can we leverage the limited information given to properly draw the features automatically as opposed to traditional digitizing?
  - Make a feature layer of the gps point data based on each unique GAI_ID (each unique feature)
  - This way, each feature can be looked at, drawn, and attributed individually
  - Run through preventative tests first, then run through main analysis

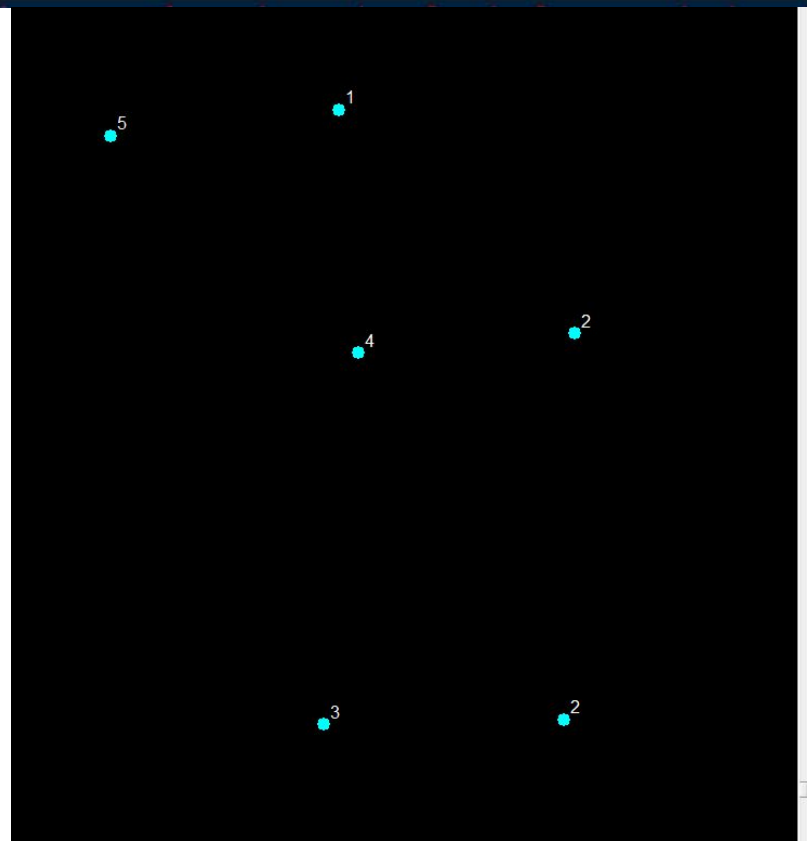
Preventative Analysis

- To save processing time, the following checks are performed before the main section of the script is executed.
- For each unique feature ID (i.e. SPA-NGP-001):
  - “Does each flag number occur only once (if center bank)?”
  - “Are there at least 2 (for stream or line features) or 3 (for wetland or polygon features) flags for the feature to be drawn from?”
  - “Do all features start with flag 1 and increase sequentially by 1”
- If one of the above statements evaluates to False, the ID is stored in a post processing manual review list, and the feature will not be drawn.

Preventative Analysis I

- Does each flag number occur only once (wetland example)? This feature was not drawn as there are two flags with the same number…possibly something wrong.

```
#Make a blank list for flag nums to make sure each number occurs only once. If more than one flag has the same number, add to Wetland_IDs_REVIEW.
wetland_lyr_flagnums = []
with arcpy.da.SearchCursor(wetland_lyr, "Flag_num") as cursor:
    for row in cursor:
        if row[0] not in wetland_lyr_flagnums:
            wetland_lyr_flagnums.append(row[0])
        else:
            print("Feature " + str(ID) + " needs reviewed. More than one flag has the same number. Adding ID to review list")
            Wetland_IDs_REVIEW.append(ID)
            break
```

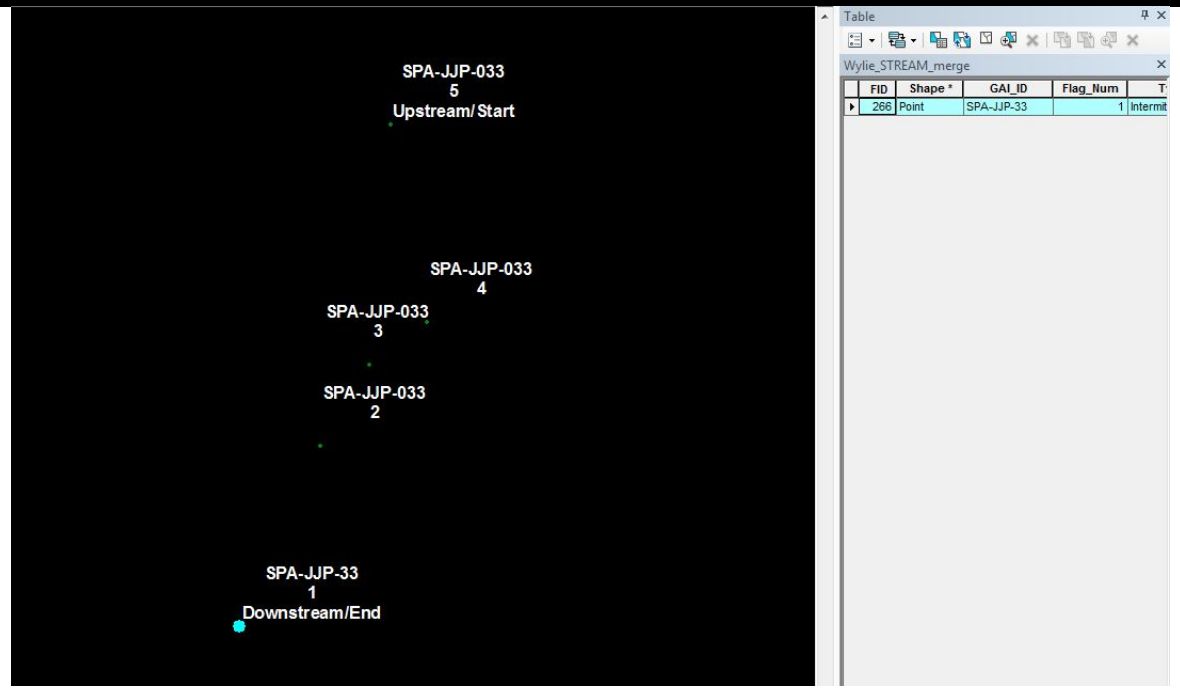


FID	Shape *	GAI_ID	Flag_Num	Type
626	Point	WPA-NGP-024	1	PEM
627	Point	WPA-NGP-024	2	PEM
628	Point	WPA-NGP-024	2	PEM
629	Point	WPA-NGP-024	3	PEM
630	Point	WPA-NGP-024	4	PEM
631	Point	WPA-NGP-024	5	PEM

Preventative Analysis II

- Are there at least 2 (for stream or line features) or 3 (for wetland or polygon features) flags for the feature to be drawn from? SPA-JJP-33 was not drawn as it is considered its own feature and there is only 1 flag (field typo).

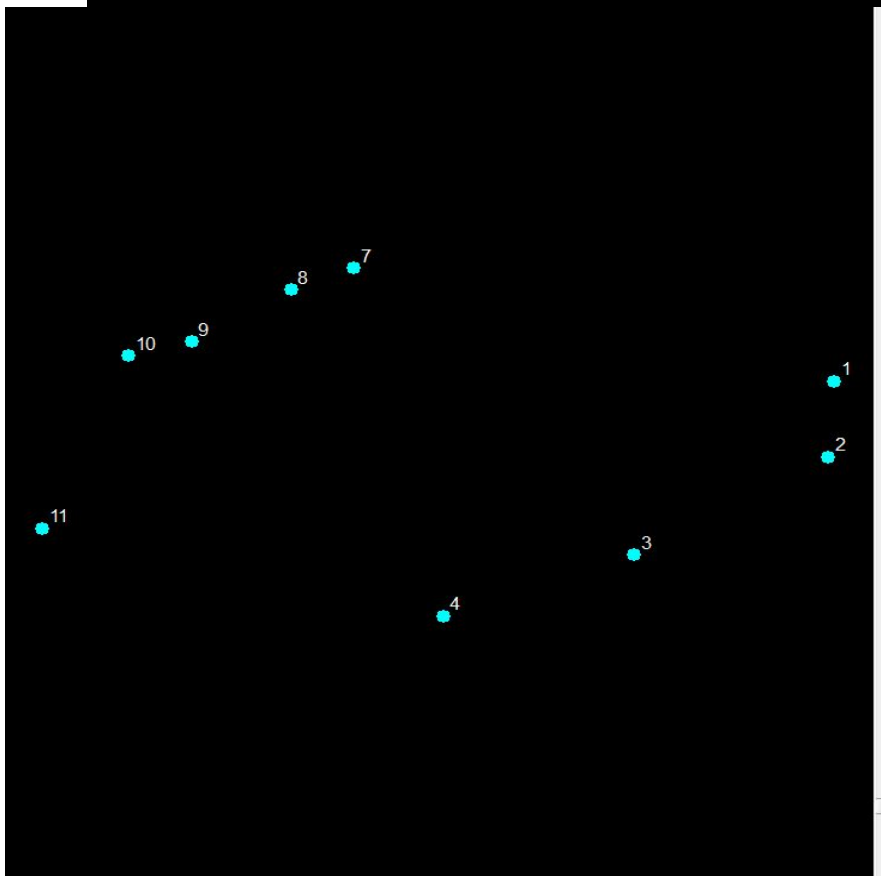
```
#Check to make sure there are at least two points for the feature to be drawn from.
#Test to make sure flags increase sequentially by an interval of 1. If not, its possible a flag was missed. Add to Stream_IDs_REVIEW if not.##############################################
length_flagnums = len(stream_lyr_flagnums)
if length_flagnums == 1 and ID not in Stream_IDs_REVIEW:
    print("Feature " + str(ID) + " consists of only one point. A stream cannot be drawn. Adding ID to review list.")
    Stream_IDs_REVIEW.append(ID)
max_flagnums = max(stream_lyr_flagnums)
#If biggest number in list flag nums is not also the length of the list, this implies that a flag number was skipped (flag nums do not all increase by interval of one)
if max_flagnums > length_flagnums and ID not in Stream_IDs_REVIEW:
    print("Feature " + str(ID) + " needs reviewed. Flag numbers do not all increase by interval of one. A flag may have possibly been skipped or querried out. Adding ID to review list.")
    Stream_IDs_REVIEW.append(ID)
```

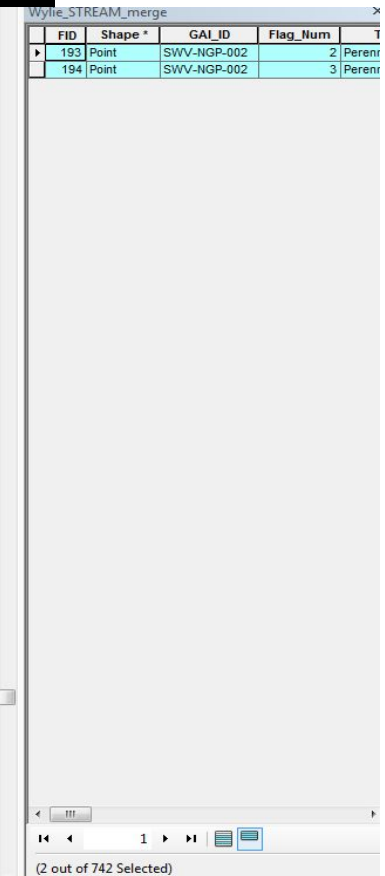


Preventative Analysis III

- Do all flags start at 1 and increase sequentially by one? Left example numerically skips flags. Right example missing flag 1.

```
#Check to make sure there are at least two points for the feature to be drawn from.
#Test to make sure flags increase sequentially by an interval of 1. If not, its possible a flag was missed. Add to Stream_IDs_REVIEW if not.##################################################
length_flagnums = len(stream_lyr_flagnums)
if length_flagnums == 1 and ID not in Stream_IDs_REVIEW:
    print("Feature " + str(ID) + " consists of only one point. A stream cannot be drawn. Adding ID to review list.")
    Stream_IDs_REVIEW.append(ID)
max_flagnums = max(stream_lyr_flagnums)
#If biggest number in list flag nums is not also the length of the list, this implies that a flag number was skipped (flag nums do not all increase by interval of one)
if max_flagnums > length_flagnums and ID not in Stream_IDs_REVIEW:
    print("Feature " + str(ID) + " needs reviewed. Flag numbers do not all increase by interval of one. A flag may have possibly been skipped or querried out. Adding ID to review list.")
    Stream_IDs_REVIEW.append(ID)
```





Wetland Workflow

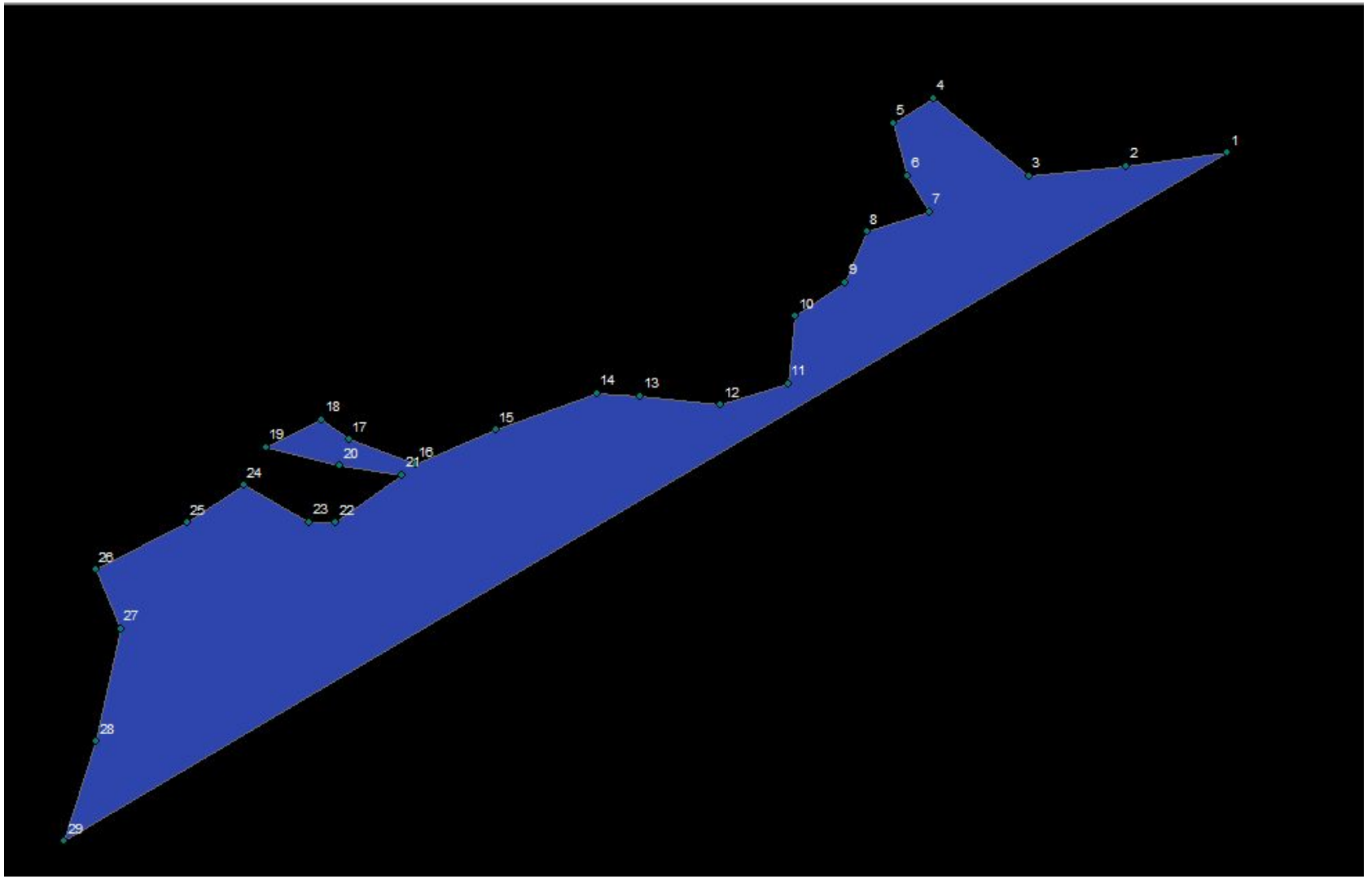
- Create a feature layer of the wetland points for each unique GAI_ID: **wetland_lyr**
- For each unique **wetland_lyr:**
- Draw the outline of the wetland using PointsToLine in order of flag number (close line): **wetland_outline**
- Make a polygon from **wetland_outline** using FeatureToPolygon: **wetland**
- Get a feature count of the derived feature **wetland.**
- If the feature count >= 2, this implies that the feature intersects itself and it must be drawn manually. It will be added to the post processing review list.
- If the feature count is not >=2,
    - Using a combination of Search and Update cursors, update the attributes of **wetland** with the attributes of **wetland_lyr**
        - Special case for “wetland_type” and “open end”. Skip to slide 21 for how attribute transfer works

Main section of Wetland Script (The comments aren't likely enough to explain every line, this is just for temporary reference).

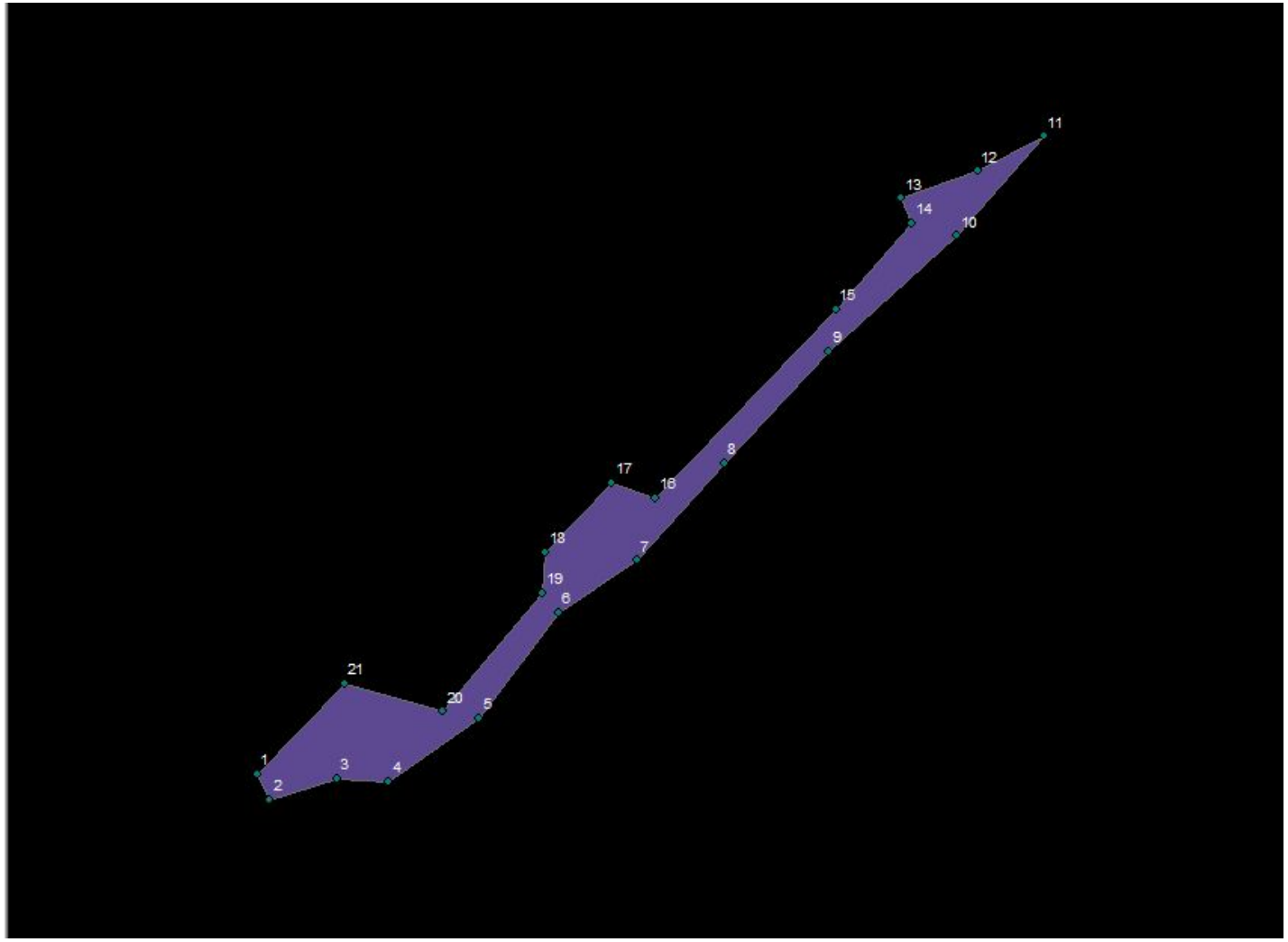
```
if ID not in Wetland_IDs_REVIEW:
    try:
        print(ID)
        wetland_outline = arcpy.PointsToLine_management(wetland_lyr, workspace + "\wetland_line_" + str(ID_replace_final) + "uID_" + str(unique_ID), Sort_Field = "Flag_num", Close_Line = "CLOSE")
        wetland = arcpy.FeatureToPolygon_management(wetland_outline, workspace + "\wetland" + str(ID_replace_final)+ "uID_" + str(unique_ID))
        unique_ID += 1
        wetland_parts = int(arcpy.GetCount_management(wetland).getOutput(0))
        if wetland_parts >= 2:
            print("Wetland consists of " + str(wetland_parts) + " parts and therefore intersects itself. Adding " + str(ID) + " to Wetland_IDs_REVIEW.")
            Wetland_IDs_REVIEW.append(ID)
            arcpy.DeleteFeatures_management(wetland)
        else:
            #Add attribute values to wetland from wetland_lyr
            for att in wetland_att_list:
                arcpy.AddField_management(*(wetland,)+att)
            #Initialize variable to store wetland type and open end.
            wetland_type = ''
            open_end = 'No'
            #For the wetland type in particular, we must confirm that each point in wetland_lyr has the same value (type). If so, the value will be used to populate the wetland_type field. If not,
            #the field will be left blank.
            #Initialize list to count if there is more than 1 wetland type for the wetland_lyr
            type1_holder = []
            with arcpy.da.SearchCursor(wetland_lyr, "Type_1") as cursor:
                for type1 in cursor:
                    if type1[0] not in type1_holder:
                        type1_holder.append(type1[0])
            if len(type1_holder) == 1:
                wetland_type = type1_holder[0]
            else:
                print('Wetland ' + str(ID) + ' is more than one type and its partitioning must be done manually.')
            open_end_holder = []
            with arcpy.da.SearchCursor(wetland_lyr, "Flag_type") as cursor:
                for i in cursor:
                    if i[0] == 'Open End' or i[0] == 'End/Open End':
                        open_end = 'Yes'
                        break

            #Update values of wetland with values from wetland_lyr points
            #Start editing
            edit = arcpy.da.Editor(workspace)
            edit.startEditing(False, True)
            edit.startOperation()
            update_fields = ['GAI_ID','Jurisdic','GPS_Date']
            with arcpy.da.SearchCursor(wetland_lyr, ['GAI_ID', 'Jurisdicti', 'GPS_Date']) as cursor1:
                for row1 in cursor1:
                    GAI_ID, Jurisdiction, GPS_Date = row1[0], row1[1], row1[2]
                    with arcpy.da.UpdateCursor(wetland, ['GAI_ID', 'Jurisdiction', 'GPS_Date']) as cursor2:
                        for row2 in cursor2:
                            for i in range(0,len(update_fields)):
                                row2[i] = row1[i]
                                cursor2.updateRow(row2)
            with arcpy.da.UpdateCursor(wetland,['Type', 'Open_End']) as cursor3:
                for row3 in cursor3:
                    if len(wetland_type) > 0:
                        row3[0] = wetland_type
                    row3[1] = open_end
                    cursor3.updateRow(row3)
            edit.stopOperation()
            edit.stopEditing(True)
            #arcpy.Append_management(wetland, ENV_GDB + "\Wetlands", "NO_TEST")
```

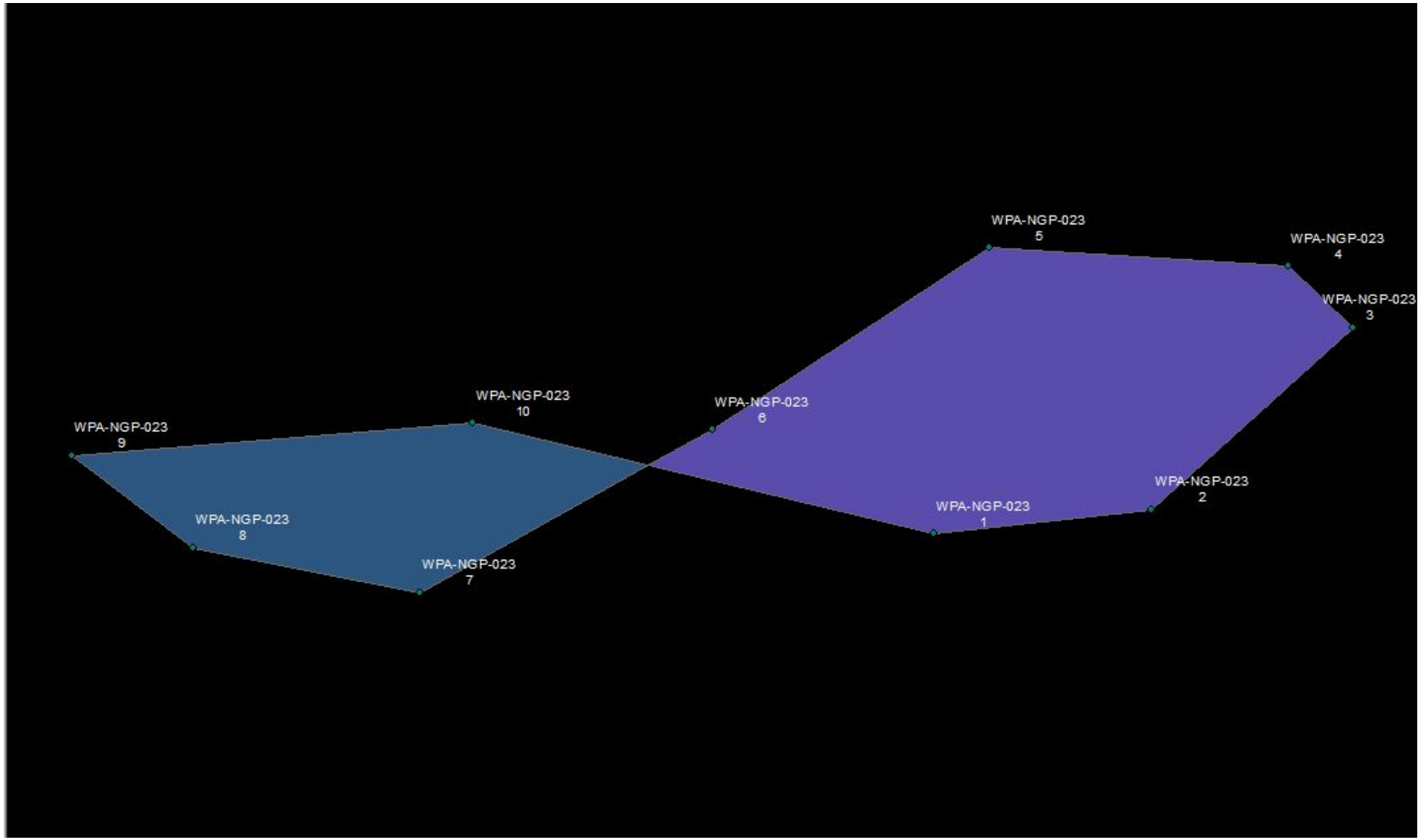
Wetland Workflow: Successfully drawn and attributed



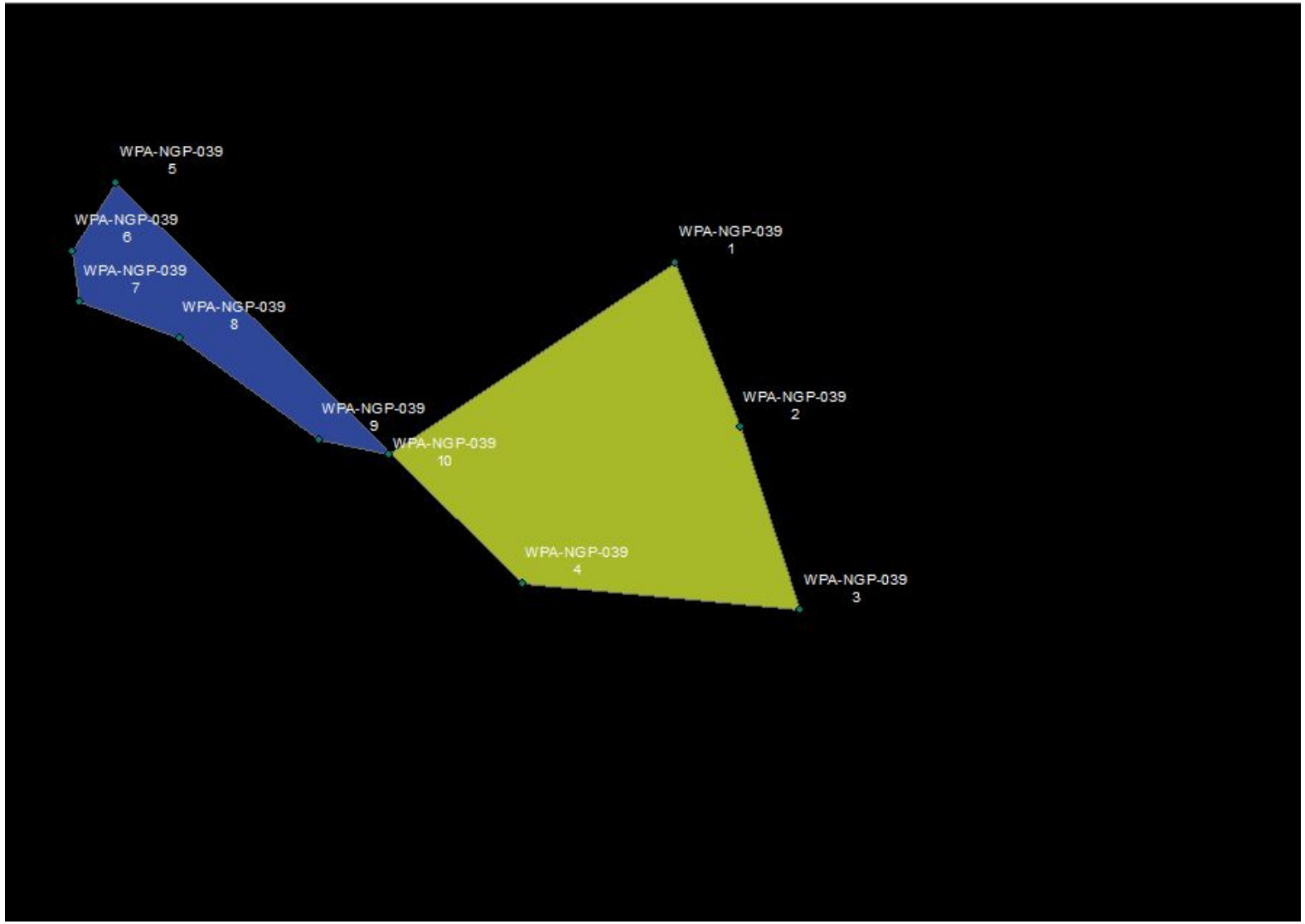
Wetland Workflow: Successfully drawn and attributed



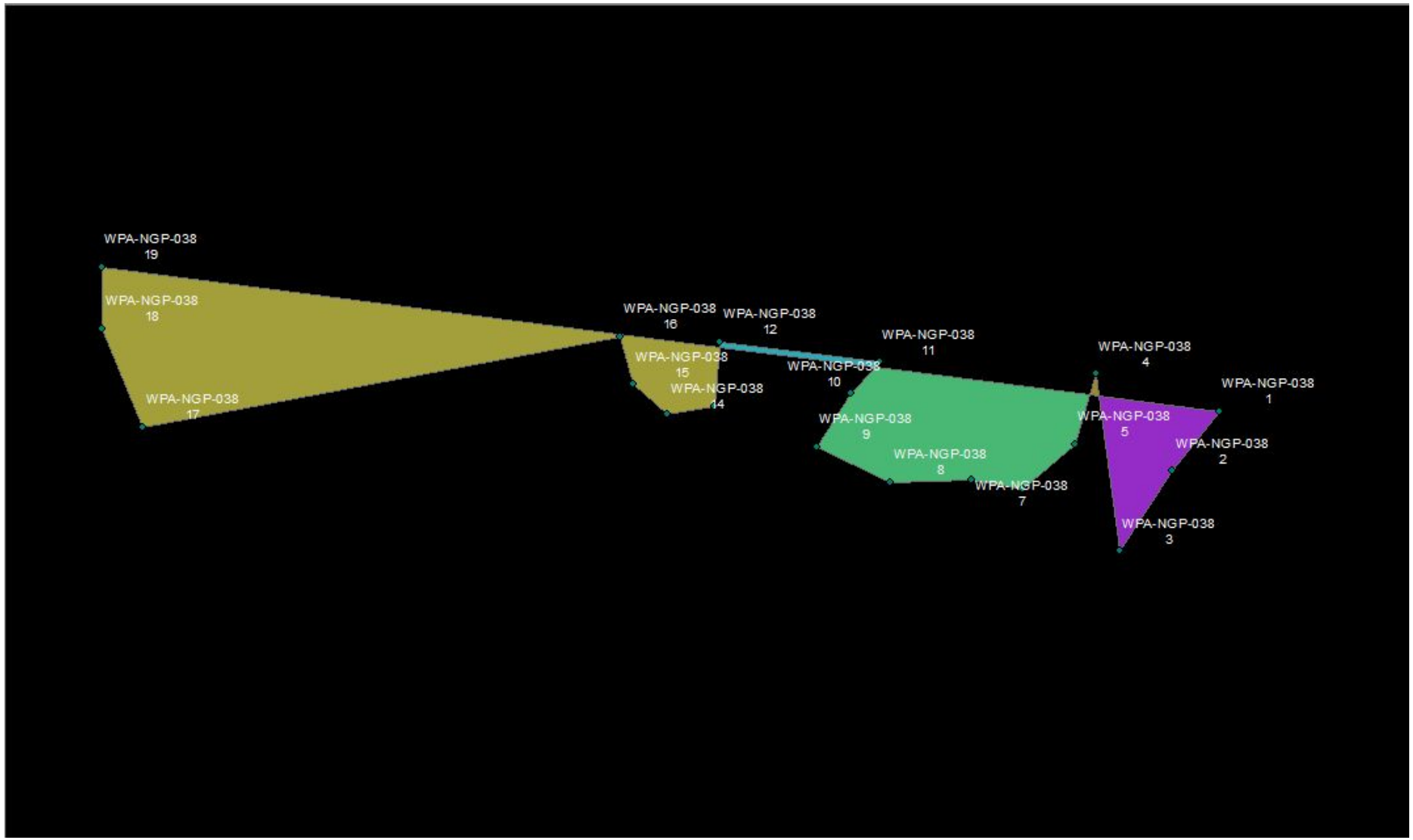
Wetland Workflow: Intersects itself and therefore will be deleted and have its GAI_ID added to a manual review list.



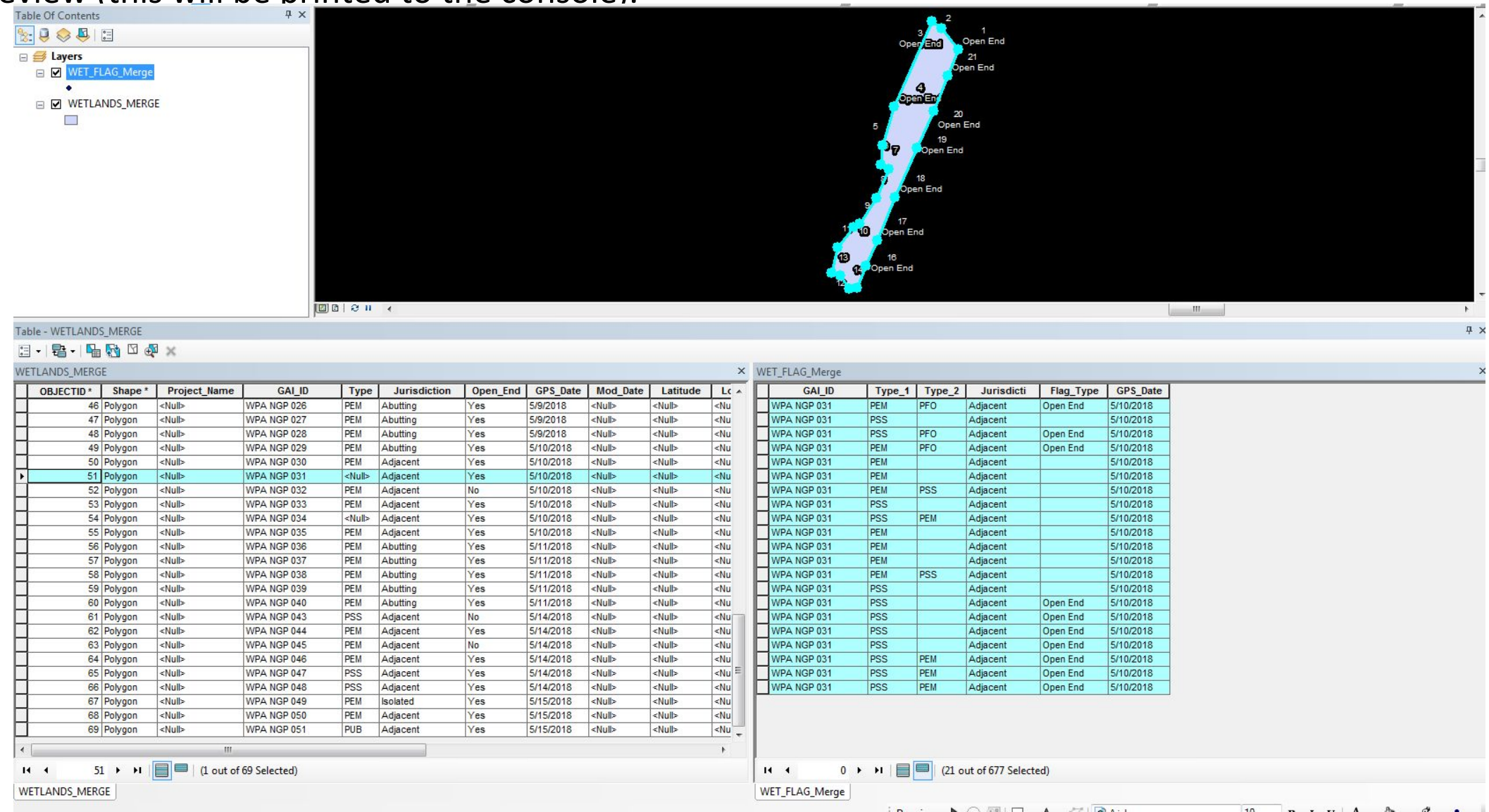
Wetland Workflow: Intersects itself and therefore will be deleted and have its GAI_ID added to a manual review list.



Wetland Workflow: Intersects itself and therefore will be deleted and have its GAI_ID added to a manual review list.



Wetland Workflow: Wetlands that need parsing (>1 value type in ‘Type_1’ field) but evaluate fine otherwise will be drawn and attributed, but will have the attribute ‘Type’ left blank for manual review (this will be printed to the console).



Open Ends

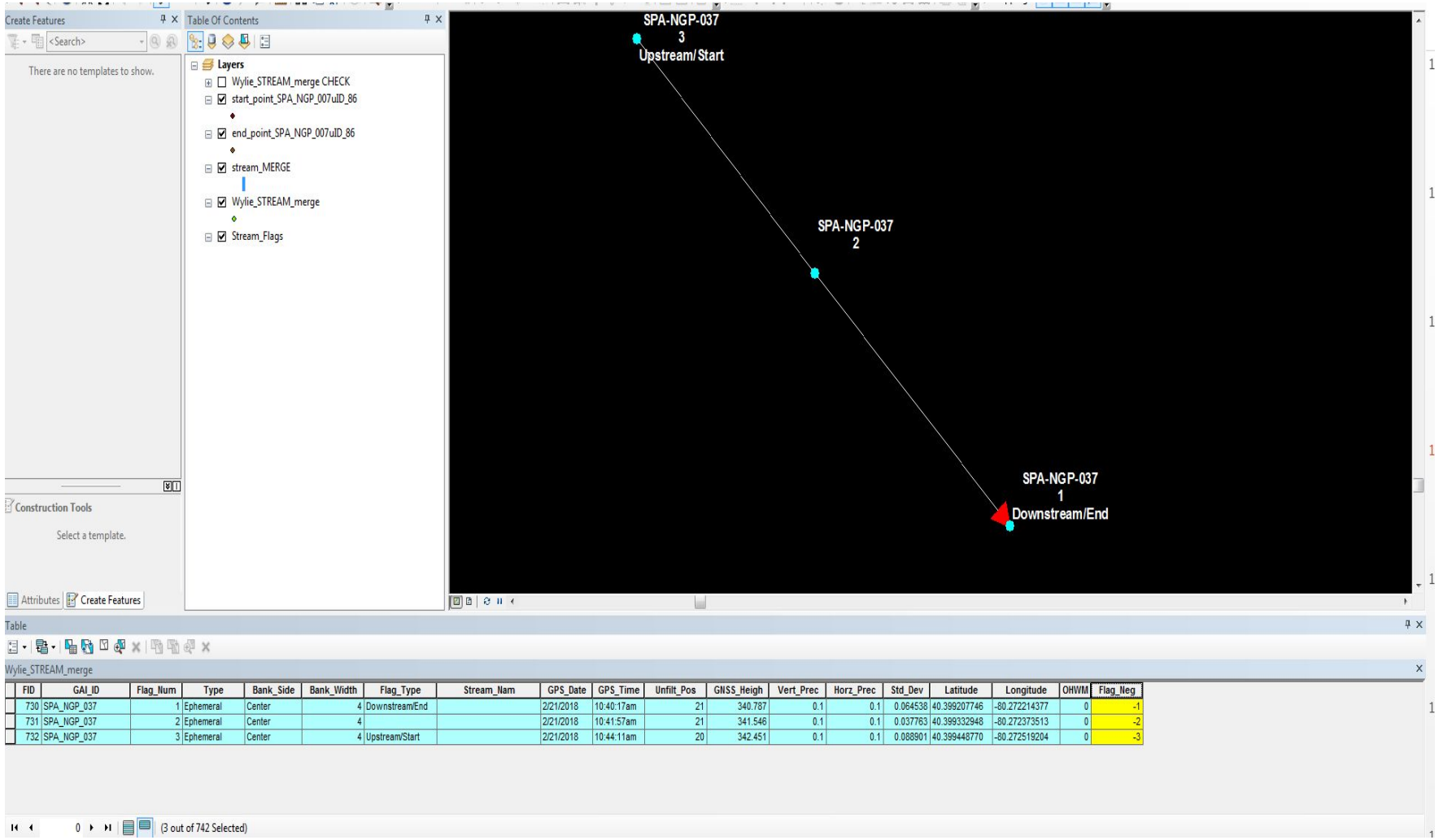
- Determine if **wetland** is open ended. If so,
- Create layer of just **open end flags** from wetland points of **wetland**
- Split **wetland** into its **lines** by vertices such that each **line** will have only two vertices (its first and last point).
- For each **line**:
  - Select by location the **open end flags** that intersect **line**
  - Get count of **open end flags**
    - If count of **open end flags** <2:
      - **Line** dismissed as non open end
    - Else:
      - **Line** added to **open ends**

Stream Workflow

- Create a feature layer of the stream flags for each unique GAI_ID: **stream_lyr**
- Draw stream/line from **stream_lyr** ***by*** Bank_Side ***in*** order of Flag_nums: **stream_line**
- Create a point feature on the start and end point of **stream_line** (this will create a set of points for each line): **start_point, end_point**
- Determine if there are one or two lines (banks)
- Evaluating **stream_line** (one line at a time if there are two), select by location the stream flags that are **identical** to **start_point, end_point**
- Get the value of the "Flag_type" field of the selected stream flag. If the value of the stream flag coincident with the **end_point** is "Upstream/Start" or "Headcut" ***OR*** if the value of the stream flag coincident with the **start_point** is "Downstream/End" or "Confluence", the stream has been drawn in the wrong direction based on these logical indicators.
    - If the line (drawn in the order of the flag numbers) is determined to be drawn in the wrong direction, this line will be deleted. A new field will be added to **stream_lyr** named "Flag_neg". It will be calculated as "Flag_num*-1". Since the points to line tool draws lines in order of least to greatest, following this field will draw the line in the opposite direction as each value is the inverse of the original.
    - Setting the logic in this manner prevents a "False" evaluation arising from an end point not attributed "down stream" or a start point not attributed "up stream" when the attribution is left blank (as long as we know one side, the other can be inferred).

Stream Workflow (Basic example): Drawing a stream in the appropriate direction using flag type

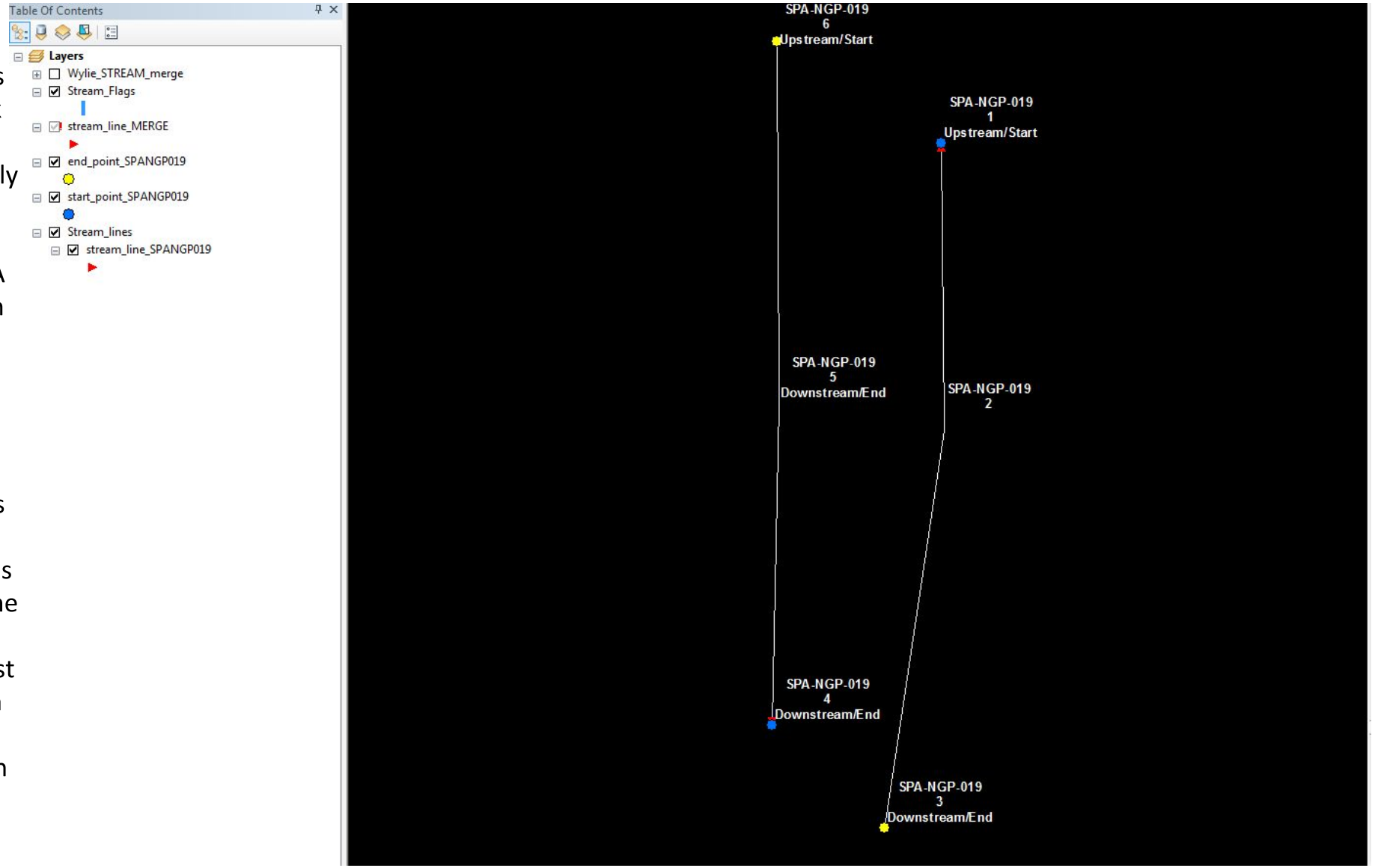
- The stream here was originally drawn flowing north by the Flag_num field. Once the stream was drawn, points were created on it's start and end locations (flags 1 and 3 respectively). The Flag_Type value for the flags coincident with the start and end points were then evaluated. Since the start point was on a "Downstream/End" flag and the end point was on an "Upstream/Start" flag, this evaluated such that the field highlighted in yellow was created, and the stream was redrawn in the order of this field from least to greatest, resulting in a line drawn in the opposite direction.



FID	GAI_ID	Flag_Num	Type	Bank_Side	Bank_Width	Flag_Type	Stream_Nam	GPS_Date	GPS_Time	Unfilt_Pos	GNSS_Heigh	Vert_Prec	Horz_Prec	Std_Dev	Latitude	Longitude	OHWM	Flag_Neg
730	SPA_NGP_037	1	Ephemeral	Center	4	Downstream/End		2/21/2018	10:40:17am	21	340.787	0.1	0.1	0.064538	40.399207746	-80.272214377	0	-1
731	SPA_NGP_037	2	Ephemeral	Center	4			2/21/2018	10:41:57am	21	341.546	0.1	0.1	0.037763	40.399332948	-80.272373513	0	-2
732	SPA_NGP_037	3	Ephemeral	Center	4	Upstream/Start		2/21/2018	10:44:11am	20	342.451	0.1	0.1	0.088901	40.399448770	-80.272519204	0	-3

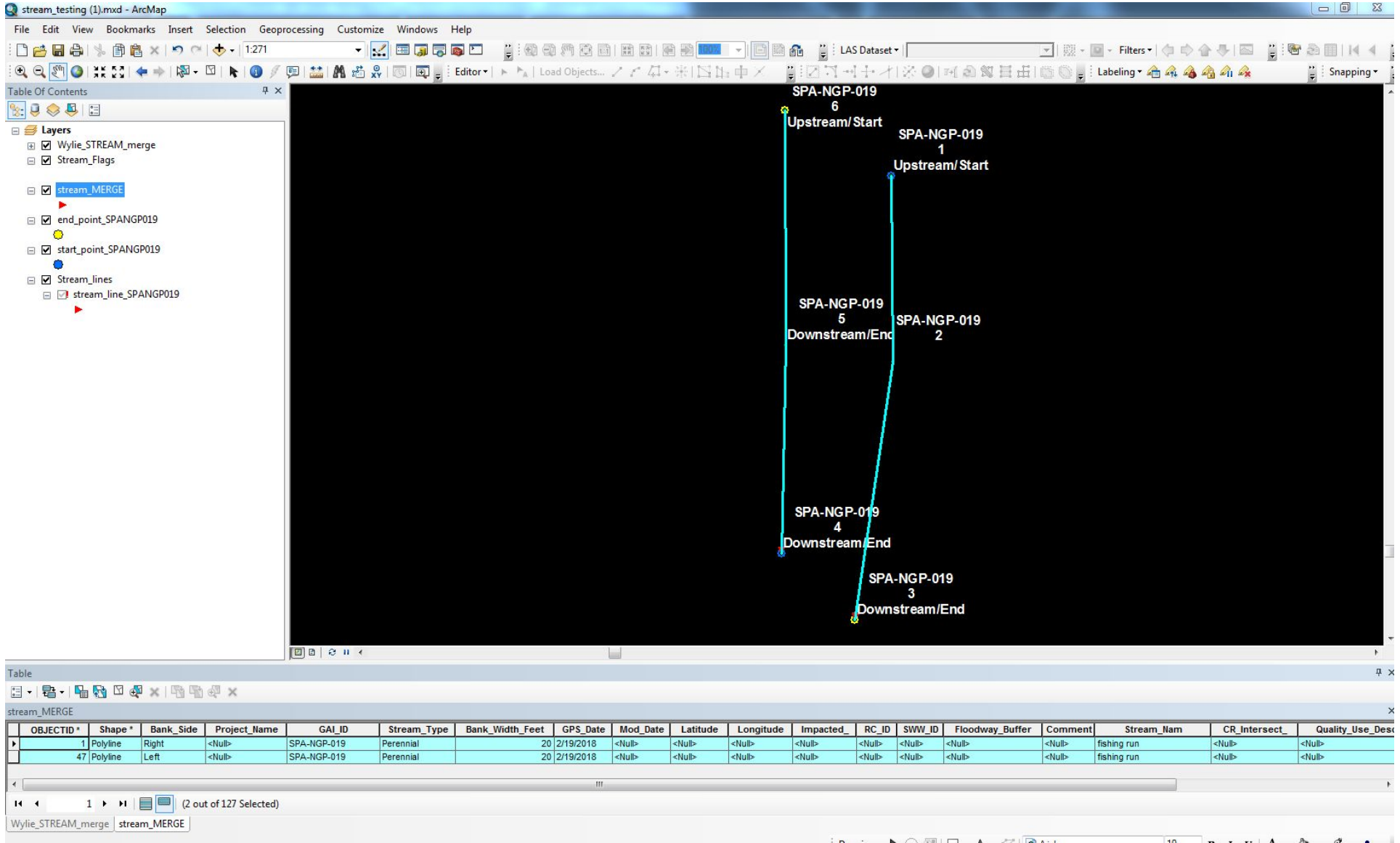
Stream Workflow (Complex example): Drawing a stream in the appropriate direction using flag type--Before

- (True direction is south) Before having the script evaluate bank sides individually, this stream was drawn as shown. The left bank (right side of screen) was taken numerically in the downstream direction, but the right bank was not. A false flag arose when the end point of the "single" stream (flag 6) was associated with an Upstream/Start flag. Before the script was changed to evaluate sides individually, this false flag switched the directions of *both* banks rather than just the right, resulting in the right bank in the appropriate direction but the left bank in the wrong direction.



Stream Workflow (Complex example): Drawing a stream in the appropriate direction using flag type--After

- Once the script was modified to look at stream direction by individual bank, the stream was correctly drawn as shown.
- This showcases some of the complex scenarios that have been accounted for.



OBJECTID *	Shape *	Bank_Side	Project_Name	GAI_ID	Stream_Type	Bank_Width_Feet	GPS_Date	Mod_Date	Latitude	Longitude	Impacted_	RC_ID	SWW_ID	Floodway_Buffer	Comment	Stream_Nam	CR_Intersect_	Quality_Use_Desc
1	Polyline	Right	<Null>	SPA-NGP-019	Perennial	20	2/19/2018	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	fishing run	<Null>	<Null>
47	Polyline	Left	<Null>	SPA-NGP-019	Perennial	20	2/19/2018	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	<Null>	fishing run	<Null>	<Null>

Main sections of Stream Script (The comments aren't likely enough to explain every line, this is just for temporary reference).

```
#Loop through and draw streams.#########################################################################################################################################
if ID not in Stream_IDs_REVIEW:
    try:
        #######Draw a stream in order of current Flag_num.
        #Initialize list of streams to attribute
        att_stream = []
        print(str(ID_replace_final) + "uID_" + str(unique_ID))
        unique_ID += 1
        stream_line = arcpy.PointsToLine_management(stream_lyr, workspace + "\stream_line_" + str(ID_replace_final) + "uID_" + str(unique_ID), Line_Field = "Bank_Side", Sort_Field = "Flag_num", Close_Line = "NO_CLOSE")
        att_stream.append("stream_line")
        ####Additionally, account for streams drawn in reverse order.
        #Set coincidence test variables.
        downstream_center, upstream_center, downstream_left, upstream_left, downstream_right, upstream_right = True, True, True, True, True, True
        #Create end and start point features for coincidence test.
        end_point = arcpy.FeatureVerticesToPoints_management(stream_line, workspace + "\end_point_" +str(ID_replace_final) + "uID_" + str(unique_ID), "END")
        start_point = arcpy.FeatureVerticesToPoints_management(stream_line, workspace + "\start_point_" +str(ID_replace_final) + "uID_" + str(unique_ID), "START")
        #If there is one end point (and therefore one start point), we know that only one line was drawn for the ID (likely center bank but possibly only one of two double bank data provided). If one
        #line, run the downstream-end point/upstream-start point coincidence test as usual. If there is more than one end point, this means that there was more than one line drawn, which implies it is
        #right/left bank. Make feature layer of each bank, and run coincidence test seperately for each line. Otherwise, the coincidence test has the potential to be invalid in particular instances.
        end_pt_cnt = int(arcpy.GetCount_management(end_point).getOutput(0))
        if end_pt_cnt == 1:
            print("1 end point")
            #Check downstream-end point coincidence.
            arcpy.SelectLayerByLocation_management(stream_lyr, "ARE_IDENTICAL_TO", end_point)
            with arcpy.da.SearchCursor(stream_lyr, "Flag_type") as cursor:
                for t in cursor:
                    if t[0] == "Upstream/start" or t[0] == "Headcut":
                        downstream_center = False
                        print("downstream center is false")
            #Check upstream-start point coincidence.
            arcpy.SelectLayerByLocation_management(stream_lyr, "ARE_IDENTICAL_TO", start_point)
            with arcpy.da.SearchCursor(stream_lyr, "Flag_type") as cursor:
                for t in cursor:
                    if t[0] == "Downstream/End" or t[0] == "Confluence":
                        upstream_center = False
                        print("upstream center is false.")
        else:
            #Make sub feature layers of just right and just left STREAM points from stream_lyr.
            print("more than one end point")
            stream_lyr_left = arcpy.MakeFeatureLayer_management(stream_lyr, "stream_lyr_left", where_clause="Bank_Side = 'Left'")
            stream_lyr_right = arcpy.MakeFeatureLayer_management(stream_lyr, "stream_lyr_right", where_clause="Bank_Side = 'Right'")
            ##Left Bank coincidence test.
            #Downstream coincidence.
            arcpy.SelectLayerByLocation_management(stream_lyr_left, "ARE_IDENTICAL_TO", end_point)
            with arcpy.da.SearchCursor(stream_lyr_left, "Flag_type") as cursor:
                for t in cursor:
                    if t[0] == "Upstream/Start" or t[0] == "Headcut":
                        downstream_left = False
                        print("downstream left is false")

            #Upstream coincidence
            arcpy.SelectLayerByLocation_management(stream_lyr_left, "ARE_IDENTICAL_TO", start_point)
            with arcpy.da.SearchCursor(stream_lyr_left, "Flag_type") as cursor:
                for t in cursor:
                    if t[0] == "Downstream/End" or t[0] == "Confluence":
                        upstream_left = False
                        print("upstream left is false.")
            ##Right bank coincidence test
            #Downstream coincidence.
            arcpy.SelectLayerByLocation_management(stream_lyr_right, "ARE_IDENTICAL_TO", end_point)
```

```
#If line is single bank and passes coincidence test, attribute and append into data base. If line is single bank but does not pass coincidence test, draw in reverse order, then attribute and append into data base.
if end_pt_cnt == 1:
    if downstream_center == True and upstream_center == True:
        #Add attribute fields to stream_line from STREAM points
        for att in stream_att_list:
            arcpy.AddField_management(*(stream_line,)+att)
        #Update values of stream_line with values from stream_lyr points.
        #Start editing
        edit = arcpy.da.Editor(workspace)
        edit.startEditing(False, True)
        edit.startOperation()
        update_fields = ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]
        with arcpy.da.SearchCursor(stream_lyr, ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]) as cursor1:
            for row1 in cursor1:
                GAI_ID, Type, Bank_Width, Stream_Name, GPS_Date = row1[0], row1[1], row1[2], row1[3], row1[4]
                with arcpy.da.UpdateCursor(stream_line, ["GAI_ID", "Stream_Type", "Bank_Width_Feet", "Stream_Nam", "GPS_Date"]) as cursor2:
                    for row2 in cursor2:
                        for i in range(0,len(update_fields)):
                            row2[i] = row1[i]
                            cursor2.updateRow(row2)
        edit.stopOperation()
        edit.stopEditing(True)
        #arcpy.Append_management(stream_line, ENV_GDB + "\Streams", "NO_TEST")

        #Garbage Collect
        #arcpy.Delete_management(end_point)
        #arcpy.Delete_management(start_point)

    #If line is single bank but fails coincidence test, draw in reverse order.
    elif downstream_center == False or upstream_center == False:
        #Stream has been drawn from downstream to upstream. Switch order of flag num by multiplying by -1. Draw line in this order which will draw stream in appropriate direction.
        arcpy.SelectLayerByAttribute_management(stream_lyr, "CLEAR_SELECTION")
        arcpy.Delete_management(stream_line)
        arcpy.AddField_management(stream_lyr, field_name="Flag_neg", field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED")
        arcpy.CalculateField_management(stream_lyr, field="Flag_neg", expression="!Flag_Num!*-1", expression_type="PYTHON_9.3")
        stream_line_inverse = arcpy.PointsToLine_management(stream_lyr, workspace + "\stream_line_" + str(ID_replace_final) + "uID_" + str(unique_ID), Line_Field = "Bank_Side", Sort_Field = "Flag_neg", Close_Line = "NO_CLOSE
        if ID == 'SPA-NGP-007':
            print("Inverse reached")
        for att in stream_att_list:
            arcpy.AddField_management(*(stream_line_inverse,)+att)
        #Update values of stream_line with values from stream_lyr points.
        #Start editing
        edit = arcpy.da.Editor(workspace)
        edit.startEditing(False, True)
        edit.startOperation()
        update_fields = ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]
        with arcpy.da.SearchCursor(stream_lyr, ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]) as cursor1:
            for row1 in cursor1:
                GAI_ID, Type, Bank_Width, Stream_Name, GPS_Date = row1[0], row1[1], row1[2], row1[3], row1[4]
                with arcpy.da.UpdateCursor(stream_line_inverse, ["GAI_ID", "Stream_Type", "Bank_Width_Feet", "Stream_Nam", "GPS_Date"]) as cursor2:
                    for row2 in cursor2:
                        for i in range(0,len(update_fields)):
                            row2[i] = row1[i]
                            cursor2.updateRow(row2)
        edit.stopOperation()
        edit.stopEditing(True)
        #arcpy.Append_management(stream_line, ENV_GDB + "\Streams", "NO_TEST")

        #Garbage Collect
        #arcpy.Delete_management(end_point)
```

```
#If line is double bank and passes coincidence test, attribute and append into data base. If line is double bank but does not pass coincidence test, determine which side(s) are incorrect, and draw correct and incorrect sides
#accordingly.
elif end_pt_cnt != 1:
    if downstream_left == True and upstream_left == True and downstream_right == True and upstream_right == True:
        #Both banks have been drawn properly. Draw and attribute as is.
        #Add attribute fields to stream_line from STREAM points
        for att in stream_att_list:
            arcpy.AddField_management(*(stream_line,)+att)
        #Update values of stream_line with values from stream_lyr points.
        #Start editing.
        edit = arcpy.da.Editor(workspace)
        edit.startEditing(False, True)
        edit.startOperation()
        update_fields = ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]
        with arcpy.da.SearchCursor(stream_lyr, ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]) as cursor1:
            for row1 in cursor1:
                GAI_ID, Type, Bank_Width, Stream_Name, GPS_Date = row1[0], row1[1], row1[2], row1[3], row1[4]
                with arcpy.da.UpdateCursor(stream_line, ["GAI_ID", "Stream_Type", "Bank_Width_Feet", "Stream_Nam", "GPS_Date"]) as cursor2:
                    for row2 in cursor2:
                        for i in range(0,len(update_fields)):
                            row2[i] = row1[i]
                            cursor2.updateRow(row2)
        edit.stopOperation()
        edit.stopEditing(True)
        #arcpy.Append_management(stream_line, ENV_GDB + "\Streams", "NO_TEST")

        #Garbage Collect
        #arcpy.Delete_management(end_point)
        #arcpy.Delete_management(start_point)
    elif downstream_left == True and upstream_left == True:
        if downstream_right == False or upstream_right == False:
            print("left true right false")
            #Right bank has been drawn from downstream to upstream. Select just the right bank from stream_lyr to delete it as the left bank is drawn correctly and should be left as is.
            #Switch order of flag num by multiplying by -1. Draw line in this order which will draw stream in appropriate direction.
            arcpy.SelectLayerByAttribute_management(stream_lyr, "CLEAR_SELECTION")
            arcpy.MakeFeatureLayer_management(stream_line, "stream_line_right", where_clause="Bank_Side = 'Right'")
            #Note: DeleteFeatures_management allows user to delete the selected feature, while Delete_management does not.
            arcpy.DeleteFeatures_management("stream_line_right")
            arcpy.AddField_management(stream_lyr_right, field_name="Flag_neg", field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED")
            arcpy.CalculateField_management(stream_lyr_right, field="Flag_neg", expression="!Flag_Num!*-1", expression_type="PYTHON_9.3")
            stream_line_right_inverse = arcpy.PointsToLine_management(stream_lyr_right, workspace + "\stream_line_right" + str(ID_replace_final) + "uID_" + str(unique_ID), Line_Field = "Bank_Side", Sort_Field = "Flag_neg", C
            #Add attribute fields to stream_line and stream_line_right_inverse from STREAM points.
            for att in stream_att_list:
                arcpy.AddField_management(*(stream_line,)+att)
            for att in stream_att_list:
                arcpy.AddField_management(*(stream_line_right_inverse,)+att)
            #Update values of stream_line and stream_line_right_inverse with values from stream_lyr points.
            #Start editing.
            for stream in att_stream:
                edit = arcpy.da.Editor(workspace)
                edit.startEditing(False, True)
                edit.startOperation()
                update_fields = ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]
                with arcpy.da.SearchCursor(stream_lyr, ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]) as cursor1:
                    for row1 in cursor1:
                        GAI_ID, Type, Bank_Width, Stream_Name, GPS_Date = row1[0], row1[1], row1[2], row1[3], row1[4]
                        with arcpy.da.UpdateCursor(stream_line, ["GAI_ID", "Stream_Type", "Bank_Width_Feet", "Stream_Nam", "GPS_Date"]) as cursor2:
                            for row2 in cursor2:
                                for i in range(0,len(update_fields)):
                                    row2[i] = row1[i]
```

Attribute Transfer

- Initialize a list of tuples of fields to be added to the derived feature.

```
#Initialize attribute field lists for features.
stream_att_list = [('Project_Name', 'TEXT','','', '50','','',''),('GAI_ID', 'TEXT','','', '16','','',''), ('Stream_Type', 'TEXT','','', '16','','',''),
                   ('Bank_Width_Feet', 'DOUBLE','','','','','',''),('GPS_Date', 'DATE','','','','','',''), ('Mod_Date', 'DATE','','','','','',''),('Latitude', 'DOUBLE','','','','','',''),
                   ('Longitude', 'DOUBLE','','','','','',''), ('Impacted?', 'TEXT','','', '50','','',''), ('RC_ID', 'TEXT','','', '50','','',''), ('SWW_ID', 'TEXT','','', '8','','',''),
                   ('Floodway_Buffer', 'DOUBLE','','','','','',''), ('Comment', 'TEXT','','', '50','','',''), ('Stream_Nam', 'TEXT','','', '50','','',''),
                   ('CR_Intersect?', 'TEXT','','', '8','','',''), ('Quality_Use_Description', 'TEXT','','', '24','','','')]
```

- The following code iterates through the above list and adds each field and its associated parameters to the derived feature.

```
#Add attribute fields to stream_line from STREAM points
for att in stream_att_list:
    arcpy.AddField_management(*(stream_line,)+att)
```

Attribute Transfer

- Align the values from the “from” layer (feature layer of gps data being assessed) and the “to” feature (derived feature) by the following:
  - Create a Search Cursor on the feature layer being assessed to queue the values of the fields requiring transfer to the derived feature
    - Create a list of fields to have their attributes transferred. The length of this list will be used to index the transfer loop
  - Create an Update Cursor on the derived feature to queue the **empty** fields receiving attributes from the corresponding fields queued in the “from” feature layer.
  - Iterate through “from” layer and “to” feature(row1 and row2, respectively) and set row2[currently empty first field to be populated] = row1[value of first field to be sent to the empty row2 field]

Attribute Transfer

```
#Update values of stream_line with values from stream_lyr points.
#Start editing
edit = arcpy.da.Editor(workspace)
edit.startEditing(False, True)
edit.startOperation()
update_fields = ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]
with arcpy.da.SearchCursor(stream_lyr, ["GAI_ID", "Type", "Bank_Width", "Stream_Nam", "GPS_Date"]) as cursor1:
    for row1 in cursor1:
        GAI_ID, Type, Bank_Width, Stream_Name, GPS_Date = row1[0], row1[1], row1[2], row1[3], row1[4]
        with arcpy.da.UpdateCursor(stream_line_inverse, ["GAI_ID", "Stream_Type", "Bank_Width_Feet", "Stream_Nam", "GPS_Date"]) as cursor2:
            for row2 in cursor2:
                for i in range(0,len(update_fields)):
                    row2[i] = row1[i]
                    cursor2.updateRow(row2)
edit.stopOperation()
edit.stopEditing(True)
```

- In the first iteration:
  - Row2[0] is the empty field of the “GAI_ID” of the derived feature
  - Row1[0] is the populated field of the “GAI_ID” of the feature layer being assessed (gps feature flag)
  - “Row2[0] = row1[0]” sets the “GAI_ID” field of the derived feature, and the .updateRow method applies the transfer
- The second iteration would be:
  - Row2[1] is the empty field of the “Stream_Type” of the derived feature
  - Row1[1] is the populated field of the “Type” of the feature layer being assessed

Results on 3 different sets of data

	A	B	C	D	E	F	G	H	I
1	**Project**	**Total number of wetlands (unique GAI_ID)**	**Wetlands drawn**	**% Drawn**	**% drawn correctly (not including parcing by type)**	**% attributed correctly**	**# Null type (>1 type known)**	**Total time (minutes:seconds)**	**Wetlands/minute**
2	Homer City-Indiana	24	19	80	100	100	0	1:50	13
3	New Castle-State Line	79	69	85	100	100	2	5:47	13
4	Wylie Ridge-Cecil	79	63	80	100	100	6	5:44	13
5		61	50	82	100	100			13
6									
7									
8	**Project**	**Total number of streams (unique GAI_ID)**	**Streams drawn**	**% Drawn**	**% drawn correctly (assuming flag type correct)**	**% attributed correctly**		**Total time (minutes:seconds)**	**Streams/minute**
9	Homer City-Indiana	36	44	100	95	98	n/a	4:35	10
10	New Castle-State Line	66	60	90	93	100	n/a	7:51	13
11	Wylie Ridge-Cecil	141	122	90	98	98	n/a	17:40	7
12		81	75	93	95	99			10
13									

- **Wetlands**
  - With the data as is, on average, the script is able to draw 82% of the wetlands given with 100% accuracy with the exception of parsing them by type. As state previously, the wetlands that evaluate fine otherwise are still drawn and attributed but the Type field is left blank (this is printed to the console).
  - With the exception of multi-type wetlands, 100% of the wetlands were attributed correctly.
  - It averages 13 wetlands per minute, which is significantly faster than manually drawing and attributing them, and may still be more efficient given the extra QC that comes with automating processes.
- **Streams**
  - With the data as is, on average, the script is able to draw 93% of the streams given with 95% of them being drawn correctly (assuming flag type is correct) and 99% of them being attributed correctly.
  - The streams are a little more complex than the wetlands (the script is longer), but the average speed is still 10 streams per minute which is still significantly faster than manually drawing and attributing them.

Pros VS Cons

- Pros:
  - The most obvious advantage is how quickly features can be drawn and attributed compared to doing them manually. This in itself may outweigh a lot of the disadvantages.
  - Reducing human error: In the past, I have found myself drawing a lot of data on, for example, 8/04/2018. After doing that for a while and moving on to data from 8/05, I was still entering 8/04 as I had been doing that for many features for a while, and I had to go back and find the mistakes. This method eliminates this possibility.
  - The benefits scale greatly as the amount of data increases. This would allow GAI to have multiple teams on multiple projects on the same day in the field collecting data without requiring an increase of corresponding magnitude in hours spent by GIS staff integrating the data.

- Cons:
  - Instances such as leaving the “Upstream/Start” attribute on for all stream points etc. Even with the dedicated fields that the logic of the script is based upon, this poses a concern (especially instances not caught by the “try-except” block).
  - Not typically as practical for surveys that extend preexisting features (AR study or modified study area).
    - Often times, a wetland extension will only consist of 2 points, or a stream extension will only consist of 1 point. Neither of these will be able to be drawn by the script as they are being treated as different features.
    - Even if the extension can be drawn, it will have to be manually merged into the preexisting feature, negating some of the benefits implicit of creating a new feature by automation.
  - Ideally, field data is to be drawn and verified on a daily (or once every few days) basis as the features will be more freshly recalled by the field team. Given the infrequency of working on 5, 10, or even 20 (not sure if 20 has ever even happened before) gps files at a time, the *substantial* benefit of automated vs manual on a large amount of data may never be realized. Even so, on a daily basis, there still may be room for this method.
  - Miscellaneous things such as a single stream having flags with different widths (only one width could be transferred to the derived features).

Final Thoughts

- Currently, the data is a little messy. Things like having a “Flag Type” populated for streams (an integral part of the script’s process) is in a way a luxury, and is not always added in the field. It is also my understanding that, given the nature of the gps units we currently use, it is very easy to unintentionally leave parameters on for multiple points, and it is a hassle to go back and fix them. For now, GIS staff can often easily discern if there was a mistake and what was originally intended.

- Despite this, the script still runs with impressive accuracy and speed. My hope is that with the new tablets that will be introduced to field staff, simple things like adding a flag type on the ends of streams or being able to see that the culvert point is still turned on is easier to enter, notice, and modify before sending to GIS.

- I am not aware of what possibilities the tablets will bring, and I am not aware even if anything in this presentation will still be applicable once the tablets are fully integrated into our workflow. But, given the numbers, I feel that this should be explored more thoroughly.

I do have a few more concepts I have thought of regarding catching more flaws in the script, but the main flaws have been worked out, and I would like to leave this project as is until it can be determined if it is worth continuing. The main question that this all comes down to is “Is the speed and accuracy of digitizing and attributing field data automatically **with** an additional QC *actually* more efficient across the board than digitizing and attributing manually as this acts, in a sense, as a QC in itself?”