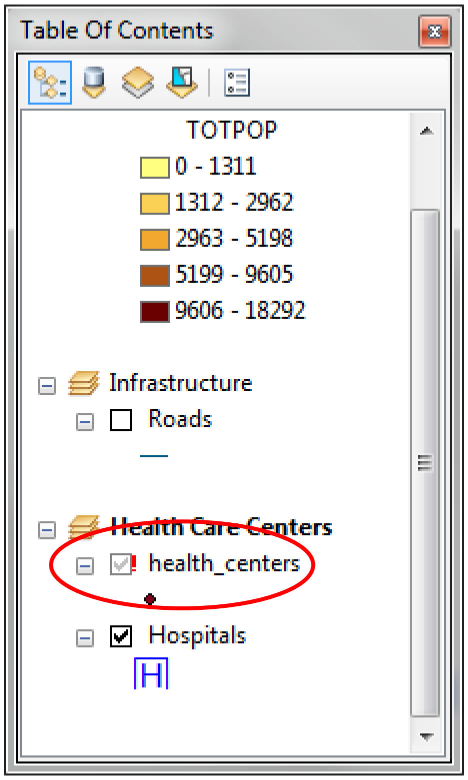
Chapter 9: Managing documents and Layers

9.1 Dealing with broken data sources

- Data sources can become broken sometimes either because the data has been moved or the data has been deleted.
- The result can be broken data sources in many map document files.
- ListBrokenDataSources() returns a list of layer objects that have a broken data connection.
- In ArcMap, a broken data connection is signified by the red exclamation point just before the layer name.



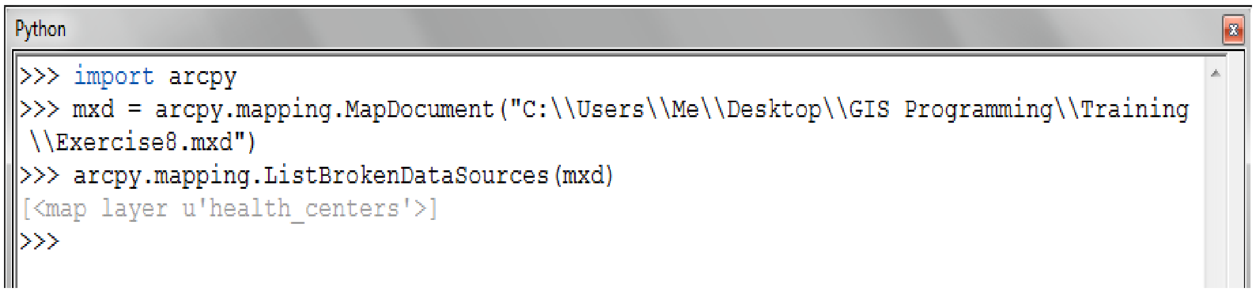
- Typically, the ListBrokenDataSources() function is used as the first step in a script that iterates through the list and fixes the data source.
- For loops on a list provide an easy mechanism to iterate through each item in the list
- Example:

```
for lyr in arcpy.mapping.ListBrokenDataSources(mxdDoc):

        Do Something
```

- Syntax

    ListBrokenDataSources (map_document_or_layer)

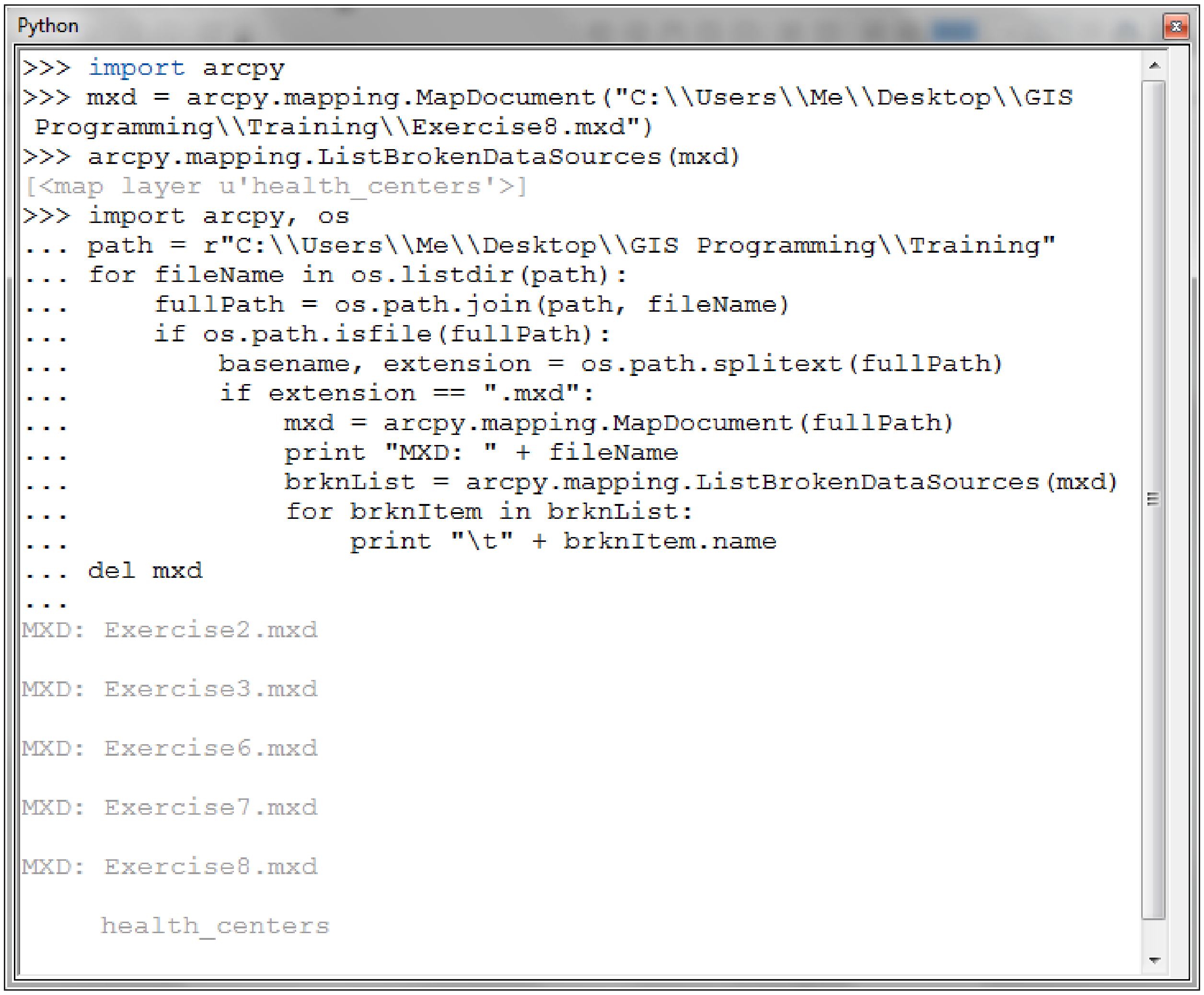
Example:



```
>>> import arcpy
>>> mxd = arcpy.mapping.MapDocument("C:\\Users\\Me\\Desktop\\GIS Programming\\Training
 \\Exercise8.mxd")
>>> arcpy.mapping.ListBrokenDataSources(mxd)
[<map layer u'health_centers'>]
>>>
```

Example 2: listBrokenDataSources2.py

This script will search for broken data sources in all map documents that exist in a single folder. A report with map document names and broken sources will be printed.

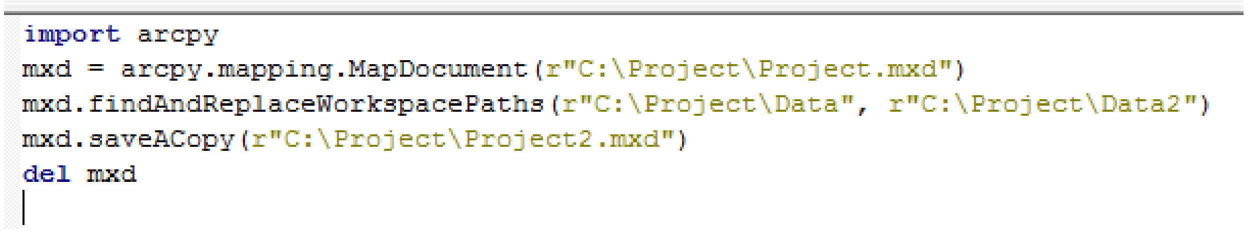


```
>>> import arcpy
>>> mxd = arcpy.mapping.MapDocument("C:\\Users\\Me\\Desktop\\GIS
 Programming\\Training\\Exercise8.mxd")
>>> arcpy.mapping.ListBrokenDataSources(mxd)
[<map layer u'health_centers'>]
>>> import arcpy, os
... path = r"C:\\Users\\Me\\Desktop\\GIS Programming\\Training"
... for fileName in os.listdir(path):
...     fullPath = os.path.join(path, fileName)
...     if os.path.isfile(fullPath):
...         basename, extension = os.path.splitext(fullPath)
...         if extension == ".mxd":
...             mxd = arcpy.mapping.MapDocument(fullPath)
...             print "MXD: " + fileName
...             brknList = arcpy.mapping.ListBrokenDataSources(mxd)
...             for brknItem in brknList:
...                 print "\t" + brknItem.name
... del mxd
...
MXD: Exercise2.mxd

MXD: Exercise3.mxd

MXD: Exercise6.mxd

MXD: Exercise7.mxd

MXD: Exercise8.mxd

     health_centers
```

- There are numerous reasons why data sources need to be repaired or redirected to different locations.
- The idea of making these changes manually in every affected map document can be overwhelming.

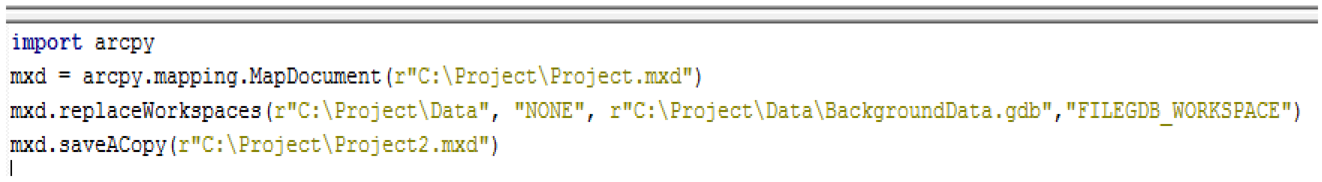
- Methods are available with the arcpy.mapping scripting environment that makes it possible to automate these changes without even having to open a map document.
- You have control of updating data sources for individual layers, or you can update all layers at once that have a common workspace.
- Are few definitions are required before we move forward

    - A **workspace** is a container for data. A workspace can be a folder that contains items like shapefiles, CAD files, or rasters, or it can be an ArcInfo coverage, personal geodatabase, file geodatabase, connection to SDE, and so on.
    - A **workspace path** is the system path to the workspace. In the case of file-based data (for example, CAD, shapefiles, rasters), it includes the name of the folder that contains the data. With file-based geodatabases, it includes the name of the geodatabase. SDE workspace paths are defined by the system path to the SDE connection file.
    - A **workspace type** is an ArcGIS supported data model format. For example, shapefile, personal geodatabase, file geodatabase, CAD, SDE, and so on.
    - A **dataset** is the feature class or table that is in a workspace. When a dataset name is required, it is the name of the object in the workspace, not the way it appears in a map document's table of contents.
    - A **data source** for a layer or table is the combination of the workspace and dataset.
- Arcpy contains 3 main classes for fixing broken data sources
    - MapDocument
    - Layer
    - TableView
- Each of these classes contain methods which can be used to fix data sources.
- The MapDocument class contain 2 methods for fixing broken data sources
    - MapDocument.**findAndReplaceWorkspacePaths** (find_workspace_path, replace_workspace_path, {validate})
        - used to perform a global find and replace workspace paths for all layers and tables in a map document that share the workspace
        - can replace the paths to multiple workspace types at once
            - cad, shapefiles, geodatabases,…
    - Example:
        - In this scenario, the data was located directly under the C:\Project\Data folder but was moved into a subfolder called Data2. This script updates a single map document.



```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
mxd.findAndReplaceWorkspacePaths(r"C:\Project\Data", r"C:\Project\Data2")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

    - MapDocument.**replaceWorkspaces** (old_workspace_path, old_workspace_type, new_workspace_path, new_workspace_type, {validate})
        - Similar to findAndReplaceWorkspacePaths
        - ALLOWS you to switch from one workspace type to another (e.g. File geodatabase to personal geodatabase)
        - ONLY allows you to work on one workspace type at a time
        - Example
          This scenario involves updating two different workspaces in a map document. First, shapefiles are redirected to a file geodatabase called Parcels.gdb. Second, layers from a personal geodatabase are redirected to another file geodatabase called Transportation.gdb.



```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
mxd.replaceWorkspaces(r"C:\Project\Data", "NONE", r"C:\Project\Data\BackgroundData.gdb","FILEGDB_WORKSPACE")
mxd.saveACopy(r"C:\Project\Project2.mxd")
```

- Both Layer and TableView also have findAndReplaceWorkspacePath methods
- These find and replace a SINGLE path for a Layer or Table
- In the cases for a Layer, it can be in a map document or a layer file
- For a Table, it can find and replace the path in a map document
- Both the Layer and TableView classes also have replaceDataSource method.
- This method is similar to findAndReplaceWorkspacePath
    - But in addition to being able to change a workspace type, the dataset can also be changed.
    - This is the only method that enables you to replace a layer or table dataset within a workspace.
    - Useful when the dataset name has been changed in the workspace
    - Layer.replaceDataSource (workspace_path, workspace_type, dataset_name, {validate})
    - TableView.replaceDataSource (workspace_path, workspace_type, dataset_name, {validate})

The optional validate parameter

- This parameter allows you to first test to see if the new data source exists before changing or replacing a layer or table's data source.
- If validate is True and the new data source exists and is valid, the original data source will be updated to the new data source; otherwise, it will remain pointing to the original data source and nothing will be updated.
- If validate is False, the new data source does not have to be valid, that is, it might not already exist. This can be useful for scenarios that require data sources to be updated ahead of the data

being created. Note, in these cases, the data would appear broken in the associated map documents or layer files.

  - When using the findAndReplaceWorkspacePath or findAndReplaceWorkspacePaths methods, the validate parameter applies to the replace_workspace_path parameter.
  - When using the replaceWorkspacePaths method, the validate parameter applies to the new_workspace_path and new_workspace_type parameters.
  - When using the replaceDataSource method, the validate parameter applies to the dataset_name parameter.

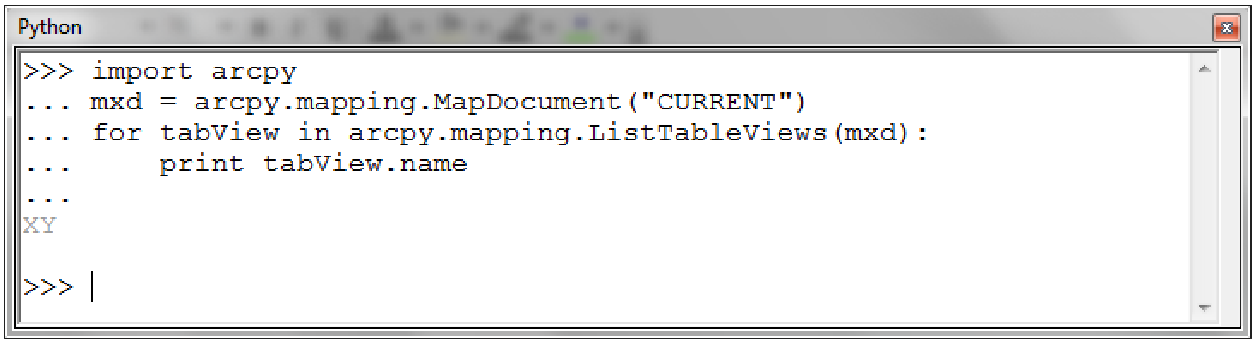
- See http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//00s30000004p000000.htm for more information.

9.2 ListTableView

- Returns a Python list of TableView objects that exist within a map document (.mxd)
- The TableView object references a stand-alone table within the ArcMap table of contents and provides access to a table's name, data source, and definition query properties.
- ListTableViews always returns a list object even if only one table is returned.
- In order to return a TableView object, an index value must be used on the list (e.g., aTable = arcpy.mapping.ListTableViews(mxd)[0]).
- For loops on a list provide an easy mechanism to iterate through each item in the list (e.g., for aTable in arcpy.mapping.ListTableViews(mxd):).
- Wildcards are used on the name property and are not case sensitive. A wildcard string of "so*" will return a layer with a name Soils. Wildcards can be skipped in the scripting syntax simply by passing an empty string (""), an asterisk (*), or entering wildcard=None, or nothing at all if it is the last optional parameter in the syntax.
- It is possible that there might be tables in a map document that have the same name. If that is the case, then other properties may need to be used to isolate a specific layer. Properties such as a tables's datasource or definitionQuery could be used to do this. It is ideal that all tables in a map document be uniquely named.
- Syntax

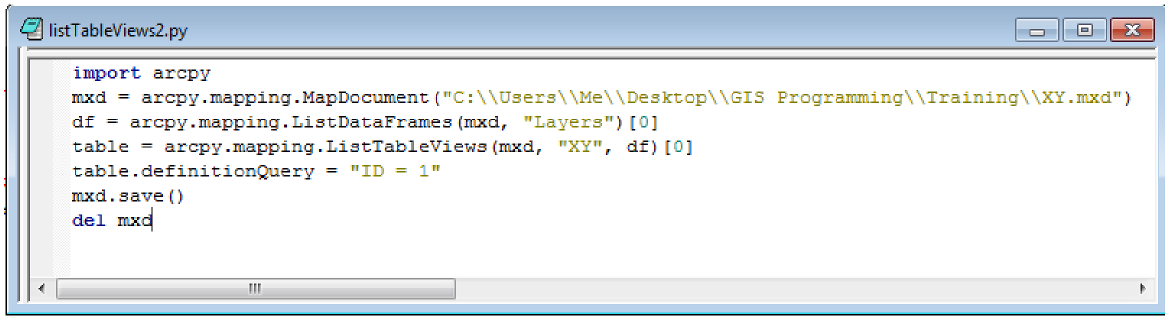
  ListTableViews (map_document, {wildcard}, {data_frame})

- Example: XY.mxd
  In this example the name of all tables will be printed using a for…loop. Both the name of the current map document and the table are XY in this case.



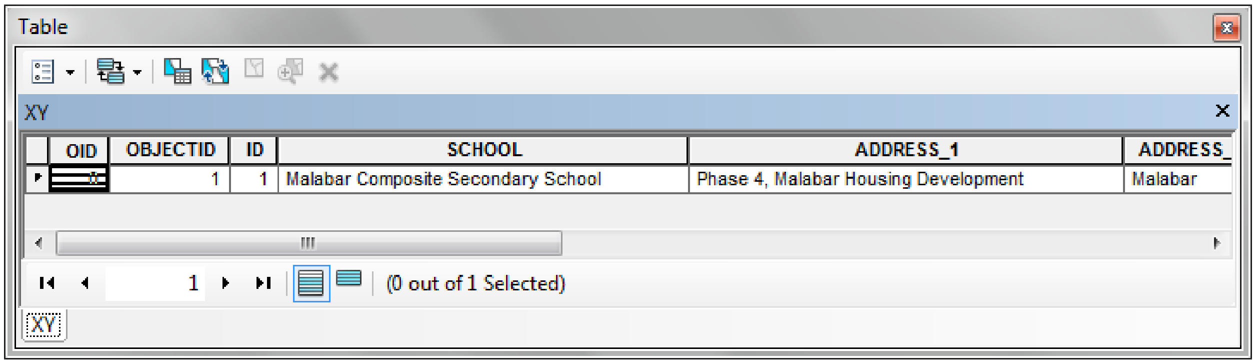
```
Python
>>> import arcpy
... mxd = arcpy.mapping.MapDocument("CURRENT")
... for tabView in arcpy.mapping.ListTableViews(mxd):
...     print tabView.name
...
XY

>>> 
```

- Example2: listTableView2.py
  The following script finds a table called ‘XY’ in a data frame named ‘Layers’ and sets a definition query



```
listTableViews2.py
import arcpy
mxd = arcpy.mapping.MapDocument("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\XY.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "Layers")[0]
table = arcpy.mapping.ListTableViews(mxd, "XY", df)[0]
table.definitionQuery = "ID = 1"
mxd.save()
del mxd
```

Open the attribute table of the XY.dbf table in the XY.mxd to verify that the definition query was set.



Table

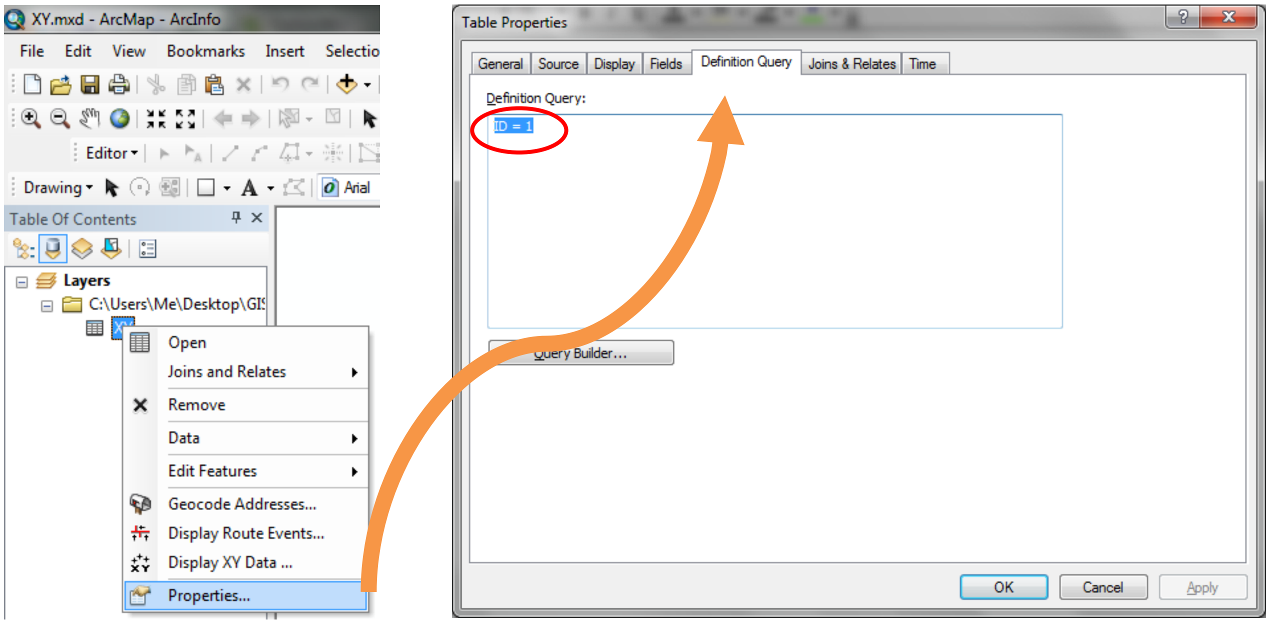
XY

OID	OBJECTID	ID	SCHOOL	ADDRESS_1	ADDRESS_
0	1	1	Malabar Composite Secondary School	Phase 4, Malabar Housing Development	Malabar

1 (0 out of 1 Selected)

XY

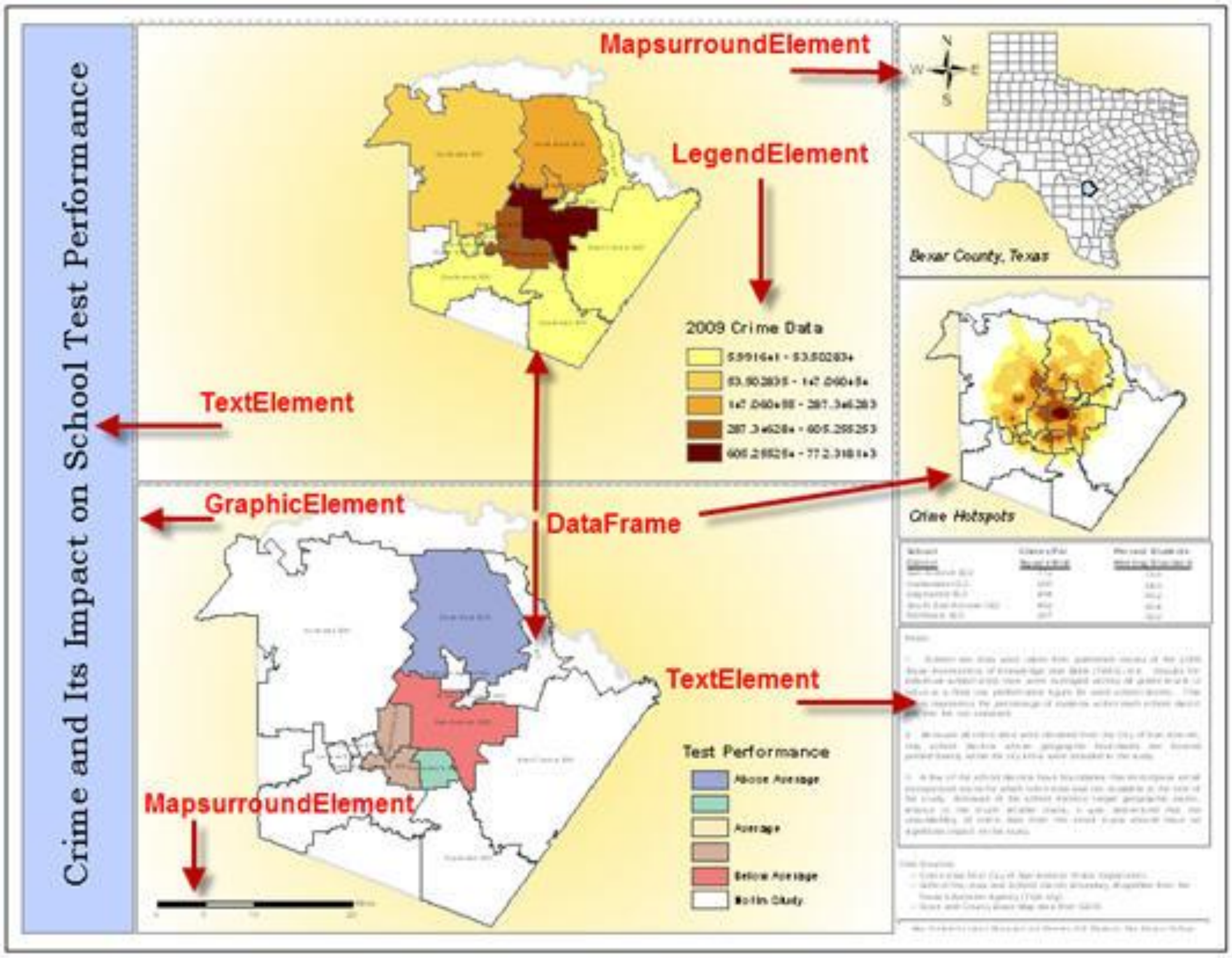
To see the definition query in ArcMap



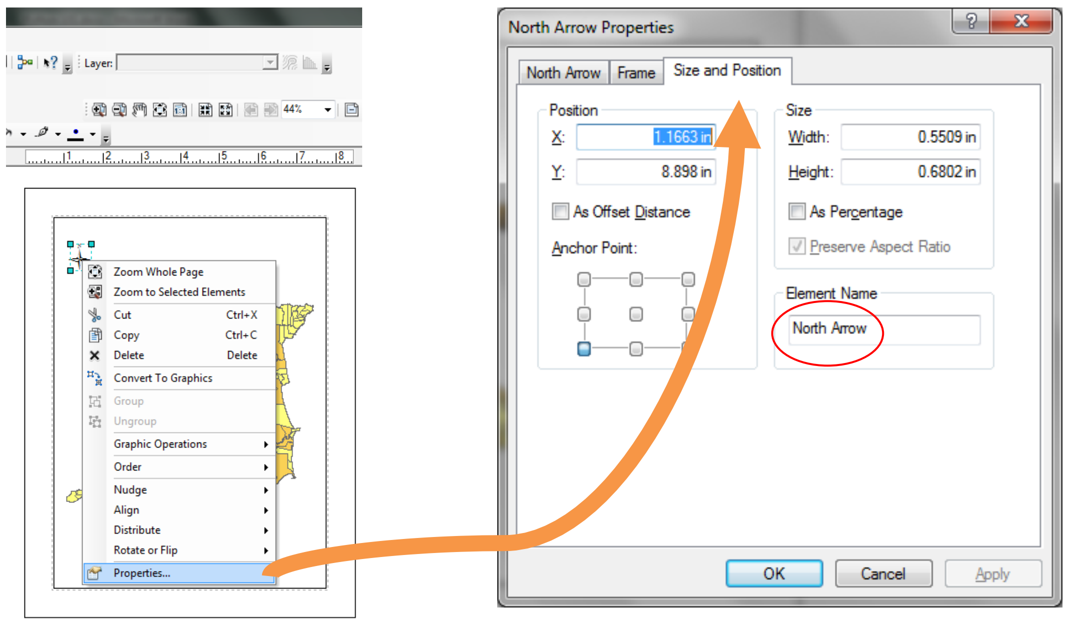
Definition Query:
ID = 1

9.3 ListLayoutElements

- Returns a Python list of layout elements that exist within a map document (.mxd) layout
- ListLayoutElements always returns a Python list object even if only one page element is returned.
- In order to return an element object, an index value must be used on the list (e.g., elm = arcpy.mapping.ListLayoutElements(mxd)[0]).
- For loops on a list provide an easy mechanism to iterate through each item in the list (e.g., for elm in arcpy.mapping.ListLayoutElements(mxd):).
- ListLayoutElements only returns elements from a page layout and not map annotation elements that may exist within a data frame.
- Each element can be one of the following classes: GraphicElement, LegendElement, PictureElement, TextElement, MapSurroundElement, or PictureElement.



- Each page element has a name property that can be set within the element properties dialog box within ArcMap (located on the Size and Position tab).



Position
X: 1.1663 in
Y: 8.898 in
As Offset Distance
Anchor Point:

Size
Width: 0.5509 in
Height: 0.6802 in
As Percentage
Preserve Aspect Ratio

Element Name
North Arrow

- It is the map document author's responsibility to ensure each page element is given a unique name so that elements can be uniquely identified.
- If two elements have the same name, there is no way for certain to ensure it is the element you want to reference.
- Syntax

  ListLayoutElements (map_document, {element_type}, {wildcard})

- ListLayoutElements will also return the elements within a group element into a flattened list. This makes it possible to easily search and replace text strings, for example, without having to navigate through a group element structure.
- The element_type parameter can be skipped simply by passing an empty string ("") or entering element_type=None.
- Wildcards are used on the name property and are not case sensitive. A wildcard string of "*title" will return a page element with a name Main Title. Wildcards can be skipped in the scripting syntax simply by passing an empty string (""), an asterisk (*), or entering wildcard=None, or nothing at all if it is the last optional parameter in the syntax.

Parameter	**Explanation**	**Data Type**
map_document	A variable that references a MapDocument object.	MapDocument
element_type	A string that represents the element type that will be used to filter the returned list of elements. • DATAFRAME_ELEMENT —Dataframe element • GRAPHIC_ELEMENT —Graphic element • LEGEND_ELEMENT —Legend element • MAPSURROUND_ELEMENT —Mapsurround element • PICTURE_ELEMENT —Picture element • TEXT_ELEMENT —Text element (The default value is None)	String
wildcard	A combination of asterisks (*) and characters can be used to help limit the results. (The default value is None)	String

Example 1:

This script will search all text elements, including elements in a group, that have a text value of Old String and replace that value with New String.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd, "TEXT_ELEMENT"):
    if elm.text == "Old String":
        elm.text = "New String"
mxd.save()
del mxd
```

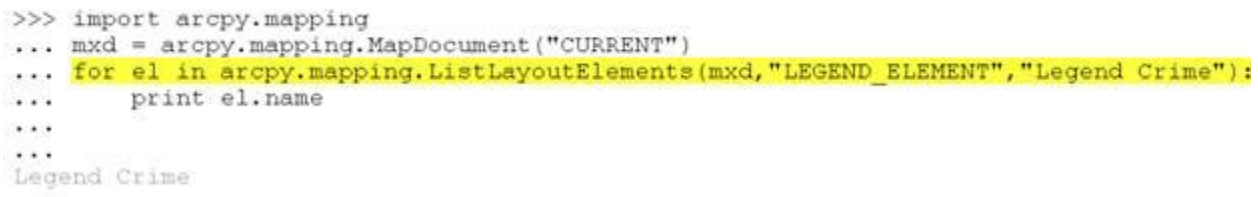
Example 2:

The following script will find a picture element using a wildcard and then change the picture's data source.

```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for elm in arcpy.mapping.ListLayoutElements(mxd,"PICTURE_ELEMENT", "*logo*"):
    if elm.name == "CityLogo":
        elm.sourceImage = r"C:\Project\Data\Photo.bmp"
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

Example 3:

In this example we are searching for legend type elements with the name ‘Legend Crime”



```
>>> import arcpy.mapping
... mxd = arcpy.mapping.MapDocument("CURRENT")
... for el in arcpy.mapping.ListLayoutElements(mxd,"LEGEND_ELEMENT","Legend Crime"):
...     print el.name
...
...
Legend Crime
```

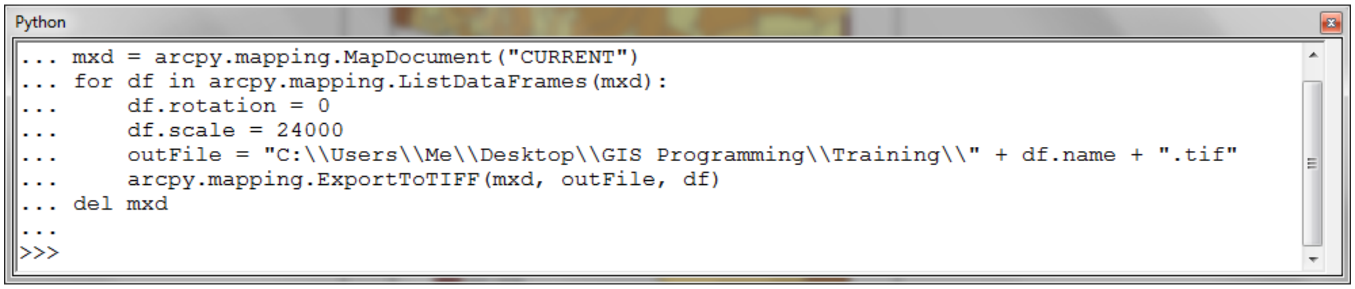
9.4 Exercise 8

9.5 Layout classes

- The **data frame layout class** provides access to data frame properties in the .mxd
- This object can work with both map units and page layout units depending upon the property being used. For example, it can be used to set map extent, scale, and rotation, as well as items like spatial reference. The DataFrame object can also be positioned (elementPositionX, elementPositionY) and/or sized (elementWidth, elementHeight) on the layout using page units.
- The DataFrame object also provides access to informational items like credits and description.
- A reference to a data frame can also be very useful when trying to reference other objects as well. For example (and as we have seen before), the ListLayers function provides an optional parameter called data_frame so that layers can be searched within a single data frame rather than the entire map document.
- The DataFrame object is also used as an optional parameter to distinguish printing or export a data frame versus a page layout. For this reason it is important to uniquely name each data frame so a specific data frame can be easily referenced.

Example: dataFrame.py

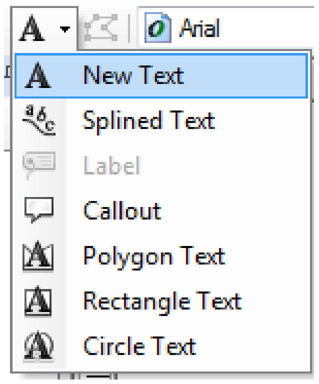
The following script will set each data frame's rotation property to 0 and scale property to 1:24000 before exporting each data frame to an individual output TIFF file using the data frame name property as the name of the output file.



```
... mxd = arcpy.mapping.MapDocument("CURRENT")
... for df in arcpy.mapping.ListDataFrames(mxd):
...     df.rotation = 0
...     df.scale = 24000
...     outFile = "C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\" + df.name + ".tif"
...     arcpy.mapping.ExportToTIFF(mxd, outFile, df)
... del mxd
...
>>>
```

See http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/DataFrame/00s300000003000000/ for more on methods for the data frame class.

- The **graphic element layout class** is a generic class for elements which can be added to the page layout
    - Tables
    - Graphs
    - Groups of elements
    - Neatlines
    - Markers
    - Lines
    - Area shapes
- MAKE SURE to give each element name a unique name for reasons mentioned before if you intend to access it through a Python script.
- http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/GraphicElement/00s300000040000000/
- **Task:**
    - Try the example given at the bottom of the web page from the link above. You are to use the current map, add a graphic element and give the element the name "Title Block"
- The **legend layout element class** provides options for positioning, modifying the title and getting legend items and the parent data frame.
- The LegendElement object has an association with a single parent data frame
- See http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/LegendElement/00s300000041000000/ for more on the legend element. Also check out the example at this link.
- The **Mapsurround layout element** can refer to north arrows, scale bars and scale texts.
- Like the legend element is associated with a single data frame
- Properties of this object permit repositioning on the page
- **Task:**
    - http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/MapsurroundElement/00s300000035000000/ Try to make the example the link given work using the CURRENT map document.
- The **picture element layout object** represents a raster or image that has been inserted into a page layout
- Properties permit repositioning
- In addition to the standard element properties, the PictureElement is unique in that is has a sourceImage property that allows you to read or modify the source location.
    - Useful when you have to change the picture
        - E.g. writing a script to change the logo in all map document files at a particular location.
- http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/PictureElement/00s30000000s000000/
- The **text layout element** represent text inserted into a page layout

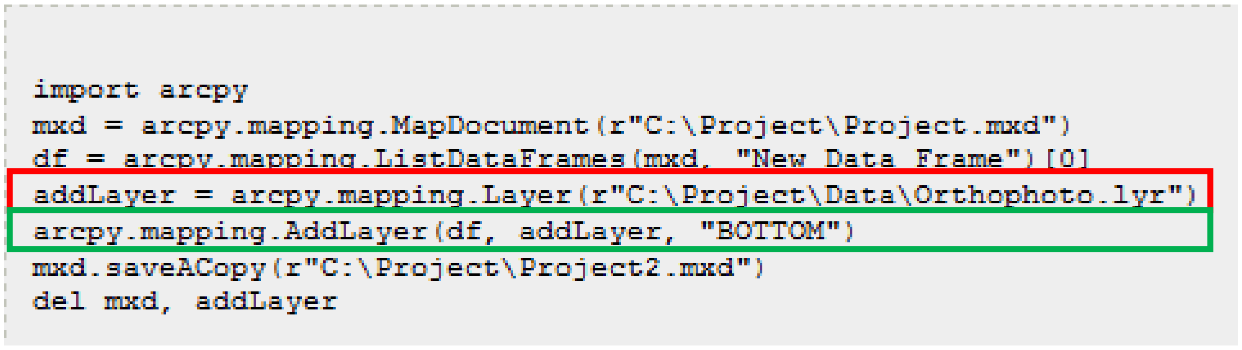


- DOES NOT include legend text or text as part of a table
- Properties include repositioning and modifying the text string
    - Useful if changing the text in multiple locations within the same and across layouts
- Check out example 1 at http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/TextElement/00s30000000m000000/

9.6 Working with layers

Adding Layers

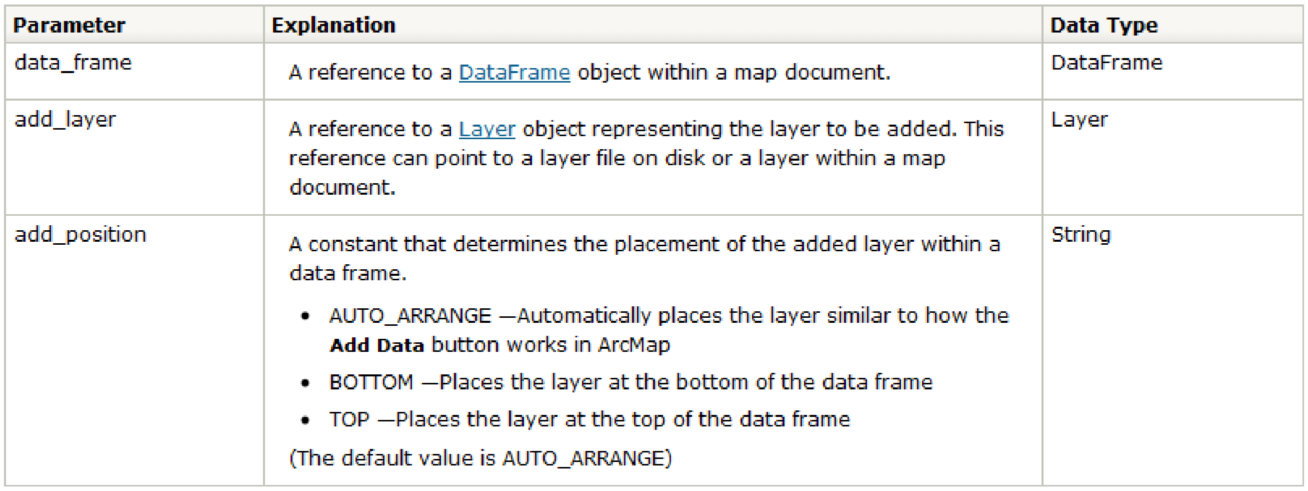
- Can add layers or group layers to and existing .mxd file
- Can use “auto arrange” to place layer in data frame for visibility
    - Like existing add data button
    - Based on layer weight rules and geometry type
- CANNOT add layers to a .lyr file
- When adding a layer you MUST reference an existing layer
    - Layer file on disk
    - Same map document and data frame
    - Same map document, different data frame
    - Separate map document
- The ListLayers function is ideal for finding and referencing specific layers in a map document or layer file.
- You must fist create an instance of a layer and then call AddLayer() function passing in this instance and rules for how it is to be positioned.



```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "New Data Frame")[0]
addLayer = arcpy.mapping.Layer(r"C:\Project\Data\Orthophoto.lyr")
arcpy.mapping.AddLayer(df, addLayer, "BOTTOM")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, addLayer
```

- Syntax

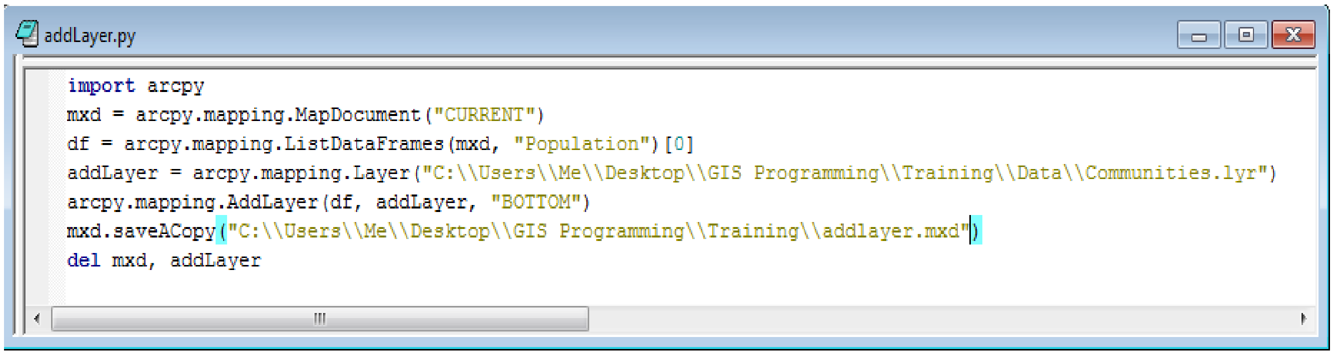
    AddLayer (data_frame, add_layer, {add_position})



Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object within a map document.	DataFrame
add_layer	A reference to a Layer object representing the layer to be added. This reference can point to a layer file on disk or a layer within a map document.	Layer
add_position	A constant that determines the placement of the added layer within a data frame. • AUTO_ARRANGE —Automatically places the layer similar to how the **Add Data** button works in ArcMap • BOTTOM —Places the layer at the bottom of the data frame • TOP —Places the layer at the top of the data frame (The default value is AUTO_ARRANGE)	String

Example: addLayer.py

In the following example, a Communities layer (.lyr) is added to the current map document and a new map document is saved. In this example a .lyr is read in.



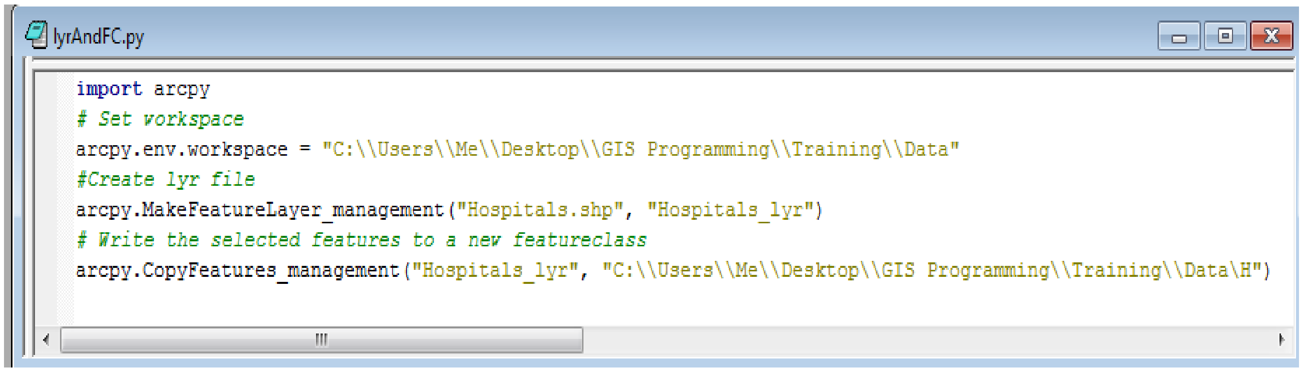
```
import arcpy
mxd = arcpy.mapping.MapDocument("CURRENT")
df = arcpy.mapping.ListDataFrames(mxd, "Population")[0]
addLayer = arcpy.mapping.Layer("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\Communities.lyr")
arcpy.mapping.AddLayer(df, addLayer, "BOTTOM")
mxd.saveACopy("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\addlayer.mxd")
del mxd, addLayer
```

- To read in a shape file, the shape file must first be converted to a layer file and then used.
- MakeFeatureLayer creates a feature layer from an input feature class or layer file http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//00170000006p000000.htm

- The layer that is created by the tool is temporary and will not persist after the session ends unless the layer is saved to disk or the map document is saved.
- The temporary feature layer can be saved as a layer file using the Save_To_Layer_File tool or can be saved as a new feature class using the Copy_Features tool.

Example: lyAndFC

In this example, the hospital shapefile is converted to a layer file ("Hospital.lyr") and this file converted to a shapefile ("H.shp")



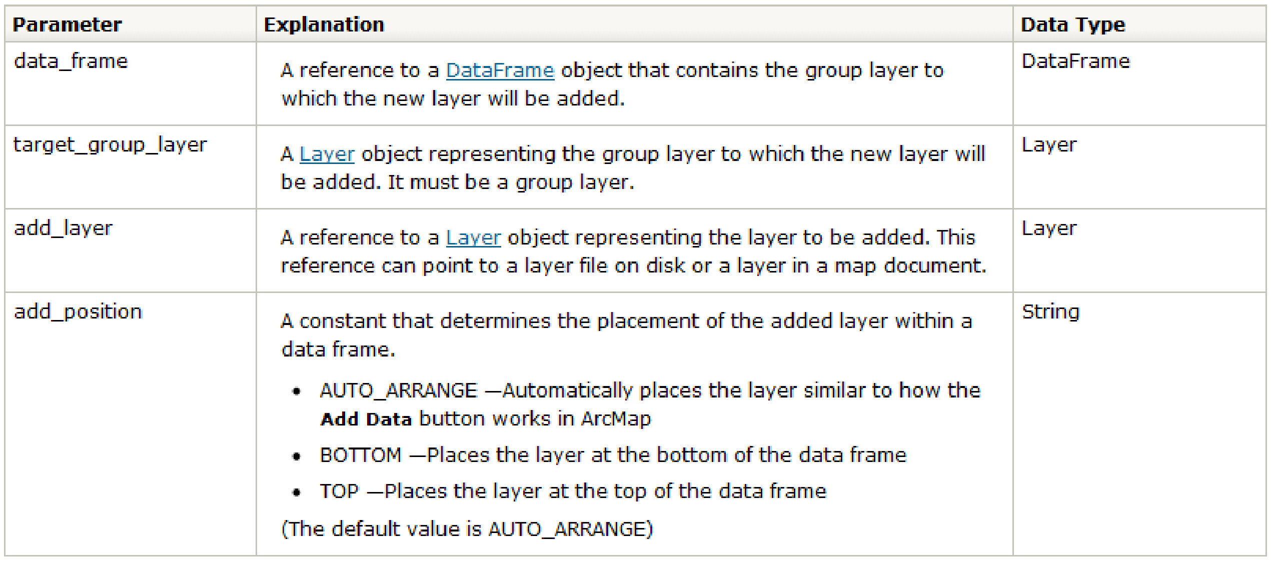
lyrAndFC.py

```
import arcpy
# Set workspace
arcpy.env.workspace = "C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data"
#Create lyr file
arcpy.MakeFeatureLayer_management("Hospitals.shp", "Hospitals_lyr")
# Write the selected features to a new featureclass
arcpy.CopyFeatures_management("Hospitals_lyr", "C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\H")
```

Adding to group layer

- Provides the ability to add a layer to a group layer within a map document (.mxd) using simple placement options
- AddLayerToGroup is an easy way to add a layer into an already existing group layer.
- It can add a layer with auto-arrange logic that places the new layer in a group layer similarly to how the Add Data button works in ArcMap; it places the layer based on layer weight rules and geometry type.
- The other placement choices are either at the top or the bottom of a group layer.
- AddLayerToGroup is the only function that can add a layer into an empty group layer.
- AddLayerToGroup does not allow you to add layers to group layers within a layer file. Layer files should be managed and authored using ArcMap.
- Syntax

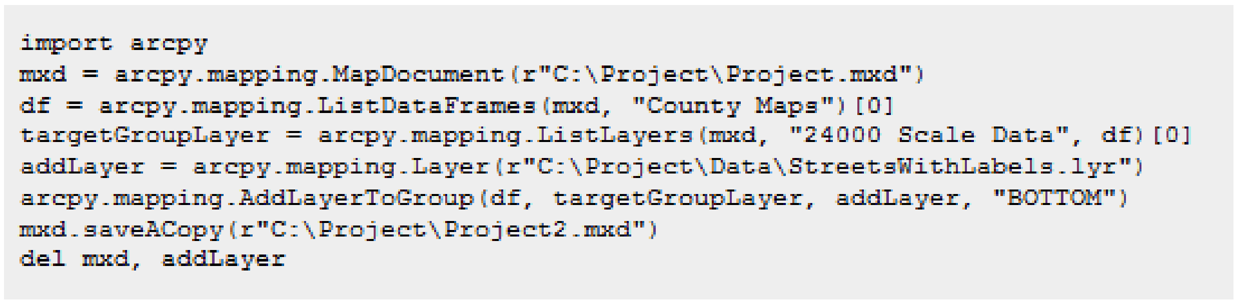
AddLayerToGroup (data_frame, target_group_layer, add_layer, {add_position})



Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the group layer to which the new layer will be added.	DataFrame
target_group_layer	A Layer object representing the group layer to which the new layer will be added. It must be a group layer.	Layer
add_layer	A reference to a Layer object representing the layer to be added. This reference can point to a layer file on disk or a layer in a map document.	Layer
add_position	A constant that determines the placement of the added layer within a data frame. • AUTO_ARRANGE —Automatically places the layer similar to how the **Add Data** button works in ArcMap • BOTTOM —Places the layer at the bottom of the data frame • TOP —Places the layer at the top of the data frame (The default value is AUTO_ARRANGE)	String

Example

The following script will add a new layer from a layer (.lyr) file on disk and place it at the bottom of a group layer called 24000 Scale Data in a data frame called County Maps.

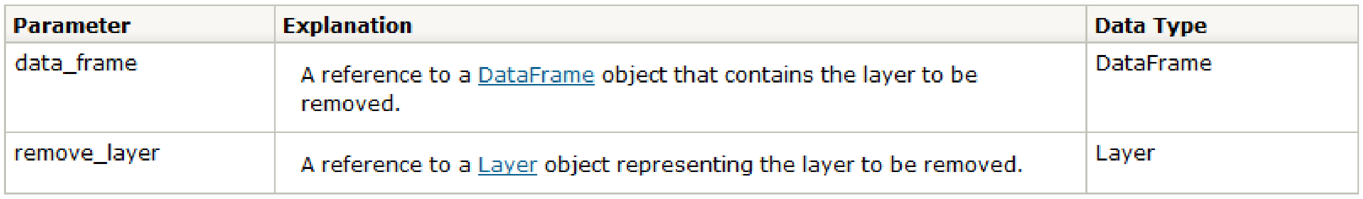


```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
targetGroupLayer = arcpy.mapping.ListLayers(mxd, "24000 Scale Data", df)[0]
addLayer = arcpy.mapping.Layer(r"C:\Project\Data\StreetsWithLabels.lyr")
arcpy.mapping.AddLayerToGroup(df, targetGroupLayer, addLayer, "BOTTOM")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, addLayer
```

Removing layers

- Provides the ability to remove a layer within a data frame in a map document (.mxd).
    - Single or group layer from specific data frame
- If there is more than one layer that meets the criteria, then only the first layer will be removed
    - Unless the script iterates through each layer in a returned list.
- Syntax

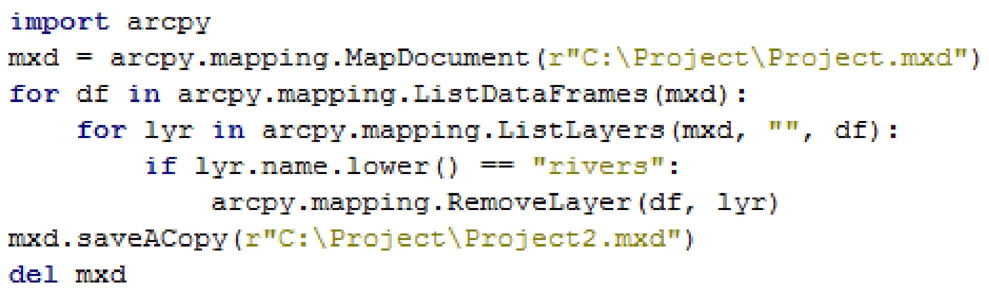
  RemoveLayer (data_frame, remove_layer)



Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be removed.	DataFrame
remove_layer	A reference to a Layer object representing the layer to be removed.	Layer

Example

The following script will remove all layers with the name Rivers from a map document.

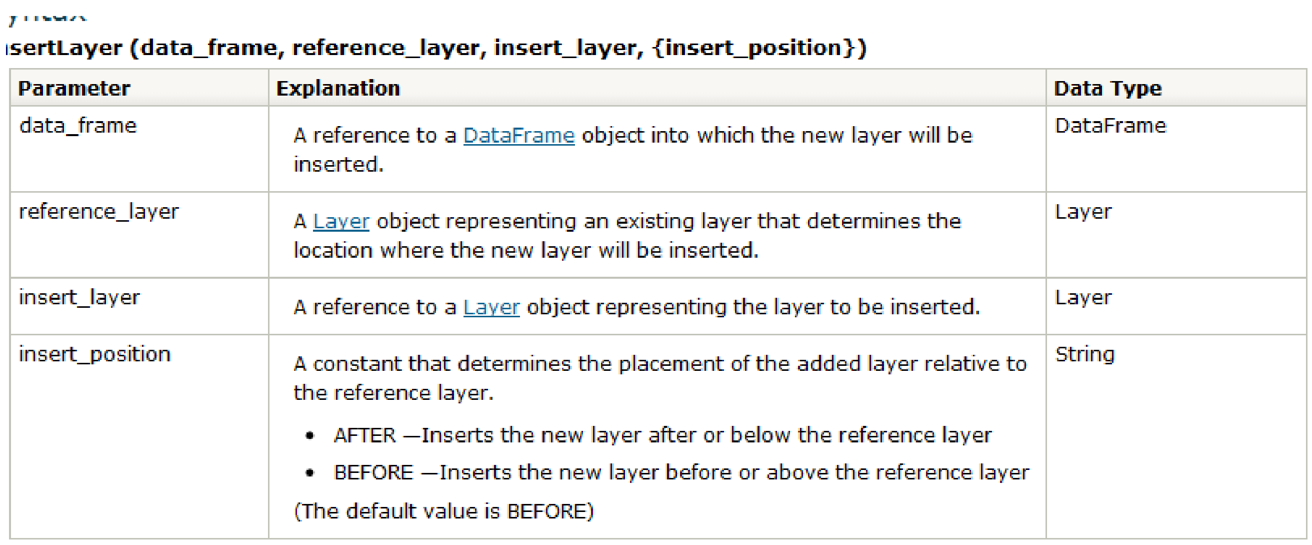


```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
for df in arcpy.mapping.ListDataFrames(mxd):
    for lyr in arcpy.mapping.ListLayers(mxd, "", df):
        if lyr.name.lower() == "rivers":
            arcpy.mapping.RemoveLayer(df, lyr)
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd
```

Inserting layers

- Allows for more precise positioning of a new layer into a data frame or group layer
- Uses a reference layer to specify location
- The layer to be inserted can be placed before or after the reference layer
- CANNOT add layer to empty data frame or empty group layer
- Syntax

    InsertLayer (data_frame, reference_layer, insert_layer, {insert_position})

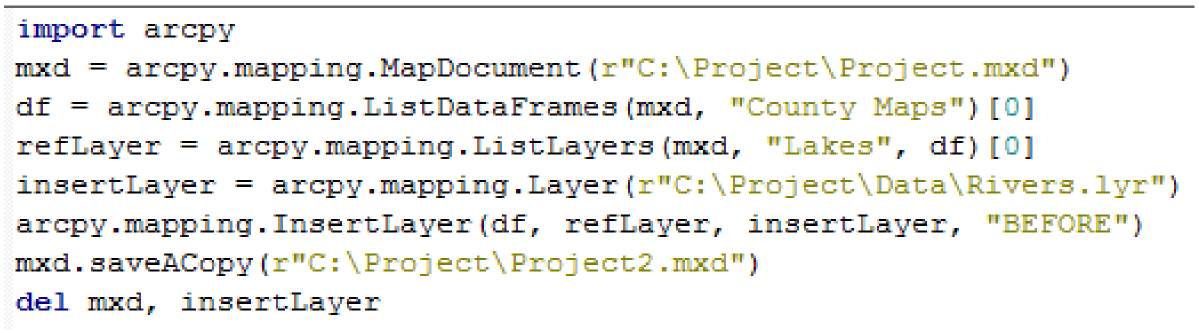


sertLayer (data_frame, reference_layer, insert_layer, {insert_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object into which the new layer will be inserted.	DataFrame
reference_layer	A Layer object representing an existing layer that determines the location where the new layer will be inserted.	Layer
insert_layer	A reference to a Layer object representing the layer to be inserted.	Layer
insert_position	A constant that determines the placement of the added layer relative to the reference layer. • AFTER —Inserts the new layer after or below the reference layer • BEFORE —Inserts the new layer before or above the reference layer (The default value is BEFORE)	String

Example:

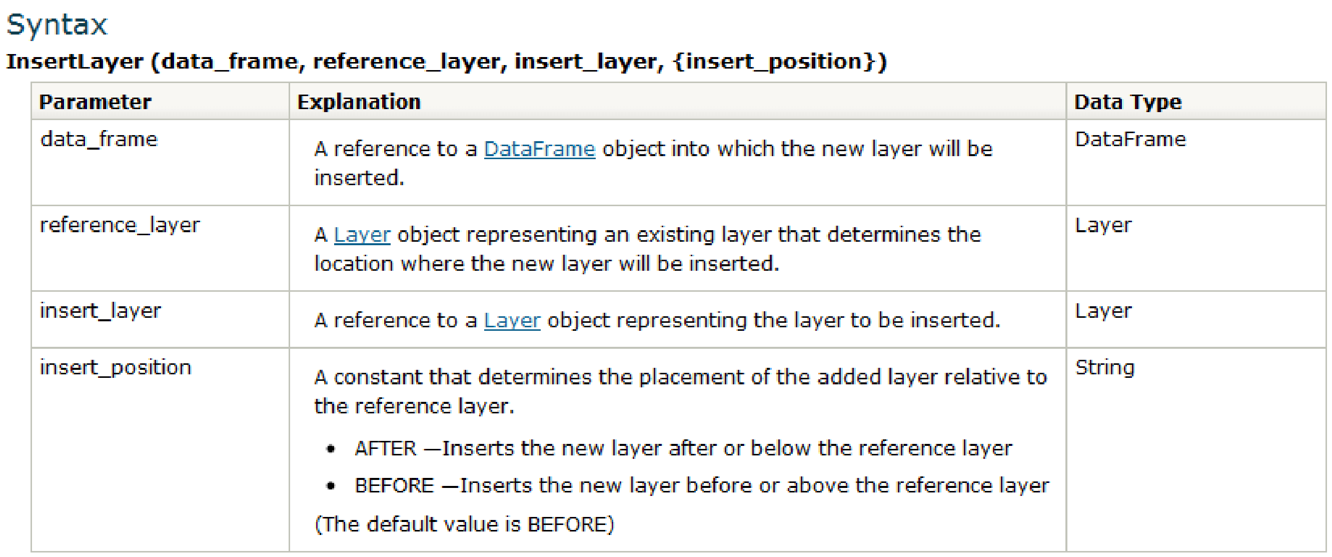
The follwing script will insert a new layer from a layer (.lyr) file on disk and place it before a layer called Lakes which is in a data frame called County Maps.



```
import arcpy
mxd = arcpy.mapping.MapDocument(r"C:\Project\Project.mxd")
df = arcpy.mapping.ListDataFrames(mxd, "County Maps")[0]
refLayer = arcpy.mapping.ListLayers(mxd, "Lakes", df)[0]
insertLayer = arcpy.mapping.Layer(r"C:\Project\Data\Rivers.lyr")
arcpy.mapping.InsertLayer(df, refLayer, insertLayer, "BEFORE")
mxd.saveACopy(r"C:\Project\Project2.mxd")
del mxd, insertLayer
```

Moving layers

- MoveLayer() provides the ability to reposition a layer within a data frame or group layer
- MUST BE in the same data frame
- Uses a reference layer
- Very similar to insertLayer() with the exception that the layer already exists within the data frame.



Syntax

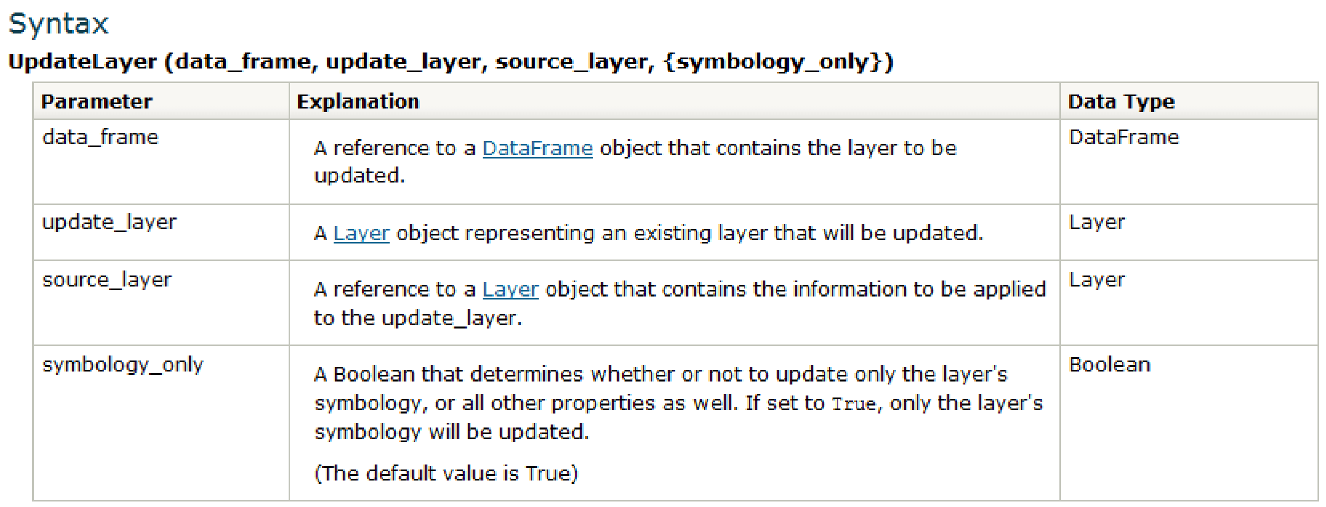
InsertLayer (data_frame, reference_layer, insert_layer, {insert_position})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object into which the new layer will be inserted.	DataFrame
reference_layer	A Layer object representing an existing layer that determines the location where the new layer will be inserted.	Layer
insert_layer	A reference to a Layer object representing the layer to be inserted.	Layer
insert_position	A constant that determines the placement of the added layer relative to the reference layer. • AFTER —Inserts the new layer after or below the reference layer • BEFORE —Inserts the new layer before or above the reference layer (The default value is BEFORE)	String

Updating layers

- UpdateLayer() can update layer properties or just the symbology of the layer
- The information is extracted from a source layer and applied to the layer in a map document that needs to be updated.

- The source_layer can either be a layer (.lyr) file or a layer within a map document.
- limited number of layer properties
- default behavior is updating symbology
- Another option is to update all or some specific layer properties in addition to symbology. For example, you may want to update the field aliases, selection symbology, query definitions, and so on, for a layer that exists in many map documents. An easy way of doing this is to use ArcMap to modify the layer with the appropriate properties and then save the layer out to a layer (.lyr) file. The layer file can then be used as a source_layer to update all the properties for a given layer.



Syntax

UpdateLayer (data_frame, update_layer, source_layer, {symbology_only})

Parameter	Explanation	Data Type
data_frame	A reference to a DataFrame object that contains the layer to be updated.	DataFrame
update_layer	A Layer object representing an existing layer that will be updated.	Layer
source_layer	A reference to a Layer object that contains the information to be applied to the update_layer.	Layer
symbology_only	A Boolean that determines whether or not to update only the layer's symbology, or all other properties as well. If set to `True`, only the layer's symbology will be updated. (The default value is True)	Boolean

- To update the symbology for a layer you MUST first make sure that the source layer and update layer has the same feature geometry
- Depending on the renderer (e.g., unique value or graduated color using a particular attribute), the attribute definitions also need to be the same.
- Can also be used to update specific or all layer properties, field aliases, selection symbology, query definitions, label fields, etc.
- UpdateLayer() is also capable of switching out unrelated layers
    - E.g. replacing a polygon layer with a group layer
    - To do this, the “symbology_only” option MUST BE set to false
- Check out the examples at http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/UpdateLayer/00s30000003p000000/

9.7 Exercise 9