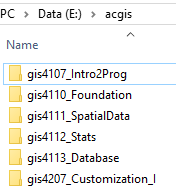
# {GIS4207 Describe and List Exercises

1. In the root of your acgis folder, create a **gis4207\_Customization\_I** folder



1. Copy MakeGIS4207Folders.bat provided to the **gis4207\_Customization\_I** folder. For the rest of this document, this folder will be referred to as the gis4207 folder.
2. Double-click MakeGIS4207Folders.bat
3. Download and follow the instructions in <https://github.com/viljoed/gis4x07/raw/master/GIS4x07_ExerciseSetup.docx> to create a Git repository for this exercise.
4. Copy/extract Data.zip in the gis4207 folder. When you are done, you should have a folder structure that looks something like:

gis4207\day02\lab\FredFandBarneyR

gis4207\Data

Please note that you will be able to fully develop (write, debug) these scripts in PyScripter. However, make sure they run from the command line as well. If you are using the Internal Engine, PyScripter keeps objects in memory that may not be in memory when you run from the command line in a new session.

## Describe01.py

Use the Describe function on data\Canada\province.shp to output the following:

BaseName: *value*

CatalogPath: *value*

DataType: *value*

Where *value* is the value returned from Describe. Make sure the path is relative to the script as outlined in WorkspaceRelativeToScript.ppt.

## Describe02.py

Same as Describe01.py but use string formatting to output the following:

Use the following format string: "{:13}: {}"

BaseName : *value*

CatalogPath: *value*

DataType : *value*

## Describe03.py

Same as Describe02.py but pass the feature class (including path) in on the command-line using sys.argv. If there is no argument, print the following usage:

Usage: Describe03.py <FeatureClassName>

## Describe04.py

Same as Describe03.py but add a delayed import of arcpy. That is, do not have the import arcpy at the top of the module where it will run every time whether or not there is an argument. Only import if there is an argument. Refer to DeferArcpyImport.pptx

## Describe05.py

Same as Describe04.py but add arcpy.Exists() before arcpy.Describe() to ensure the feature class exists before running the Describe. If the feature class does not exist, output the following.

*value* does not exist.

Where *value* is the name of the feature class passed as an argument.

## Describe06.py

Same as Describe05.py but add a field report that looks like the following

Field Name Field Type Length

---------- ---------- ------

somefield1 fieldtype1 6

somefield2 fieldtype2 13

somefield3 fieldtype3 20

For the headings, use the following code:

fmt = "{:15}{:15}{}"

headings = ("Field Name", "Field Type", "Length")

print fmt.format(\*headings)

The format string (fmt) is expecting three arguments. The headings tuple is one value. The \* in front of headings will unpack each element in the tuple so it can be used properly with the format string. Use the same format string with the following to create the underlines

(len(headings[0]) \* '-',

len(headings[1]) \* '-',

len(headings[2]) \* '-')

Understand the above code.

Use the following format string for each record in the output

For the field names, types, and length, modify the format string so the field length is right-justified from the letter “t” in “Length” as shown at the beginning of Describe06.py

## List01.py

Use ListFeatureClasses() to print the name of each feature class in the data\SanFrancisco workspace (one feature class per line) Make sure the path is relative to the script as outlined in WorkspaceRelativeToScript.ppt.

## List02.py

Same as List01.py but pass in the workspace as an argument on the command line (sys.argv) as you did with Describe03.py. Add delayed import as you did with Describe04.py and use Exists as you did in Describe05.py.

## List03.py

Same as List02.py but use feature\_type as an argument to filter the feature classes by feature type (e.g. list only Line feature classes). Make sure both the workspace and FeatureType provided as arguments are valid before the rest of the script executes.

## List04.py

Use ListWorkspaces() to list all workspaces given a root folder passed at the command line. Use the delayed import, display usage, use .Exists, etc. as you have in previous scripts.

## List05.py

Use os.walk() to output every folder and sub-folder starting at a *rootFolder* provided in the command-line.

Please note that arcpy is not needed for this script. To check if a folder exists or not, use os.path.exists(path). If there is no argument, print

Usage: List05.py <rootFolder>

If the provided rootFolder does not exist, print

*value* does not exist.

Where *value* is the name of the rootFolder.

## List06.py

Use os.walk() and arcpy.ListWorkspaces() to output the full path of every workspace in every folder and sub-folder starting at a rootFolder provided in the command-line. If there is no argument or the rootFolder does not exist, use code similar to List05.py to inform the user. To print the full path, use os.path.abspath(*path*). If *path* is a relative path, it will be returned as an absolute path. If it is an absolute path, it will be returned as is.

## List07.py

Same as List06.py but output a complete list of feature classes in each workspace as well. So this code will output the full path to every feature class in the entire folder tree under the provided rootFolder.

## List08.py

Same as List07.py with another command line parameter (outFileName). In this script, the list of Workspaces and feature classes will be written to the file specified.

Make sure you check for the right number of command-line arguments.