Exercise 6: Messaging and Error Handling

During the execution of ArcGIS geoprocessing tools and function, various messages are returned. These messages can be informational in nature or indicate warning or error conditions that can result in the tool not creating the expected output or in outright failure of the tool to execute. These messages do not appear as message boxes. Instead, you will need to retrieve them using various ArcPy function. To this point in the course we have ignored the existence of these messages, warnings, and errors. This is mainly due to the fact that I wanted you to concentrate on learning some basic concepts without adding the extra layer of code complexity that is necessary for creating robust geoprocessing scripts that can handle error situations gracefully. That being said, it's now time that you learn how to create the geoprocessing and Python exception handling structures that will enable you to create robust geoprocessing scripts that can handle messages that indicate warnings, errors, and general information which are generated while your script is running. These code details will help make your scripts more flexible and less error prone.

In this exercise you are going to learn how to work with these tools in your Python scripts. You're going to use PythonWin for this exercise. At the end of this exercise you will have learned the following:

- How to use Python exception handling structures including try/except
- How to use the ArcPy GetMessages() function to return all messages returned during a geoprocessing tool execution
- Filter messages by severity level
- Examine individual messages

Step 1: Explore the Default Python Error Message In this step we will create and run a script that intentionally contains error conditions. We will not include any geoprocessing or Python exception handling techniques in the script. We're intentionally doing this because I want you to see the error information returned by Python

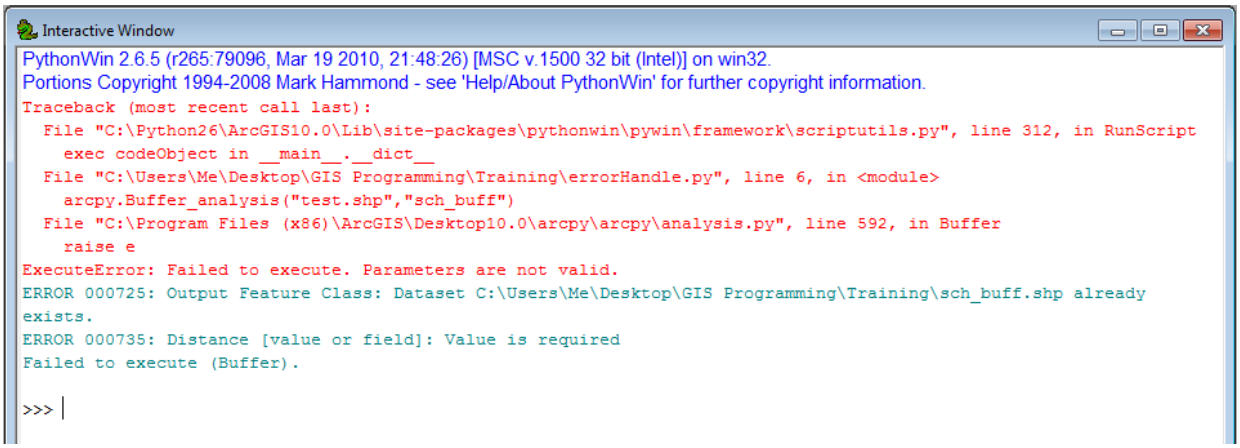
- Open PythonWin and create a new script.
- Insert comments including your name, date, and the purpose of this script.
- Save the script to a location and with a name of your choice.
- Import the ArcPy module
- Set the workspace to Exercise4.mdb
- Call the Buffer tool. The Buffer tool requires a buffer distance be entered as one of its parameters.
  Syntax:
  Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field, {line_side}, {line_end_type}, {dissolve_option}, {dissolve_field})

  In this code block we have forgotten to enter the distance.

errorHandle.py

```
#Name: John Doe
#Date: Aug, 2011
#Purpose: Working with errors in Python
import arcpy
arcpy.env.workspace = "C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\Exercise4.mdb"
arcpy.buffer_analysis("Schools","sch_buff")
```

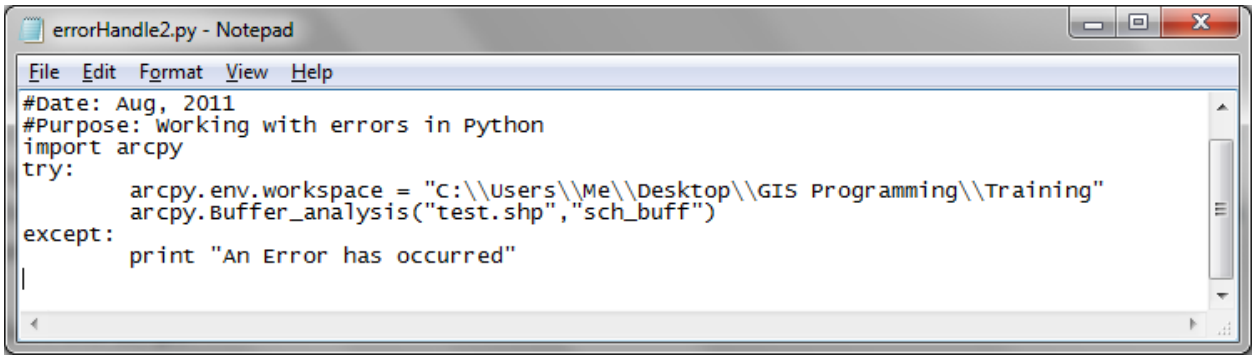
- Run the script. What you'll see is some form of gibberish similar to what you see in the figure below. Now, if you're a fairly experienced programmer you'll generally be able to make out what the problem is in this case (did not include buffer distance). However, in many cases the returned error message is not anywhere near as instructive as in this case. In any event, we can structure our code to more gracefully handle errors such as this.



Step 2: Add some basic Python exception handling structures

In Step 2 we're going to add in some basic Python exception handling structures. From the lecture you will be familiar with the try/except structure. There are several variations of this structure. In this step we'll start with a very simple structure.

- Add the try/except structure to your code. Alter your code as follows:



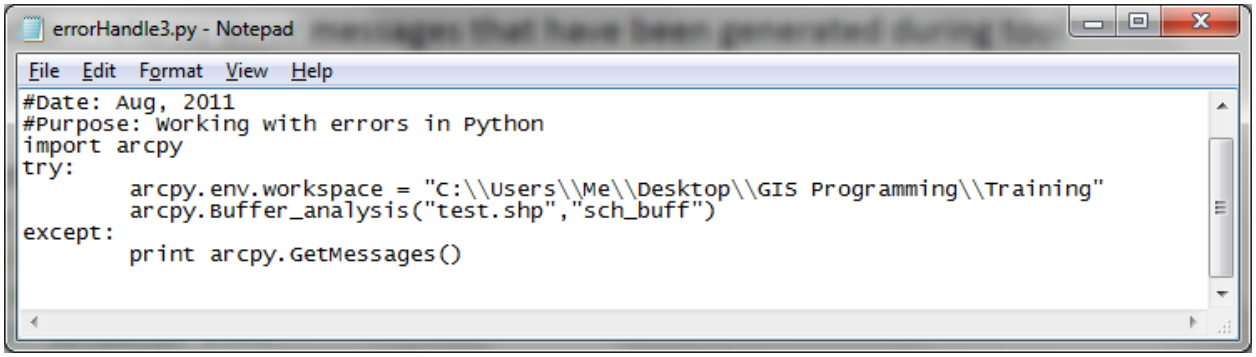
This is an extremely simple structure. The try block indicates that everything indented under the try: statement will be subject to exception handling. If an exception of any type is found, control of the code processing jumps to the except: section and prints out the error message(s) Now, this is hardly informative to your users, but hopefully it gives you a basic idea of how try/except blocks work, and as a programmer you will better understand any errors reported by your users.

- Save the script
- Run the script
- Now, instead of the gibberish you saw in Step 1, the script will return the message "An Error has occurred"

```
>>> An Error has occured
```

Step 3: Add a GetMessages() function to retrieve any message returned by the geoprocessing tool
Now let's move on to a better way of handling exceptions and errors. The geoprocessor stores messages that are generated while a script is running. These messages can be accessed and returned to the user as more descriptive error information.

- Alter your "except" block as follows:



The GetMessages() function returns error messages that have been generated during tool execution.

- Save and run your script. This time the error message should be much more informative.

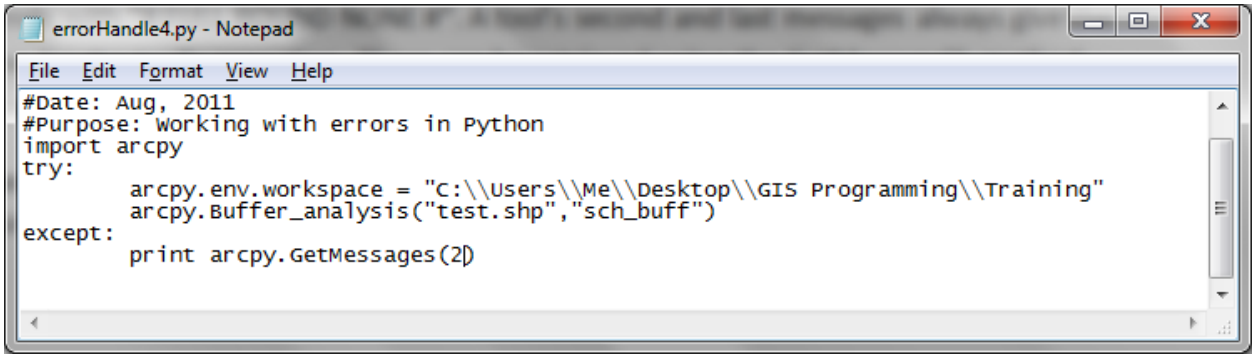
```
Executing: Buffer "C:\Users\Me\Desktop\GIS Programming\Training\test.shp" "C:\Users\Me\Desktop\GIS
Programming\Training\sch_buff.shp" # FULL ROUND NONE #
Start Time: Wed Jul 20 16:32:59 2011
Failed to execute. Parameters are not valid.
ERROR 000725: Output Feature Class: Dataset C:\Users\Me\Desktop\GIS Programming\Training\sch_buff.shp already
exists.
ERROR 000735: Distance [value or field]: Value is required
Failed to execute (Buffer).
Failed at Wed Jul 20 16:32:59 2011 (Elapsed Time: 0.00 seconds)
```

Your script will now return more specific information regarding the nature of the problem. In this case we have simply neglected to enter a buffer distance value or field which is a required parameter. This time you'll see a whole slew of information returned as you can see. Some of it may not make sense to you but let's take some time to explain what you're seeing. The GetMessages() method can take an optional numeric parameter of 0, 1, or 2. A value of 0 indicates all messages, 1 indicates warnings, and a

value of 2 indicates that you'd like to see error messages. Not specifying a value, as we've done in this example, is the same as entering a value of 0 and will return all messages. The first message returned is the actual tool being executed along with any parameters. This is indicated by the line "Executing: Buffer……" and ends with "# FULL ROUND NONE #". A tool's second and last messages always give the start and end time for the tool's execution. These can be retrieved using the GetMessage(i) method which we'll see later.

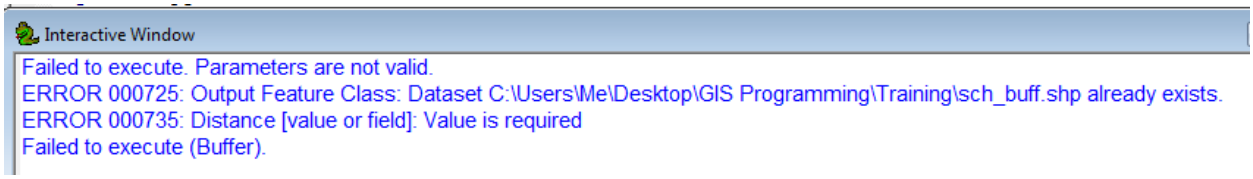
Step 4: Add a parameter to GetMessages so that only error are returned

- Alter your except statement so that it appears as follows:



As I mentioned in the last step the GetMessages() method can accept an integer argument of 0, 1, or 2. Passing a value of 0 indicates that all messages should be returned while passing a value of 1 indicates that you wish to see warnings. In our case, we have passed a value of 2 which indicates that we only want to see error messages.

- Save and run the script. Now you should only see the error message as seen below. This message much more accurately describes the reason that our script failed.



Step 5: Return individual messages with GetMessage()

Up to this point we have been returning all messages generated by the tool. However, you can return individual messages to your user through the GetMessage() method which takes an integer as a parameter indicating the particular message you'd like to retrieve. Each message generated by the tool is placed into a message list or array. We discussed list objects earlier in the course so you'll remember that this is just a collection of some type of object. Just as a reminder, lists are 0 based meaning that the first item in the list is located at position 0. For example, GetMessage(0) would return the first message in the list, while GetMessage(1) would return the second message in the list. The first message is going to be the tool being executed along with any parameters. The second message returns the start time of the tool while the last message returns the end time of the script. Let's see an example of this.

- Alter your except block as follows:

errorHandle5.py - Notepad

File Edit Format View Help

```
#Date: Aug, 2011
#Purpose: Working with errors in Python
import arcpy
try:
        arcpy.env.workspace = "C:\\Users\\Me\\Desktop\\GIS Programming\\Training"
        arcpy.Buffer_analysis("test.shp","sch_buff")
except:
        print arcpy.GetMessage(1)
        print arcpy.GetMessage (arcpy.GetMessageCount()-1)
```

The MessageCount property may be new to you. This property returns the number of messages returned by the tool. Remember that our list of messages is 0 based so we have to subtract one from the MessageCount to arrive at the last message in the list. Otherwise, we'd be attempting to access a message that does not exist.

- Save and run the script.

Interactive Window

```
Start Time: Wed Jul 20 16:44:39 2011
Failed at Wed Jul 20 16:44:39 2011 (Elapsed Time: 0.00 seconds)
```

In this case the start and end times are the same because the script doesn't really do anything. However, it does illustrate how to access the individual messages generated by the tool.