Chapter 7
Getting User Input

Abstract Scripts that are designed to be flexible can be reused with varying parameter values. Relying on hard-coded parameters, specified with string literals, numbers and so forth, means users can't change the values without opening the script and modifying the code. A flexible script that accepts arguments for the parameters can be easier to reuse. The geoprocessing examples in Chapter 6 specified the tool parameters inside the script. These processes can be made dynamic by accepting user input. This chapter covers the `arcpy` approach to receiving arguments and contrasts this with using `sys.argv`. Then it discusses argument spacing and `os` module methods for handling file path arguments and getting the script path.

Chapter Objectives
After reading this chapter, you'll be able to do the following:

- Capture user input with two different techniques.
- Predict script behavior in case of too few arguments.
- Group characters in arguments appropriately.
- Explain why single quotations cannot be used to group arguments.
- Extract the file name, directory, and file extension from a full path file name.
- Join base file names with paths.
- Get the size of a file.
- Get the path of the current script.

7.1 Hard-coding versus Soft-coding

Hard-coded tool parameters are encoded specifically within the script, so that the code needs to be altered in order to change this values. More flexible scripts get input from the user—they are *soft-coded*. The input and output features ("park.shp" and "boundingBoxes.shp") in Example 7.1 are hard-coded.

L. Tateosian, *Python For ArcGIS*, DOI 10.1007/978-3-319-18398-5_7

Example 7.1

```
# boundingGeom.py
# Purpose: Find the minimum bounding geometry of a set of features.

import arcpy

arcpy.env.overwriteOutput = True
arcpy.env.workspace = 'C:/gispy/data/ch07'

inputFeatures = 'park.shp'
outputFeatures = 'boundingBoxes.shp'

arcpy.MinimumBoundingGeometry_management(inputFeatures,
                                         outputFeatures)
```

In this chapter, we compare two techniques for accessing user arguments:

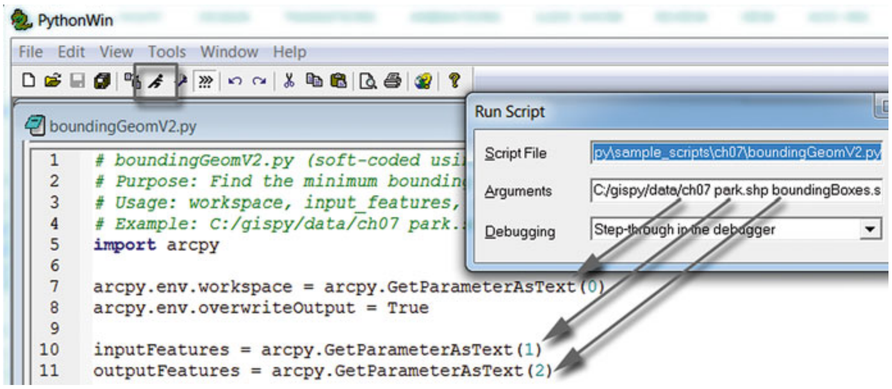
1. An `arcpy` function named `GetParameterAsText`.
2. The `sys` Python standard module variable named `sys.argv`.

7.2 Using GetParameterAsText

The `arcpy` package has a `GetParameterAsText` function that gets input from the user. To use this `arcpy` function:

1. Use `arcpy.GetParameterAsText(index)`, starting with `index = 0`, to replace the hard-coded parameters.
2. Add usage and example comments in the header to tell the user how to run the script.

Example 7.2 shows the script modified with these changes. Indices start at zero and correspond to the order in which the arguments are passed. To run the script



```
# boundingGeomV2.py (soft-coded usi
# Purpose: Find the minimum boundin
# Usage: workspace, input_features,
# Example: C:/gispy/data/ch07 park.
import arcpy

arcpy.env.workspace = arcpy.GetParameterAsText(0)
arcpy.env.overwriteOutput = True

inputFeatures = arcpy.GetParameterAsText(1)
outputFeatures = arcpy.GetParameterAsText(2)
```

Script File: py\sample_scripts\ch07\boundingGeomV2.py
Arguments: C:/gispy/data/ch07 park.shp boundingBoxes.s
Debugging: Step-through in the debugger

Figure 7.1 User arguments are consumed by the `GetParameterAsText` function.

with user input in PythonWin, place arguments, separated by spaces, in the 'Arguments' list in the 'Run Script' dialog. In PyScripter, go to Run>Command Line Parameters… and place the arguments in the text box (space separated). Make sure that 'Use Command Line Parameters?' is checked. Figure 7.1 shows 'boundingGeomV2.py' being run in PythonWin with these arguments: "C:/gispy/data/ch07" "park.shp" "boundingBoxes.shp".

Example 7.2

```
# boundingGeomV2.py (soft-coded using arcpy)
# Purpose: Find the minimum bounding geometry of a set of features.
# Usage: workspace, input_features, output_features
# Example: C:/gispy/data/ch07 park.shp boundingBoxes.shp

import arcpy

arcpy.env.overwriteOutput = True
arcpy.env.workspace = arcpy.GetParameterAsText(0)

inputFeatures = arcpy.GetParameterAsText(1)
outputFeatures = arcpy.GetParameterAsText(2)

arcpy.MinimumBoundingGeometry_management(inputFeatures,
                                         outputFeatures)
```

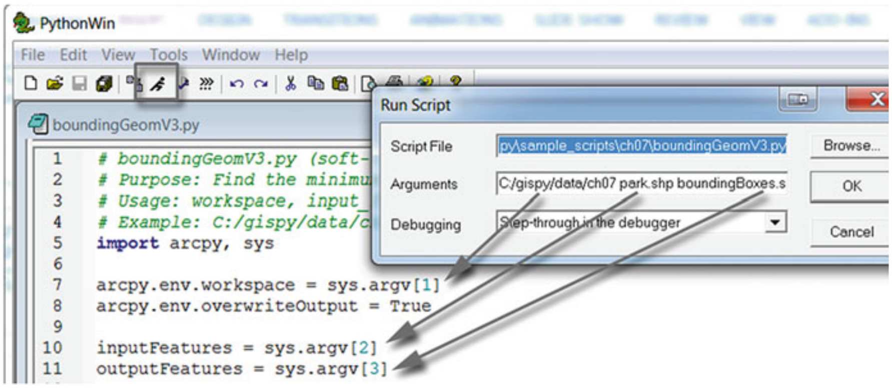
7.3 Using `sys.argv`

If the script imports the `arcpy` package, `GetParameterAsText` is a good approach to use. However, if the script is designed to be used on a machine that might not have ArcGIS installed, the built-in system module should be used instead. This `sys` module has a property named `argv` (short for argument vector). The `sys.argv` variable stores:

1. The name of the script, including the path where it is stored (the full path file name of the script).
2. The arguments passed into the script by the user.

`sys.argv` is a zero-based Python list that holds the script name and the arguments. *The first item in the list is the full path file name of the script itself.* The next item (indexed 1) is the first user argument. To soft-code a script using the `sys` module, you need to do these three things:

1. Import sys.
2. Replace hard-coded parameters with sys.argv[index], starting with index = 1 (since the zero index is used for the script path).
3. Add usage and example comments in the header to tell the user how to run the script.



```
# boundingGeomV3.py (soft-
# Purpose: Find the minimu
# Usage: workspace, input_
# Example: C:/gispy/data/c
import arcpy, sys

arcpy.env.workspace = sys.argv[1]
arcpy.env.overwriteOutput = True

inputFeatures = sys.argv[2]
outputFeatures = sys.argv[3]
```

Script File: py\sample_scripts\ch07\boundingGeomV3.py
Arguments: C:/gispy/data/ch07 park.shp boundingBoxes.s
Debugging: Step-through in the debugger

Figure 7.2 User arguments automatically populate the `sys.argv` list.

To get three arguments passed by the user, 'boundingGeomV3.py' in Example 7.3 uses `sys.argv[1]`, `sys.argv[2]`, and `sys.argv[3]`.

Example 7.3

```
# boundingGeomV3.py (soft-coded using sys)
# Purpose: Find the minimum bounding geometry of a set of features.
# Usage: workspace, input_features, output_features
# Example: C:/gispy/data/ch07 park.shp boundingBoxes.shp

import arcpy, sys

arcpy.env.overwriteOutput = True
arcpy.env.workspace = sys.argv[1]

inputFeatures = sys.argv[2]
outputFeatures = sys.argv[3]

arcpy.MinimumBoundingGeometry_management(inputFeatures,
                                         outputFeatures)
```

Figure 7.2 shows 'boundingGeomV3.py' being run in PythonWin with these arguments: "C:/gispy/data/ch07" "park.shp" "boundingBoxes.shp".

7.4 Missing Arguments

A script can fail if the user does not provide enough arguments. When an argument is missing, the `GetParameterAsText` method itself does not throw an error. Instead, it returns an empty string. If the script in Example 7.2 is run without any arguments, the script assigns an empty string to the input and output features. Then when the tool is called, the following error is thrown, since the tool is being run on empty strings (only the last three lines of the error are shown here):

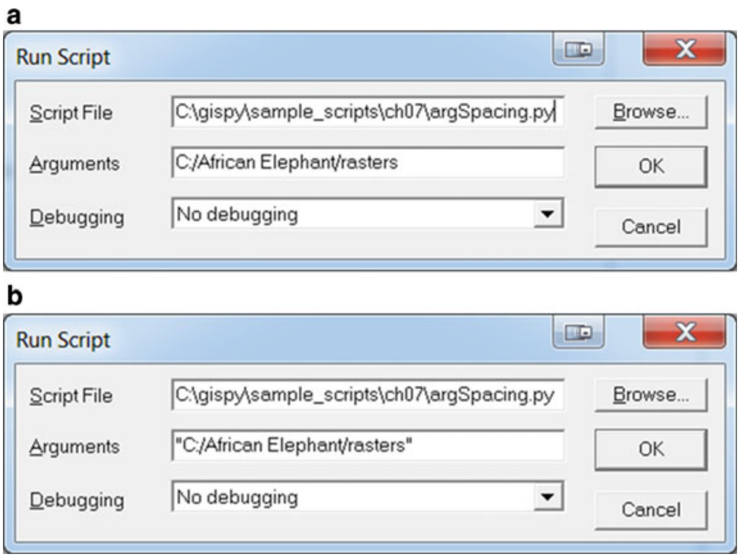
```
ERROR 000735: Input Features: Value is required
ERROR 000735: Output Feature Class: Value is required
Failed to execute (MinimumBoundingGeometry).
```

When an argument is missing in a script that instead uses the `sys.argv` method, the script throws an `IndexError` because `sys.argv` is a Python list and the code is trying to use a list index that is beyond the end of the list. If Example 7.3 is run without arguments, the following error is thrown:

```
arcpy.env.workspace = sys.argv[1]
IndexError: list index out of range
```

7.5 Argument Spacing

Arguments are space delimited in the 'Arguments' list. Consequently, if an argument itself has embedded spaces, it must be surrounded by quotations else it will be interpreted as more than one argument. For example, if the intended argument is a file path such as "C:/African Elephant/rasters", Figure 7.3a will produce an incorrect result, but Figure 7.3b will produce the desired results.



a

Run Script

Script File: C:\gispy\sample_scripts\ch07\argSpacing.py

Arguments: C:/African Elephant/rasters

Debugging: No debugging

b

Run Script

Script File: C:\gispy\sample_scripts\ch07\argSpacing.py

Arguments: "C:/African Elephant/rasters"

Debugging: No debugging

Figure 7.3 The argument is a file path with an embedded space between 'African' and 'Elephants'. Run script (**a**) will interpret it as two arguments ('C:/African' and 'Elephant/rasters'). Run script (**b**) will interpret it as one argument ('C:/African Elephant/rasters').

'argSpacing.py' in Example 7.4 illustrates this problem. Running it with the input shown in Figure 7.3a produces this output:

```
>>> Number of user arguments: 2
The first argument: C:/African
The second argument: Elephant/rasters
```

Running it with the input shown in Figure 7.3b produces this output:

```
>>> Number of user arguments: 1
The first argument: C:/African Elephant/rasters
The second argument:
```

Usually in Python, single and double quotes are interchangeable. Arguments are an exception. The quotation marks around arguments passed in through an IDE must be double, not single. PythonWin and PyScripter do not interpret the single quotations as an indication of grouping, so passing 'C:/My data/rasters' into 'argSpacing.py' returns two arguments:

```
>>> Number of user arguments: 2
The first argument: 'C:/African
The second argument: Elephant/rasters'
```

The single quotation marks appear in the printed output too. Python IDE's do not strip off the single quotes, so even if you do not have spaces within an argument, using single quotes can cause failure. For example, use "park.shp" as a script argument not 'park.shp'. You may be able avoid the file path problem by using file paths that don't contain spaces. But some arguments, such as linear or areal units, must have embedded spaces. For simplicity, you can use quotation marks around arguments only when necessary and then be sure to use double quotation marks instead of single.

Example 7.4

```
# argSpacing.py
# Purpose: Print the number of incoming user
# arguments and the first 2 arguments.

import arcpy

numArgs = arcpy.GetArgumentCount()
print 'Number of user arguments: {0}'.format(numArgs)
print 'The first argument: {0}'.format(arcpy.GetParameterAsText(0))
print 'The second argument: {0}'.format(arcpy.GetParameterAsText(1))
```

Note: Use Double Quotes for IDE Arguments

No quotation marks are needed for arguments that don't have embedded spaces. However, if you do need a space within a script argument, you must use double quotes, not single quotes. For example, use "5 miles", not '5 miles'.

7.6 Handling File Names and Paths with `os` Module Functions

GIS scripts often need to separate or join file names and file paths. The standard `os` (operating system) module introduced briefly in Chapter 2, has a `path` sub-module for file path name manipulation, including the functions `dirname`, `basename`, `join`, and `getsize`. Since `path` is a submodule of `os`, double dot notation (`os.path.functionName`) is used to access these functions. The following code uses the `basename` method to extract the file's base name from the full path file name:

```
>>> import os
>>> inFile = 'C:/gispy/data/ch07/park.shp'
>>> # Get only the file name.
>>> fileName = os.path.basename(inFile)
>>> fileName
'park.shp'
```

The following code uses the `dirname` method to extract the directory path from the full path file name:

```
>>> # Get only the path.
>>> filePath = os.path.dirname(inFile)
>>> filePath
'C:/gispy/data/ch07'
```

The following code uses `join` to create a full path file name from the two parts:

```
>>> # Join the arguments into a valid file path.
>>> fullPath = os.path.join(filePath, fileName)
>>> fullPath
'C:/gispy/data/ch07\\park.shp'
```

This operation could be performed with concatenation too, but the join automatically takes care of slashes. In Example 7.5, 'copyFile.py' uses `os.path.basename` to get the input file name so that it can be copied to a backup directory. This script also uses `os.path.join` to create the new full path name.

Example 7.5

```
# copyFile.py
# Purpose: Copy a file.
# Usage: source_full_path_file_name, destination_directory
# Example: C:/gispy/data/ch07/park.shp C:/gispy/scratch/
# The C:/gispy/scratch directory must already exist
# for this example to work.

import arcpy, os

inputFile = arcpy.GetParameterAsText(0)
outputDir = arcpy.GetParameterAsText(1)

baseName = os.path.basename(inputFile)
outputFile = os.path.join(outputDir, baseName)

arcpy.Copy_management(inputFile, outputFile)

print 'inputFile =', inputFile
print 'outputDir =', outputDir
print
print 'baseName =', baseName
print 'outputFile = ', outputFile
```

Printed output:

```
inputFile = C:/gispy/data/ch07/park.shp
outputDir = C:/gispy/scratch

baseName = park.shp
outputFile = C:/gispy/scratch\park.shp
```

Other `os.path` methods are useful as well. In Example 7.6, 'compact.py' uses the `getsize` method to compare the size in bytes of a Microsoft database file before and after it has been compacted. The output from running this script with C:/gispy/data/ch07/cities.mdb looks something like this:

```
>>> cities.mdb file size before compact: 1552384 bytes.
>>> cities.mdb file size AFTER compact: 397312 bytes.
```

Example 7.6

```
# compact.py
# Purpose: Compact a file
# Usage: Full path file name of an mdb file.
# Example: C:/gispy/data/ch07/cities.mdb

import arcpy, os

# Get user input and extract base file name.
fileName = arcpy.GetParameterAsText(0)
baseName = os.path.basename(fileName)

# Check size
size = os.path.getsize(fileName)
print '{0} file size before compact: {1} bytes.'.format(baseName,
                                                        size)

# Compact the file
arcpy.Compact_management(fileName)

# Check size
size = os.path.getsize(fileName)
print '{0} file size AFTER compact: {1} bytes'.format(baseName,
                                                      size)
```

Chapter 6 demonstrated how to use slicing to separate a file path from its extension when the length of the extension is known. For example, the following code removes a three character extension (and the dot):

```
>>> myShapefile = 'parks.shp'
>>> rootName = myShapefile[:-4]
>>> rootName
'parks'
```

If the extension length is unknown, an `os.path` method can be used to split a file extension from its name. The `os.path.splitext` splits the file name at the dot in the name (if there is more than one dot, it uses the last one). It returns a tuple containing the two parts, the root name and the extension:

```
>>> os.path.splitext(myShapefile)
('parks', '.shp')
```

If the name has no extension, the first item is the name and the second is an empty string:

```
fc = 'farms'
>>> os.path.splitext(fc)
('farms', '')
```

Indexing the first item retrieves the root name:

```
>>> os.path.splitext(myShapefile)[0]
'parks'
```

```
>>> os.path.splitext(fc)[0]
'farms'
```

Slicing, on the other hand, may not work as expected, if the file extension length is unknown:

```
>>> fc[:-4]
'f'
```

7.6.1 Getting the Script Path

A simple way to share a script is to provide the script and the data it needs within a single common folder. The relative path to the data remains the same as long as the recipient leaves the data in the same relative position. In this case, the script can use its own location as the workspace. The command `os.path.abspath(__file__)` can be used inside a script to get the full path file name of the script being run by PythonWin. The `__file__` variable returns the name of the script. Depending on how the script is run, this may or may not be the full path file name. But calling the `abspath` method on the `__file__` constant returns the full path file name. The `dirname` method can then be used to get the script's directory as shown in Example 7.7. This script also demonstrates another useful `os` command method mentioned in a previous chapter. The `listdir` method returns a list of file names for every file in the directory that's passed into it as an argument. The output from running this script in a directory containing 9 files looks like this:

```
['argSpacing.py', 'boundingGeom.py', 'boundingGeomV2.py',
'boundingGeomV3.py', 'buffer_clipv2.py', 'compact.py', 'copyFile.py',
'near.py', 'scriptPath.py']
```

Example 7.7

```
# scriptPath.py
# Purpose: List the files in the current directory.
# Usage: No user arguments needed.
import os

# Get the script location
scriptPath = os.path.abspath(__file__)
scriptDir = os.path.dirname(scriptPath)

print '{0} contains the following files:'.format(scriptDir)
print os.listdir(scriptDir)
```

Avoid NameErrors!

PythonWin sometimes causes us to overlook missing import statements or assignment statements because variables keep their values throughout a PythonWin session. Try the following:

1. Run 'scriptPath.py' and keep PythonWin open.
2. Delete the import statements and save the script.
3. Run it again. It runs successfully. No need to import `sys` and `os`, right?
4. Close PythonWin.
5. Reopen PythonWin and run 'scriptPath.py' a third time. Now it throws a `NameError` exception.

If you run a script that imports a module, other scripts run within the same PythonWin session won't need to import that module to use it. A `NameError` won't occur unless you close PythonWin, then reopen it and run the script which is missing the import.

Always perform a final test within a fresh PythonWin session before sharing code or, alternatively, run scripts with PyScripter, which checks for imports with each script run.

7.7 Key Terms

Hard-coding vs. Soft-coding
`arcpy.GetParameterAsText(index)`
`sys.argv`
`os.path.dirname(fileName)`
`os.path.basename(fileName)`
`os.path.splitext(fileName)`
`os.path.abspath(path)`
`__file__`
File base name
Full path file name

7.8 Exercises

1. The code in 'near.py' uses the Near (Analysis) tool, which determines the distance from each feature in one dataset to the nearest feature in another dataset, within a given search radius. (e.g., a hiker can find historic landmarks close to the trails). The script default values point to trail and landmarks shapefiles. Inspect and run 'near.py' as described below to answer the following questions:

   (a) Run with arguments: # # #
   Which two shapefiles did the script use for the Near (Analysis) tool? How many nearby features were found?

(b) Run with arguments: # #
    Why didn't the script run successfully?
(c) Run with arguments: # # "100 kilometers"
    How many nearby features were found? Why does this give a higher value than part a?
(d) Run with arguments: # # 100 kilometers
    How many nearby features were found? Why does this give a lower value than part c?
(e) Run with arguments:
    'C:/gispy/data/ch07/data/landmarks.shp' 'C:/gispy/data/ch07/data/trails.shp' '50 miles'
    Why does this throw an error?
(f) Run with arguments:
    C:/gispy/data/ch07/My data/landmarks.shp C:/gispy/data/ch07/My data/trails.shp "50 miles"
    Why does this throw an error?
(g) Run with arguments:
    "C:/gispy/data/ch07/My data/landmarks.shp" "C:/gispy/data/ch07/My data/trails.shp" "50 miles"
    Why do you need all of these quotation marks?

2. Sample script 'buffer_clipv2.py' takes an input workspace, an output directory, a fire damage shapefile, a buffer distance, and an input polygon shapefile as arguments (It is just like sample script 'buffer_clipv2.py' except for the soft-coding). Which of the following arguments for sample script 'buffer_clipv2.py' is invalid input and why?

   C:/gispy/data/ch07/ C:/gispy/scratch/ special_regions.shp "1 mile" park.shp
   "C:/gispy/data/ch07/" "C:/gispy/scratch/" "special_regions.shp" "1 mile" "park.shp"
   C:/gispy/data/ch07/ C:/gispy/scratch/ special_regions.shp 1 mile park.shp
   C:/gispy/data/ch07/ C:/gispy/scratch/ special_regions.shp '1 mile' park.shp

3. **buffer_clipv3.py** Revise 'buffer_clipv2.py', replacing the `sys.argv` approach for gathering user input with `GetParameterAsText` methods, and save it as 'buffer_clipv3.py'. Test 'buffer_clipv3.py' with the input example in the header comments of the script. Then test it with no arguments. Run sample script 'buffer_clipv2.py' with no arguments and explain what error occurs and why there is a difference in the errors.

4. **handlePaths.py** Practice using `os` module commands by writing a script that takes two arguments, a full path file name and a base file name of another file. Then use `os` commands in the script to print each of the following:

   (a) The base name of the first file name.
   (b) The extension of the first file name (make sure this works for files without extensions).
   (c) The full path file name of the second file, assuming it's in the same workspace as the first.
   (d) The size of the second file.
   (e) A list of files in the script's directory.

Example input: C:/gispy/data/ch07/park.shp xy1.txt

Example output:

```
The first file is: park.shp
The first file extension is: .shp
The full name of the second file is: C:/gispy/data/ch07\xy1.txt
The size of the second file is: 42 bytes
C:\gispy\sample_scripts\ch07 contains the following files:
['argSpacing.py', 'boundingGeom.py', 'boundingGeomV2.py',
'boundingGeomV3.py', 'buffer_clipv2.py', 'compact.py',
'copyFile.py', 'handlePaths.py', 'near.py', 'scriptPath.py']
```

5. **shape2kml.py** It takes two steps to convert shapefiles to KML using Esri tools:

   (a) Make a feature layer from the input shapefile with the Make Feature Layer (Data Management) tool.
   (b) Use the Layer to KML (Conversion) tool to convert the layer to KML.

   Write a script that converts a shapefile to a KML file. It should take four arguments: the workspace, the shapefile, the full path output KML file name, and the scale. Notice, the Layer to KML tool requires that the output name has a ".kmz" extension. So, for example, to convert the shapefile 'C:/gispy/data/ch07/park.shp' to a file named 'C:/gispy/scratch/park.kmz' at a scale of 1:24000, the script arguments should be as follows:

   Example input:
   C:/gispy/data/ch07/ park.shp C:/gispy/scratch/park.kmz 24000

   Example output:

   ```
   Temporary layer tempLayer created.
   C:/gispy/scratch/park.kmz created.
   ```

6. **calcGeom.py** Write a script that adds a new float field to a given polyline or polygon shapefile and uses the Calculate Field tool to calculate the length or area of each feature in the given shapefile and populate the new field with the result. The script should also allow the user to specify the unit of measure (e.g., acres, miles, meters,…). Use a Python expression (see Section 6.3.2) to specify the field calculation. Program this without using conditional constructs (discussed in an upcoming chapter). Instead, use the string `format` method to construct the Python expression based on the given geometry and unit of measure. The script should take four arguments ordered as follows: the full path file name of the input data, a field name, the geometric property (area or length), and a unit of measure. As an example, say you want to calculate the length of the trails in kilometers in the 'C:/gispy/data/ch07/trails.shp' file and you want to place the results in a new field named 'kmLength'. The input arguments for this scenario would appear as follows:

   C:/gispy/data/ch07/trails.shp kmLength length kilometers

7. **fileCompare.py** Write a script that compares two ASCII text files to determine if there are any differences. Use the `arcpy` File Compare (Data Management) tool which returns a `Result` object and writes a report on the differences between the files. Use the default setting for `file_type`. Set the `continue_compare` parameter to `'CONTINUE_COMPARE'`. The script should take three arguments, the full path file names of both input files, and the full path file name of the output file. It should create the output file and print feedback as shown in the example output. The file comparison verdict (true/false) is the second piece of information in the `Result` object. So, to print the true or false verdict, you'll need to use something like this:

   ```
   resultObj.getOutput(1).
   ```

   Example input:
   C:/gispy/data/ch07/xy1.txt C:/gispy/data/ch07/xy_current.txt C:/gispy/scratch/diffOutput.txt

   Example output:
   ```
   >>> Comparison results have been written to:
   C:/gispy/scratch/diffOutput.txt
   Are the input files (xy1.txt and xy_current.txt) the same? false
   ```