

Script Debugging



This video will present some tips for troubleshooting errors in your script.

Common syntax problems

- Syntax mistakes are easy to make in scripting
 - no spelling or grammar checker
- Common problems include...
 - typos
 - undefined variables
 - incorrect capitalization
 - incorrect / invalid file pathname
 - missing `r` before file names
 - missing parenthesis
 - incorrect indentation



There are several types of syntax problems that are common when first learning to write scripts. Syntax errors and typos can be tedious and frustrating to correct during the debugging process but they can easily avoided by paying a little extra attention to the details when writing the script.

Tips for preventing syntax mistakes

- Copy (Ctrl-c) and paste (Ctrl-v) long variable names.
- Double-check statements after completing...
 - close parenthesis.
 - use proper indentation (more on this next lecture).
 - include placeholders for optional ArcTool statements when necessary.
- Use Python's auto-complete feature.
- Test new statements in Python Shell...
 - find mistakes right away.
- Pay attention!!
 - preventing mistakes saves time and aggravation.



Copying and pasting is an easy way to avoid typos in file names and for long variable names. File workspaces and names can be copied directly from ArcCatalog, ArcMap, or Windows Explorer. Variables and expressions in Python are case-sensitive. When using the python interface, key words will change color when typed correctly. Using the auto-complete feature will help avoid mistakes in spelling method and property names. You can test individual statements in the Python Shell to catch any mistakes immediately.

The error message

- Python's Traceback module automatically provides information on errors...

```
Python Shell
File Edit Shell Debug Options Windows Help

Traceback (most recent call last):
  File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py",
    line 34 in <module>
    arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp")
NameError: name 'soils' is not defined

>>> |
```

When an error occurs while running a script, Python will automatically provide the information needed for tracking down the error. Information relating to errors will appear as red text in the Python Shell.

The error message begins with the file pathname of the script that failed.

Next is the line number that corresponds to the failure...

Followed by the actual statement that failed.

The last part of the message gives a description of the error.

Locating the cause of the error

- Cause of error may or may not be in the failed statement...
 - check failed statement for typos, missing or invalid parameters, etc.
 - then work backward through statements that provide input(s) for the failed statement.
- For general syntax (i.e. unclosed parenthesis) and indentation errors, problem is always somewhere above the line indicated.



The line reported in the error message is the place to start when troubleshooting your script; however, keep in mind that it is not necessarily the source of the problem.

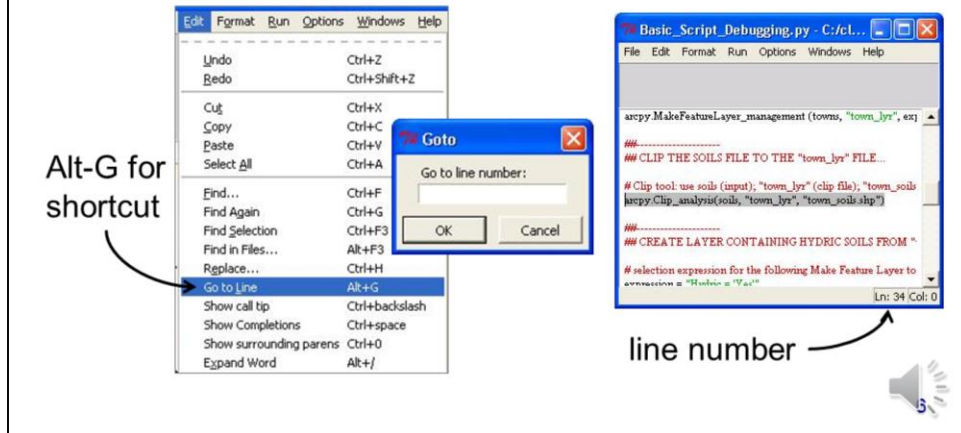
Begin by checking for any apparent problems in the line reported in Python's error message. If you don't find the source of the problem in that line...

Then you will need to work backward in your script through the statements that provided the inputs to the failed line.

Keep in mind that the source of general syntax errors, such as an unclosed parenthesis, is in a statement above the reported line (usually in the preceding statement).

Finding a line by number

- Go to the line number indicated in the error message to find failed statement...



To go to a specific line, type alt-G...

And enter the line number.

Line numbers in the IDLE script window are indicated in the lower right corner of the window.

Identifying the error

```
arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp")
```

- Error message provides a clue for some errors...

NameError: name 'soils' is not defined

- is soils spelled correctly? (remember, variables are case-sensitive)
- has soils been defined in a previous line?
- is soils supposed to be a variable? (i.e. are quotes needed?)
- locate definition of soils...



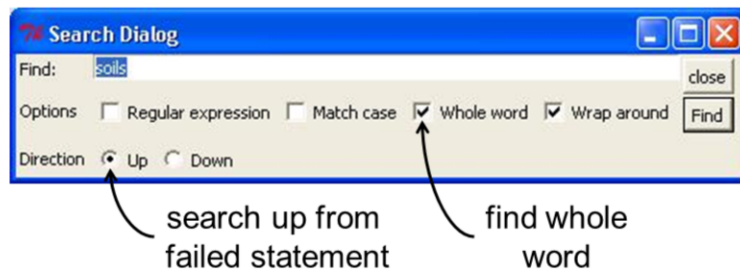
Let's work through an example of the process of troubleshooting an error. Assume that this is the statement on which the script crashed.

First, examine the error description for clues.

In this case, Python doesn't know how to interpret **soils**. If it's supposed to be a variable, it either hasn't been defined previously or it has been misspelled in the current line. If **soils** is supposed to be a string, then it needs to have quotes.

Finding expressions

- To search for a word...
 - type Ctrl-F to open word search window.
 - try searching up, from statement, to find lines where input parameters are defined.



If we're sure that **soils** should be a variable, then we need to find out where and if it is defined in order to figure out why Python doesn't recognize it. To search for a word in your script, type ctrl-F to open the search window. Type in the word you're searching for and make sure the **up** direction is selected – the variable should have been defined on a previous line. If your search doesn't find a variable named soils on a previous line, then you'll have confirmed the problem is an undefined or misspelled variable.

Object has no attribute

Traceback (most recent call last):

File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py" line 55, in <module>
arcpy.AddField_management("wetland_buf.shp", "Area", "DOUBLE")

AttributeError: 'module' object has no attribute 'Addfield_management'

incorrect spelling of method is likely

arcpy.AddField_management("wetland_buf.shp", "Area", "DOUBLE")

← correction

arcpy.AddField_management(
"wetland_buf.shp", "Area", "DOUBLE")




In this example, the error message tells us ...

the failed statement is on line 55...

And the **Addfield_management** method we're trying to use doesn't exist. This error typically indicates a spelling error.

If we look up the ArcGIS help page for this tool, we'd find that the "F" should be uppercase. Using auto-complete helps avoid this type of error.

General syntax error



The screenshot shows a Python script with the following content:

```
# Make Feature Layer tool: use towns for input; "town_lyr" :
arcpy.MakeFea...s, "town_lyr", €

##-----
## CLIP THE ...yr" FILE...

# Clip tool: use soils (input); "town_lyr" (clip file); "town_sc
arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp"

##-----
## CREATE LAYER CONTAINING HYDRIC SOILS FRO

# selection expression for the following Make Feature Layer
expression = "Hydric = 'Yes'"
```

A dialog box titled "Syntax error" is overlaid on the script, displaying the message: "There's an error in your program: invalid syntax".

Annotations include:

- A red arrow pointing to the closing parenthesis of the `Clip_analysis` function call, with the text "parenthesis not closed".
- A blue arrow pointing to the `expression = "Hydric = 'Yes'"` line, with the text "check highlighted statement and line(s) above indicated line".

A speaker icon is visible in the bottom right corner of the screenshot.

A general syntax error will cause

a **Syntax error** window to open when you run your script...

and a line to be highlighted. The syntax problem is typically in the preceding statement.

In this case, the error was due to an open parenthesis.

General syntax error



```
while X <= maxX - cellsize  
for x, y in [[X, Y], [X, Y+cellsize], [X+cellsize,  
pnt = arcpy.Point(x,y)  
array.append(pnt)
```

missing colon

Note: Header lines for compound statements must end in a colon – more on this next week...



Another type of syntax error is associated with compound statements. Compound statements are multi-line statements, such as loops, and start with a header line.

The header line of a compound statements must end with a colon.

The error in this case is due to a missing colon.

General syntax error



```
while X <= maxX - cellsize:
```

```
    for x, y in [[X, Y], [X, Y+cellsize]]
```

inconsistent
indentation

```
        pnt = arcpy.Point(x,y)
        array.append(pnt)
```

Note: code blocks defined by indentation. Always use tabs to ensure consistent indentation – more on this next week...

Another error associated with compound statements has to do with indentation.

Statements under the header line must have consistent indentation...

any extra space in the indentation will cause an error.

Invalid ArcTool parameter (input file)


```
Traceback (most recent call last):
  File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py",
  line 34, in <module>
    arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp")
  File "C:\Program Files\ArcGIS\Desktop10.0\arcpy\arcpy\analysis.py", line 55, in Clip
    raise e
ExecuteError: Failed to execute. Parameters are not valid.
ERROR 000732: Input Features: Dataset C:\Python_workshop\middlesexsoils.shp does not exist or is not supported
Failed to execute (Clip).
```

Find line associated with your script

Failure in line 34; Clip_analysis tool

Error type invalid parameter

ArcTool error input file does not exist



When running ArcTools or Python modules, the error message may list multiple scripts.

In this case, you'll need to find the information associated with your script – it is most likely an error in your script that caused the ArcTool or Python tool to crash.

The error description indicates an invalid value was used for a parameter.

The error description further states that a particular input file does not exist.

Invalid ArcTool parameter (input file)

- Check failed statement

```
arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp")
```

- Check if input file does not exist...

```
arcpy.Exists(soils) ← output is True or False
```

- If soils file does not exist, check definition...

```
soils = r"C:\NRE_5585\middlesexsoils.shp"
```

fix workspace

```
soils = r"C:\NRE_5585\Data\middlesexsoils.shp"
```



Once we locate the failed statement in our script,

Then we'll want to confirm that the input file, indicated in the error message, does not exist. We can test this in the Python Shell using the `arcpy.Exists` tool.

If the test returns a `False` value, then we'll need to check the workspace and name associated with the input file.

In this case, an incorrect workspace was the source of the problem.

Invalid ArcTool parameter

```
Traceback (most recent call last):
  File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py", line 49, in <module>
    arcpy.Buffer_analysis("town_wetland_lyr", "wetland_buffer.shp", "100 FEET", "", "ALL")
  File "C:\Program Files\ArcGIS\Desktop10.0\arcpy\arcpy\analysis.py", line 592, in Buffer_analysis
    raise e
ExecuteError: Failed to execute. Parameters are not valid.
WARNING 000725: Output Feature Class: Dataset C:\Python_workshop\results\wetland_buffer.shp already exists.
ERROR 000800: The value is not a member of ROUND | FLAT.
Failed to execute (Buffer).
```

Failure in line 49;
Buffer_analysis

ArcGIS module
– not relevant

Just a warning,
not an error

ArcTool error
invalid value

In another example,

We need to find the error associated with our script...

And disregard errors associated with arcpy or other modules.

Note that we have a warning message – this is not necessarily a problem. In this case the warning is not associated with the error.

The error description indicates that there is an invalid parameter value.

Invalid ArcTool parameter

- Check parameters in failed statement. No indication of input/output file error, start with other parameters...

```
arcpy.Buffer_analysis("town_wetland_lyr",  
    "wetland_buf.shp", "100 FEET", "", "ALL")
```

dissolve type

Syntax

```
Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field,  
{line_side}, {line_end_type}, {dissolve_option}, {dissolve_field})
```

- missing line_end_type parameter...

```
arcpy.Buffer_analysis("town_wetland_lyr",  
    "wetland_buf.shp", "100 FEET", "", "ALL")
```

add in placeholder



In this case, the error message did not indicate any problems with the input/output files so we can focus on checking the remaining parameters in the statement.

Note that "ALL" was intended to be the value for the tool's **dissolve type** parameter.

Open the tool's help page and check through each parameter making sure that the specified parameters have appropriate values and that no required parameters have been skipped.

In this case, checking the parameters reveals that the **line_end_type** parameter is missing a placeholder.

Although **line_end_type** is an optional parameter, and we chose not to specify a value, we still need to include the empty quotes as a placeholder so that the value for the **dissolve_type** parameter will be in the correct position.

File lock error

```
Traceback (most recent call last):
  File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py", line 34, in <module>
    arcpy.Clip_analysis(soils, "town_lyr", "town_soils.shp")
  File "C:\Program Files\ArcGIS\Desktop10.0\arcpy\arcpy\analysis.py", line 55, in Clip
    raise e
ExecuteError: ERROR 000464: Cannot get exclusive schema lock. Either being edited or in use by another application.
Failed to execute (Clip).
```

- File is locked and in use by another application
 - file locked to editing if open in ArcMap
 - sometimes locked up by previous run of the script
- Close ArcMap
- Restart Python



The error in this example is due to a file lock which usually results from the file being open in another application such as...

ArcMap...

Or your script.

The solution is to remove the corresponding layer from ArcMap. Sometimes closing ArcMap or ArcCatalog is also necessary.

If closing ArcMap failed to release the lock, then you may need to restart Python to release any file locks.

Invalid SQL statement

```
Traceback (most recent call last):
  File "C:/classes/Python_course_fall_2011/Exercises/Lecture/Week_3/Basic_Script_Debugging.py", line 49, in <module>
    arcpy.Buffer_analysis("town_wetland_lyr", "wetland_buffer", "100 FEET", "", "", "ALL")
  File "C:\Program Files\ArcGIS\Desktop10.0\arcpy\arcpy\analysis.py", line 592, in Buffer
    raise e
ExecuteError: ERROR 999999: Error executing function.
An invalid SQL statement was used.
An invalid SQL statement was used.
An invalid SQL statement was used.
An invalid SQL statement was used.
An invalid SQL statement was used.
An invalid SQL statement was used.
Failed to execute (Buffer).
```

- Failure in line 49
 - Buffering tool
- Invalid SQL statement
- No SQL statement used by Buffer tool
 - error not in Buffer tool statement
 - need to trace error back to origin...



In this case, we have a failure associated with a statement in...

Line 49 which uses arcpy's buffering tool.

The error description indicates that an "Invalid SQL statement was used". This error should not be expected because...

The buffering tool uses no SQL statements.

Thus the actual error is not in the statement that failed

And we need to trace the error back to the source.

Invalid SQL statement

1. Check failed statement for syntax errors, missing parameters, etc...

```
arcpy.Buffer_analysis("town_wetland_lyr",  
    "wetland_buf.shp", "100 FEET", "", "", "ALL")
```

2. Check input and output parameter files...

- Find "town_wetland_lyr" origin using word search tool...

```
arcpy.MakeFeatureLayer_management("town_soils.shp",  
    "town_wetland_lyr", expression)
```

SQL expression 



The first step, as always, is to check the failed statement – even if you suspect the error is in a previous statement.

While checking the statement, using the tool's help page as a guide, you'll notice that there is no parameter for which there could be an invalid SQL statement.

The next step is to check the files specified for the input and output files.

The first input file in this case is a **layer file** – note that layer files need no file extension. Layers can be thought of as a temporary copy of an actual dataset. One purpose of layer files is to allow SQL queries to be used to select items from a dataset.

In this case, the "town_wetland_lyr" was created using arcpy's **MakeFeatureLayer** tool.

This tool contains an SQL expression as a parameter – this is a likely source of the error that we're trying to troubleshoot.

Invalid SQL statement

3. Find origin of expression using word search...

```
expression = "Hydric = 'Yes"
```

missing single quote

4. Correct SQL statement...

```
expression = "Hydric = 'Yes'"
```

- Source of error may be far removed from location of failure.
- Frequent testing, while writing the script, will make syntax errors much easier to catch.



When we trace the **expression** variable to its definition...

We find that there is a missing single quote...

In an SQL statement, there need to be single quotes around strings within the statement.

The error in this example shows that the actual problem may be far removed from the statement that crashed the script. Frequent testing of your script, as you develop it, can allow you to catch errors while they are still easy to find.

Getting a feature count

- Often useful to get a count of the number of features in a dataset...
 - i.e. for the output of selections by Make Feature Layer, Select, or Select Layer by Attribute/Location
 - some ArcTools cannot be run on empty datasets.
 - script can skip operation(s) if no features.
- To get a feature count...

```
int(arcpy.GetCount_management(dataset).getOutput(0))
```



It is often useful to have a script get a count of the number of features in layer or dataset ...

This is helpful for checking that selection expressions were correctly applied when creating layer files. ...

It can also allow you to prevent the script from running a process on empty datasets which could cause a crash

The **GetCount** tool allows you to get a count of the number of features in a dataset or layer. The output of the tool is a **result** object and the **getOutput** method is needed to extract the value. The value is extracted as a string so you will want to convert the string to an integer number using the **int** function.