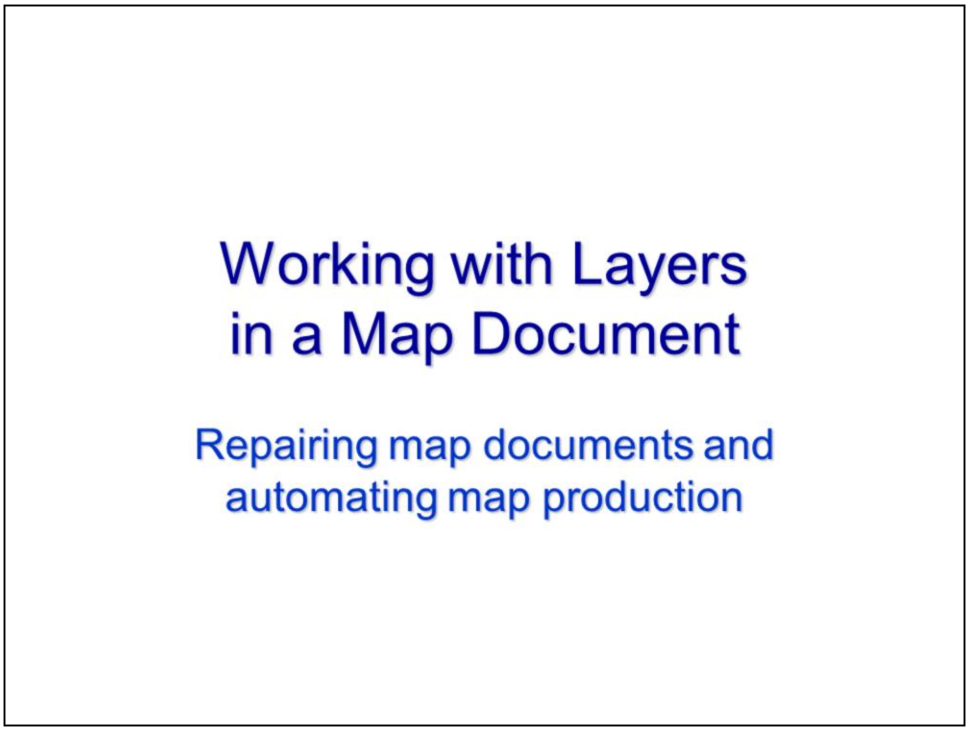
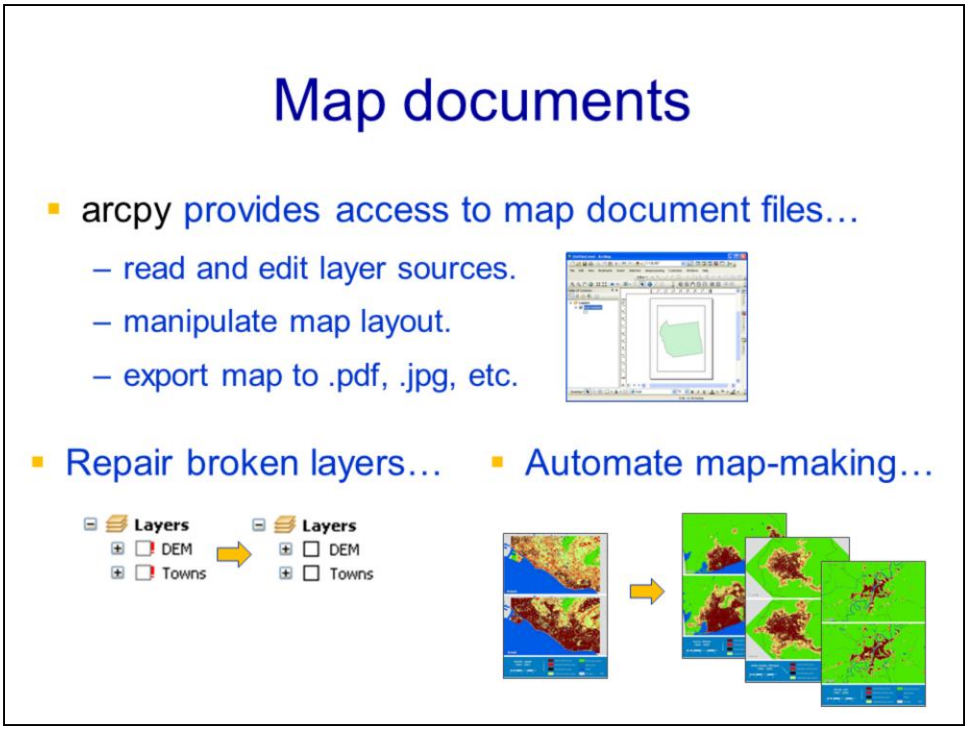


Working with Layers in a Map Document

Repairing map documents and automating map production

This video will discuss how to work with map documents as well as how to work with layers.



Map documents

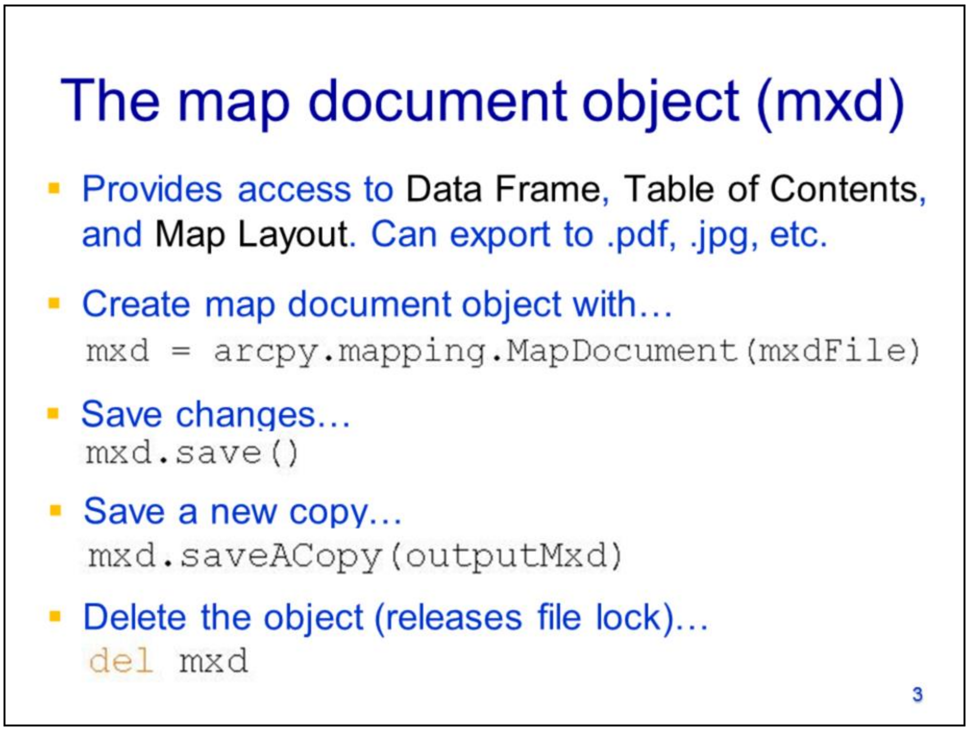
- arcpy provides access to map document files…
  - read and edit layer sources.
  - manipulate map layout.
  - export map to .pdf, .jpg, etc.
- Repair broken layers…
- Automate map-making…

Layers
DEM
Towns
Layers
DEM
Towns

Arcpy allows a script to access map document files.

This access allows a script to repair broken data layers in the map document

Or use a template to automate the production of a set of maps.



The map document object (mxd)

- Provides access to Data Frame, Table of Contents, and Map Layout. Can export to .pdf, .jpg, etc.
- Create map document object with…

```
mxd = arcpy.mapping.MapDocument(mxdFile)
```

- Save changes…

```
mxd.save()
```

- Save a new copy…

```
mxd.saveACopy(outputMxd)
```

- Delete the object (releases file lock)…

```
del mxd
```

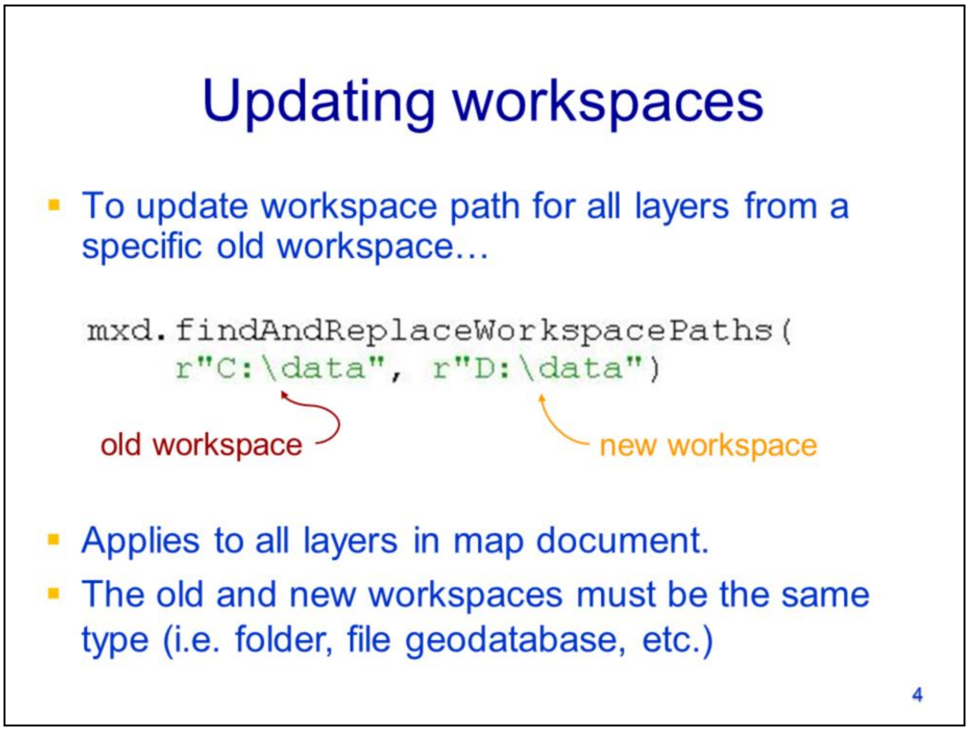
3

The map document object provides access to the **data frame(s)**, **table of contents**, and the **map layout**. It also allows maps to be exported as a .jpg or .pdf.

The **mapping submodule** contains all of arcpy’s capabilities for working with map documents. To get **the map document object**, use the **MapDocument** method in the mapping module.

The map document object can be saved either with the current name or as a copy.

To unlock the file., the map document object should be deleted when you are finished with it.



Updating workspaces

- To update workspace path for all layers from a specific old workspace…

```
mxd.findAndReplaceWorkspacePaths(
     r"C:\data", r"D:\data")
```

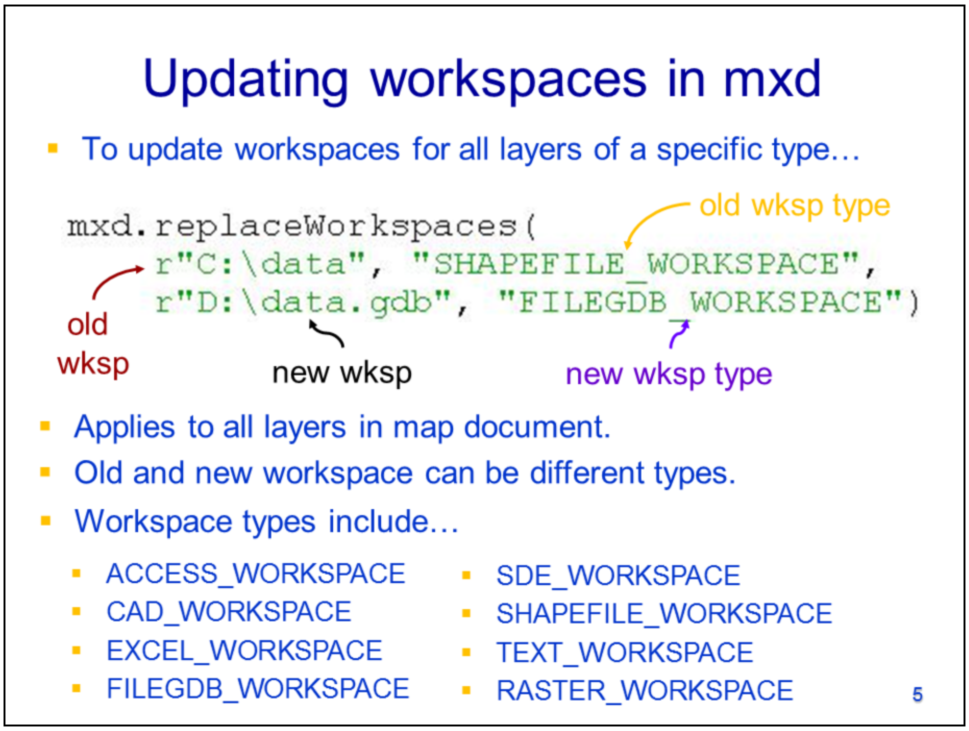
old workspace
new workspace

- Applies to all layers in map document.
- The old and new workspaces must be the same type (i.e. folder, file geodatabase, etc.)

4

The workspaces for all layers in the map document's table of contents can be changed using the map document object's **findAndReplaceWorkspacePaths** tool. This tool can be useful when the data referenced by layers in the map document have been relocated.

This tool applies to all layers in a map document. Note that the old and new workspaces must be the same type.



Updating workspaces in mxd

- To update workspaces for all layers of a specific type…

```
mxd.replaceWorkspaces(
        r"C:\data", "SHAPEFILE_WORKSPACE",
        r"D:\data.gdb", "FILEGDB_WORKSPACE")
```

old wksp type
old wksp
new wksp
new wksp type

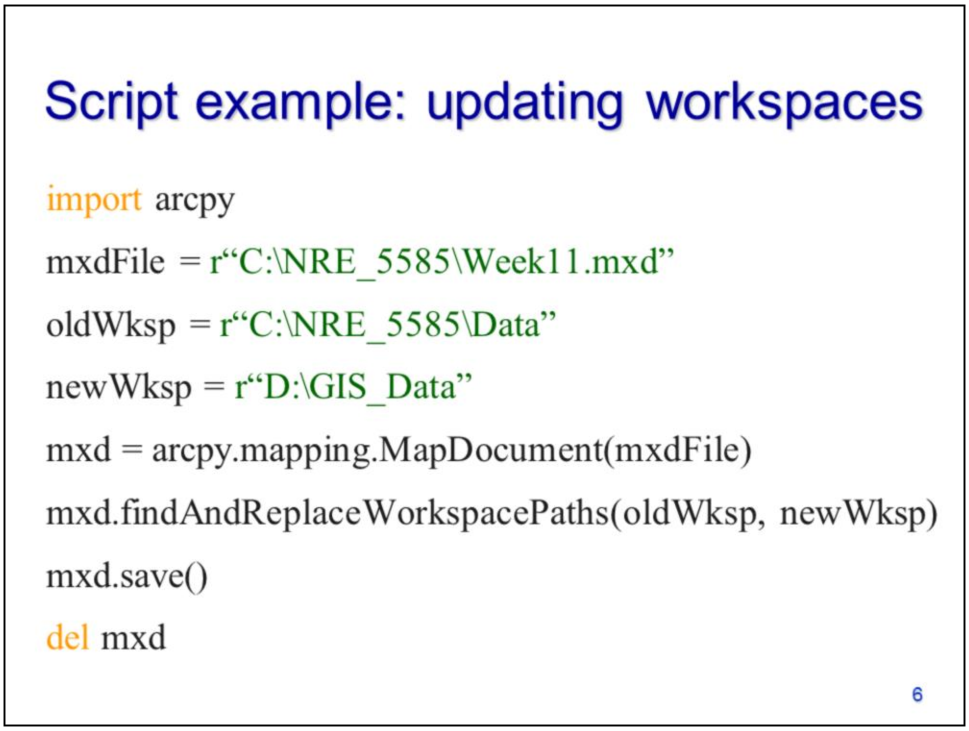
- Applies to all layers in map document.
- Old and new workspace can be different types.
- Workspace types include…
  - ACCESS_WORKSPACE
  - CAD_WORKSPACE
  - EXCEL_WORKSPACE
  - FILEGDB_WORKSPACE
  - SDE_WORKSPACE
  - SHAPEFILE_WORKSPACE
  - TEXT_WORKSPACE
  - RASTER_WORKSPACE

5

The map document object’s **replaceWorkspaces** tool will update the workspace only for a specific type of layer.

The tool applies to all layers that have the data type associated with the original workspace. The new workspace can have a different type than the original workspace.

The types of available workspaces are listed here.



Script example: updating workspaces

```
import arcpy
mxdFile = r"C:\NRE_5585\Week11.mxd"
oldWksp = r"C:\NRE_5585\Data"
newWksp = r"D:\GIS_Data"
mxd = arcpy.mapping.MapDocument(mxdFile)
mxd.findAndReplaceWorkspacePaths(oldWksp, newWksp)
mxd.save()
del mxd
```

6

In this slide, we’ll see an example script that demonstrates how to update the workspaces for all layers in a map document.

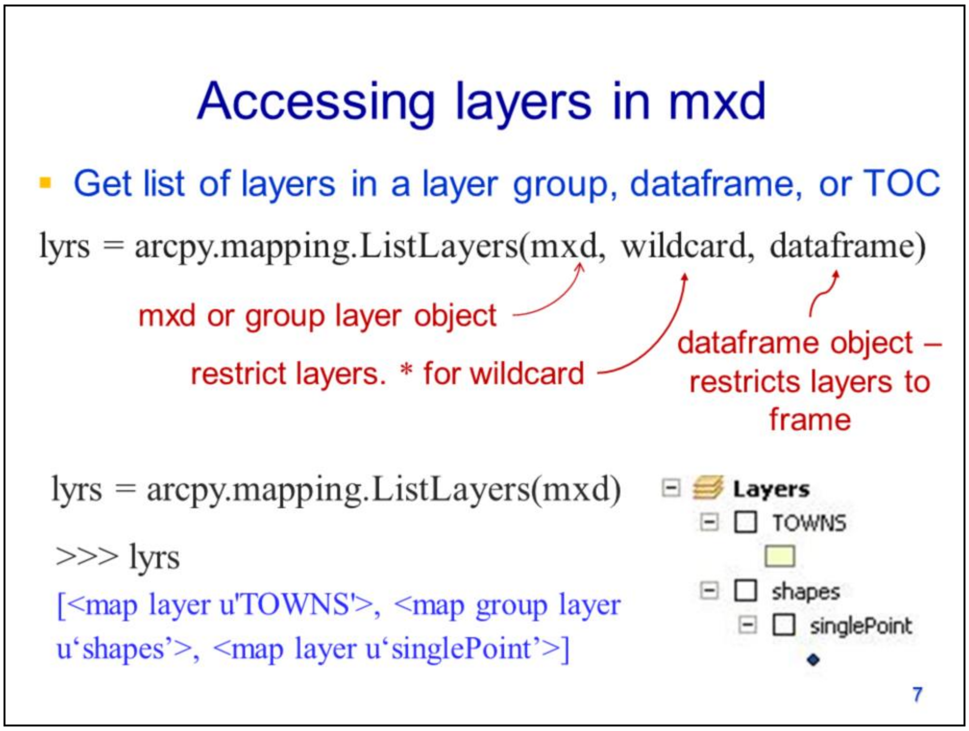
Note that an ArcMap map document file has the .mxd extension.

The old workspace is where the data were originally located; the new workspace is where the data have been moved.

The map document object is created for the associated .mxd file using the mapping module’s MapDocument method.

The **findAndReplaceWorkspacePaths** tool is used to update the workspaces for all layers in the map document.

The map document is saved and the map document object is deleted.


Accessing layers in mxd

- Get list of layers in a layer group, dataframe, or TOC

```
lyrs = arcpy.mapping.ListLayers(mxd, wildcard, dataframe)
```

mxd or group layer object
restrict layers. * for wildcard
dataframe object – restricts layers to frame

```
lyrs = arcpy.mapping.ListLayers(mxd)
>>> lyrs
[<map layer u'TOWNS'>, <map group layer u'shapes'>, <map layer u'singlePoint'>]
```

Layers
TOWNS
shapes
singlePoint
7

A list of the layers in the map document's Table of Contents can be retrieved using the **ListLayers** method.

The map document object is the only required parameter.

A wildcard may be specified to restrict the names of the layers that are returned – the asterisk is used as the wildcard character, as usual.

The dataframe object can also be specified to restrict the returned layers to those contained in a specific dataframe. We'll see in a later video how to create the dataframe object.

This statement shows an example of retrieving all layers in the table of contents..

The layers are returned as layer objects in a python list. Note that group layers (i.e. "shapes") are also returned.

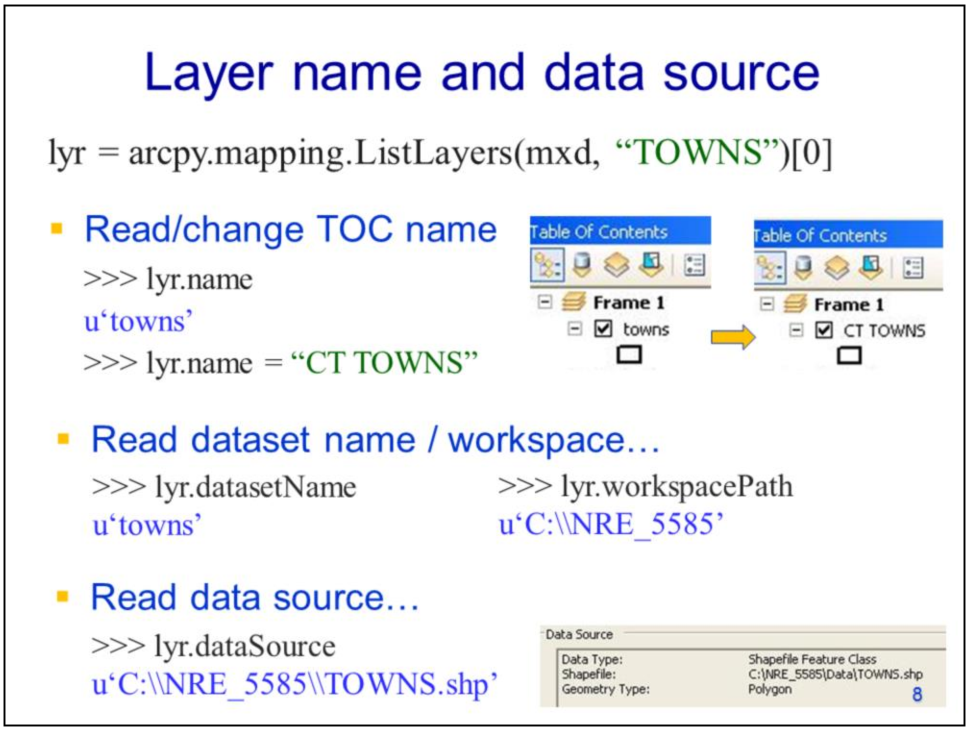


Table Of Contents
Frame 1
towns
Table Of Contents
Frame 1
CT TOWNS

Layer name and data source

```
lyr = arcpy.mapping.ListLayers(mxd, "TOWNS")[0]
```

- Read/change TOC name

```
>>> lyr.name
u'towns'
>>> lyr.name = "CT TOWNS"
```

- Read dataset name / workspace…

```
>>> lyr.datasetName
u'towns'
```

```
>>> lyr.workspacePath
u'C:\\NRE_5585'
```

- Read data source…

```
>>> lyr.dataSource
u'C:\\NRE_5585\\TOWNS.shp'
```

Data Source

Data Type:	Shapefile Feature Class
Shapefile:	C:\NRE_5585\Data\TOWNS.shp
Geometry Type:	Polygon

8

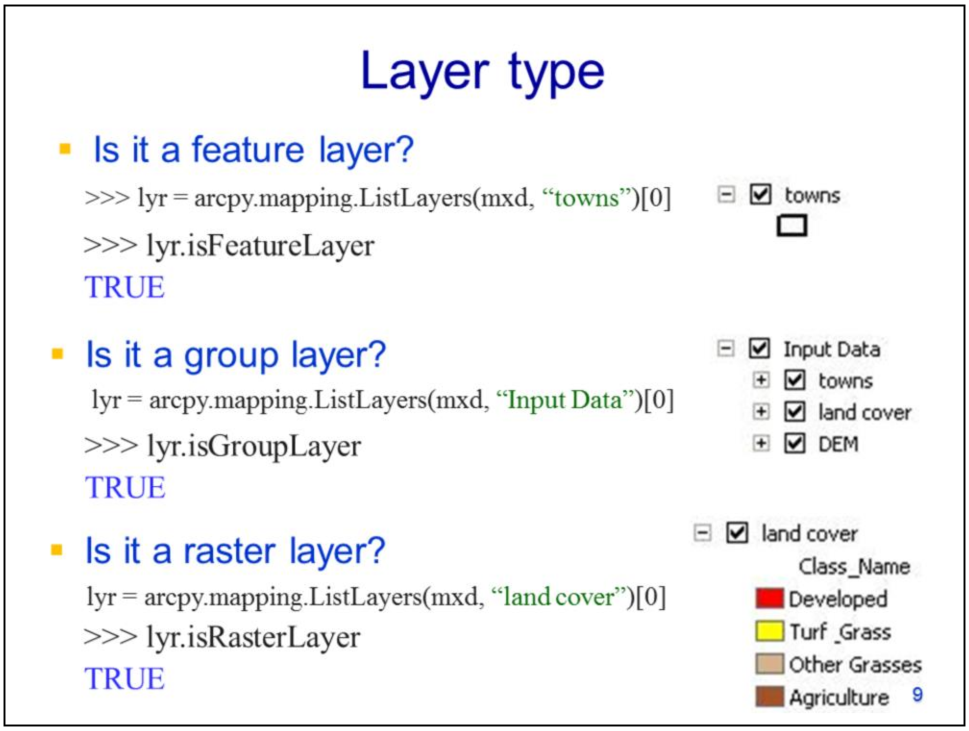
This slide will explore some properties of layer objects. The example statement retrieves the layer object corresponding to the "TOWNS" layer – we will use this layer object for all examples on the next few slides.

The **name** property allows a layer's name in the table of contents to be read or changed.

The **datasetName** property provides the basename of the dataset that is referenced by the layer - note that the extension is not included.

The **workspacePath** property provides the workspace of the layer's dataset.

The **dataSource** property provides the path name for the layer's dataset. Note that the datasetName, workspacePath, and dataSource properties are read-only and so cannot be modified.



Layer type

- Is it a feature layer?

```
>>> lyr = arcpy.mapping.ListLayers(mxd, "towns")[0]
>>> lyr.isFeatureLayer
TRUE
```

- Is it a group layer?

```
lyr = arcpy.mapping.ListLayers(mxd, "Input Data")[0]
>>> lyr.isGroupLayer
TRUE
```

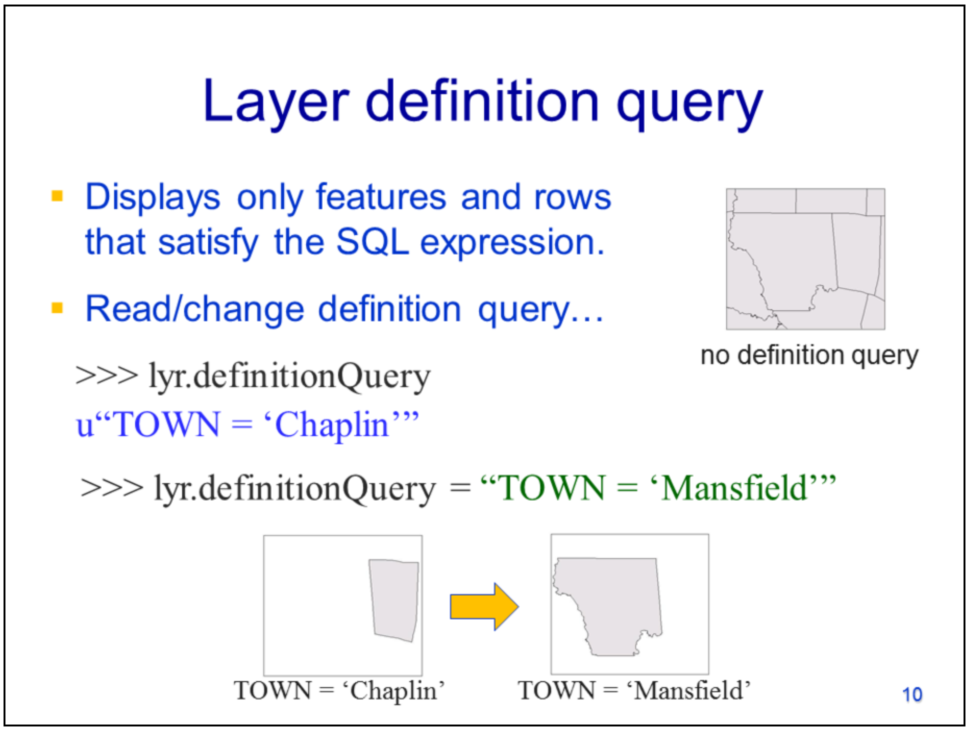
- Is it a raster layer?

```
lyr = arcpy.mapping.ListLayers(mxd, "land cover")[0]
>>> lyr.isRasterLayer
TRUE
```

The **isFeatureLayer** property returns a True value if the layer corresponds to a feature class; otherwise a False value will be returned.

The **isGroupLayer** property will return a True value if the layer object is a group layer.

The **isRasterLayer** property will return a True value if the layer object corresponds to a raster.



Layer definition query

- Displays only features and rows that satisfy the SQL expression.
- Read/change definition query…

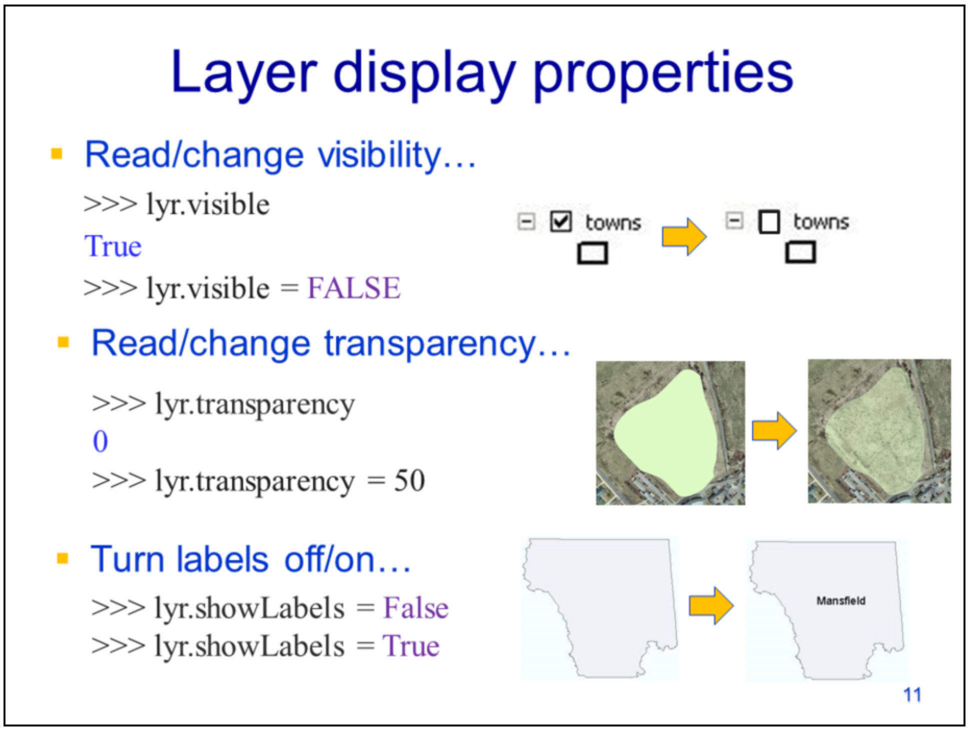
```
>>> lyr.definitionQuery
u"TOWN = 'Chaplin'"
>>> lyr.definitionQuery = "TOWN = 'Mansfield'"
```

no definition query
TOWN = 'Chaplin'
TOWN = 'Mansfield'
10

A layer's **definition query** can be used to restrict the features and rows that are displayed – only features and rows that satisfy the SQL expression will be displayed in ArcMap.

The **definitionQuery** property can be read or set a definition query using a standard SQL expression.

In this example, the definitionQuery is changed from "Town = Chaplin" to "Town = Mansfield".



Layer display properties

- Read/change visibility…

```
>>> lyr.visible
True
>>> lyr.visible = FALSE
```

towns
towns

- Read/change transparency…

```
>>> lyr.transparency
0
>>> lyr.transparency = 50
```

- Turn labels off/on…

```
>>> lyr.showLabels = False
>>> lyr.showLabels = True
```

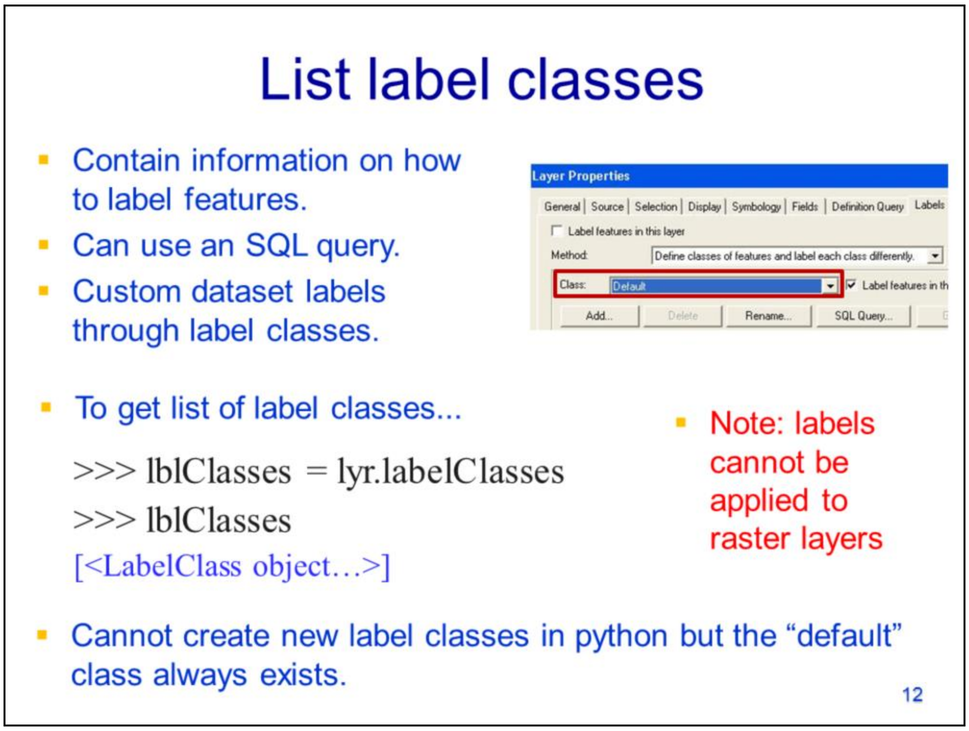
Mansfield

11

The layer object's **visible** property can be set to True or False to turn the layer on or off in ArcMap. In this example, the layer is turned off by setting the visible property to False.

The **transparency** property of the layer can be set from 0 to 100 with 0 being opaque and 100 being completely transparent.

The **showLabels** property turns the layer's labels on (True) or off (False).



List label classes

- Contain information on how to label features.
- Can use an SQL query.
- Custom dataset labels through label classes.

Layer Properties
General | Source | Selection | Display | Symbology | Fields | Definition Query | Labels
Add... | Delete | Rename... | SQL Query...

☐ Label features in this layer

Method: Define classes of features and label each class differently.

Class: Default ☑ Label features in th

- To get list of label classes...

```
>>> lblClasses = lyr.labelClasses
>>> lblClasses
[<LabelClass object…>]
```

- Note: labels cannot be applied to raster layers

- Cannot create new label classes in python but the "default" class always exists.

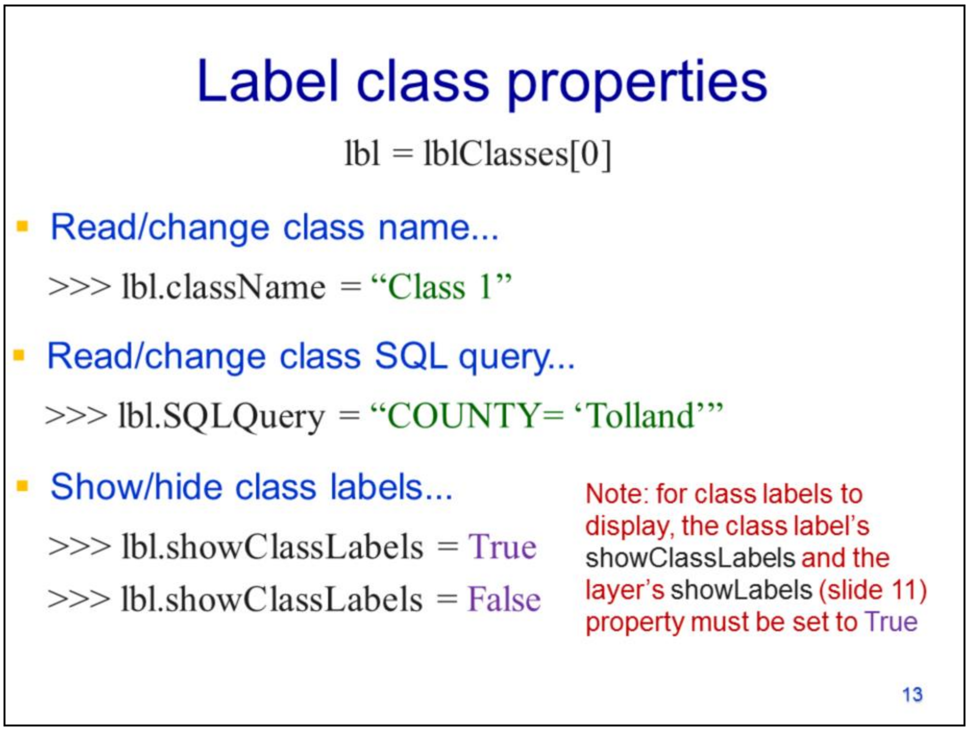
12

The label class allows you to customize how feature labels will be displayed - it is equivalent to accessing the layer properties page in ArcMap.

Use the layer object's **labelClasses** property to get a list of label classes that correspond to the layer.

Note that labels cannot be applied to raster datasets.

Arcpy can only work with existing label classes. Layers always have a "default" label class that can be accessed through arcpy.



Label class properties

lbl = lblClasses[0]

- Read/change class name...

```
>>> lbl.className = "Class 1"
```

- Read/change class SQL query...

```
>>> lbl.SQLQuery = "COUNTY= 'Tolland'"
```

- Show/hide class labels...

```
>>> lbl.showClassLabels = True
>>> lbl.showClassLabels = False
```

Note: for class labels to display, the class label's showClassLabels and the layer's showLabels (slide 11) property must be set to True

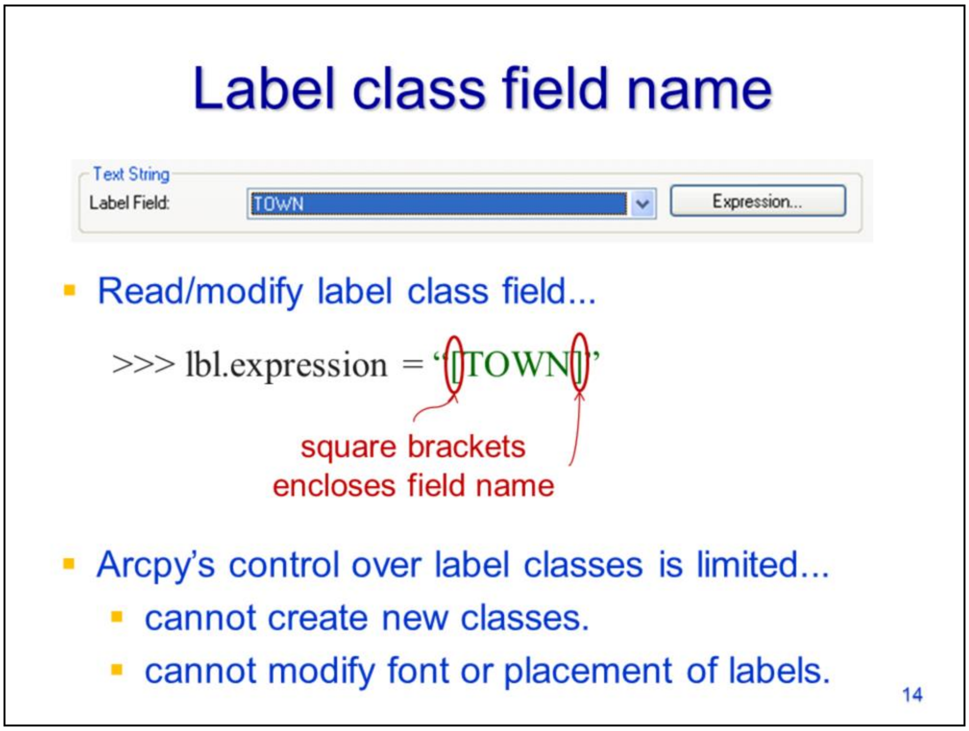
13

The name of a label class can be read or changed by accessing the **className** property.

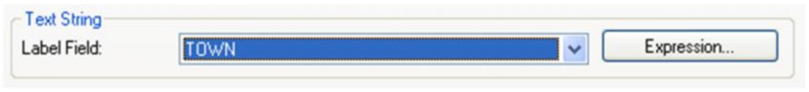
The **SQLQuery** property allows an expression to be used to apply label class properties only to features that satisfy the expression.

The **showClassLabels** property must be True in order for labels to be displayed for the label class.

Note that the layer's **showLabels** property must also be True for labels to be displayed.



Label class field name



- Read/modify label class field...

```
>>> lbl.expression = '[TOWN]'
```

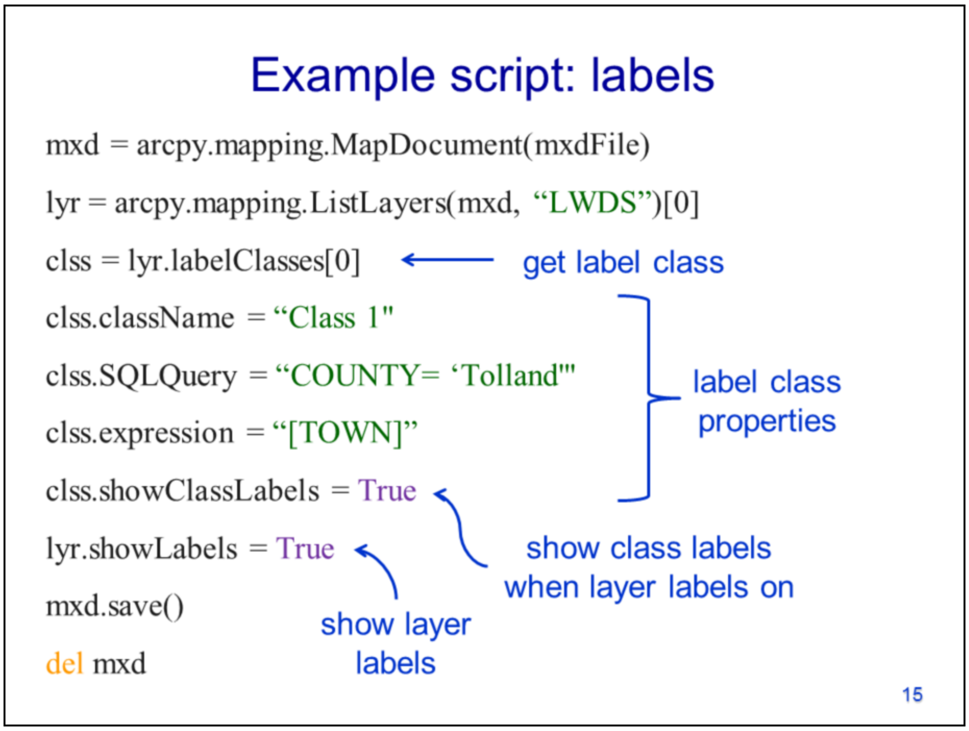
- Arcpy's control over label classes is limited...
  - cannot create new classes.
  - cannot modify font or placement of labels.

14

The field that will be used to display labels is set using the **expression** property of the label. In this example, the values from the "TOWN" field will be used for the labels.

Note that the field name should be enclosed in square brackets.

Arcpy is limited in what it can do with labels. It cannot create new label classes and it cannot modify font or label placement settings.



Example script: labels

```
mxd = arcpy.mapping.MapDocument(mxdFile)
lyr = arcpy.mapping.ListLayers(mxd, “LWDS”)[0]
clss = lyr.labelClasses[0]
clss.className = “Class 1"
clss.SQLQuery = “COUNTY= ‘Tolland'"
clss.expression = “[TOWN]”
clss.showClassLabels = True
lyr.showLabels = True
mxd.save()
del mxd
```

get label class
label class properties
show class labels when layer labels on
show layer labels
15

In this slide, we’ll see an example of a script that works with labels. The first statement gets the map document object.

The mapping module’s ListLayers method is used to get a list of layers with the name “LWDS”. The layer object is extracted from the list.

The layer object’s **labelClasses** method is used to get a list of label classes for the layer. The label class is extracted from the list.

The label classes’ **className** property is used to set the name of the label class to “Class 1”.

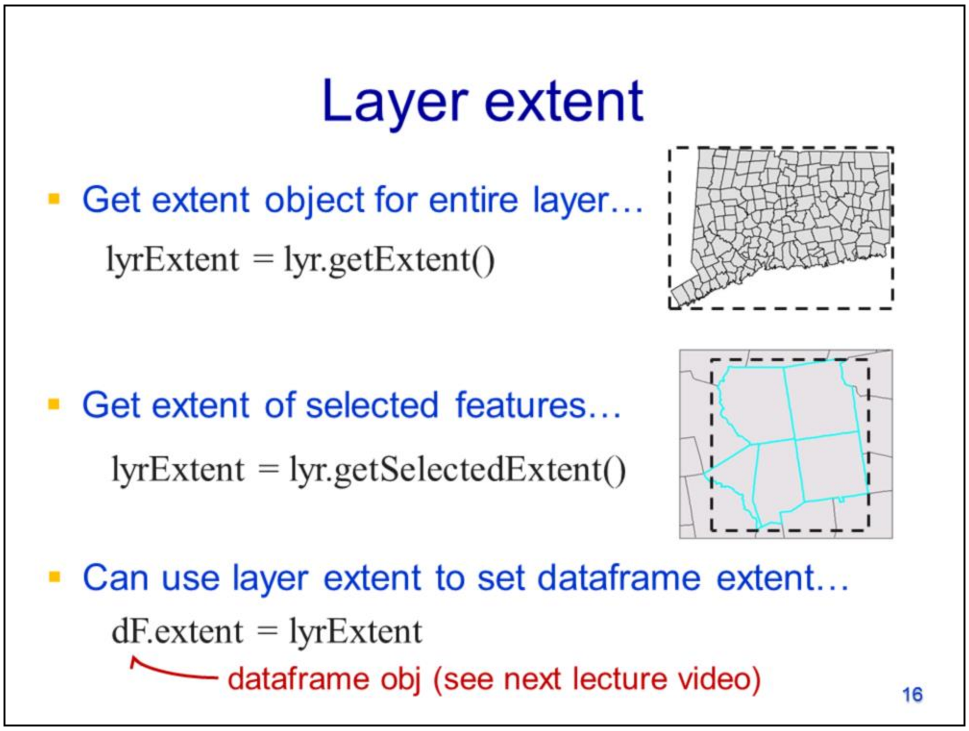
The label classes’ **SQLQuery** property is used to set an expression so that labels will only be displayed for features that satisfy the expression.

The label classes’ **expression** property was used to set the label field name to “TOWN”. Values from this field will be used for the labels.

The label classes’ **showClassLabels** property must be set to True in order for labels to be displayed.

The layer’s **showLabels** property must also be set to True in order to display labels.

The mxd object is saved and the object is deleted to unlock the file.



Layer extent

- Get extent object for entire layer…

```
lyrExtent = lyr.getExtent()
```

- Get extent of selected features…

```
lyrExtent = lyr.getSelectedExtent()
```

- Can use layer extent to set dataframe extent…

```
dF.extent = lyrExtent
```

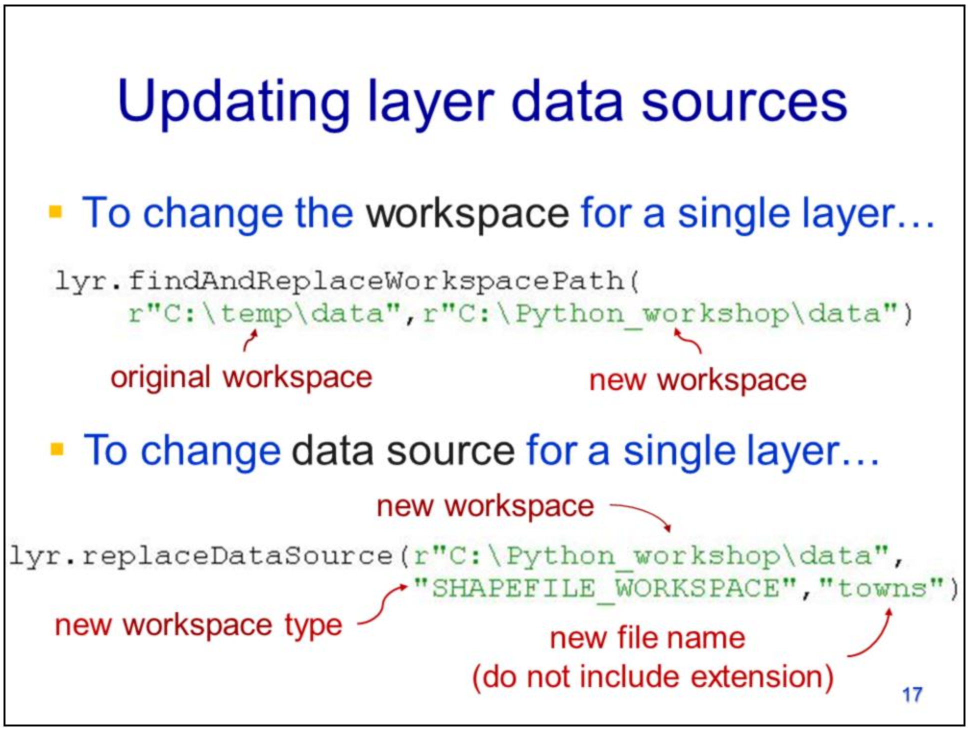
dataframe obj (see next lecture video)

16

The layer’s **getExtent** method will get an extent object that corresponds to the entire layer.

The layer’s **getSelectedExtent** method will get the extent object that corresponds to any selected features in the layer.

The extent object can be used to set the extent of the **dataFrame**. We’ll discuss how to access the dataFrame in the next lecture video.



Updating layer data sources

- To change the workspace for a single layer…

```
lyr.findAndReplaceWorkspacePath(
    r"C:\temp\data",r"C:\Python_workshop\data")
```

original workspace
new workspace

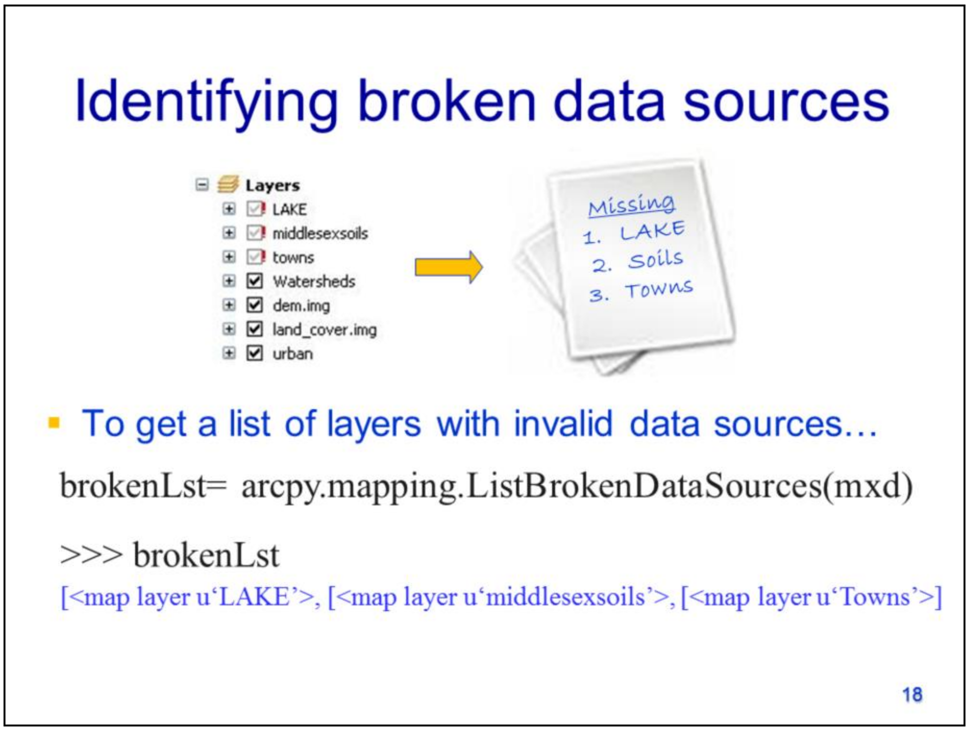
- To change data source for a single layer…

```
lyr.replaceDataSource(r"C:\Python_workshop\data",
                      "SHAPEFILE_WORKSPACE","towns")
```

new workspace
new workspace type
new file name
(do not include extension)
17

The workspace for a layer can be updated by using the layer object's **findAndReplaceWorkspacePath** method.

The **data source** can be changed for a layer by using the layer object's **replaceDataSource** method. Note that the extension should not be included in the basename of the new dataset.



Identifying broken data sources

Layers
LAKE
middlesexsoils
towns
Watersheds
dem.img
land_cover.img
urban
Missing
1. LAKE
2. Soils
3. Towns

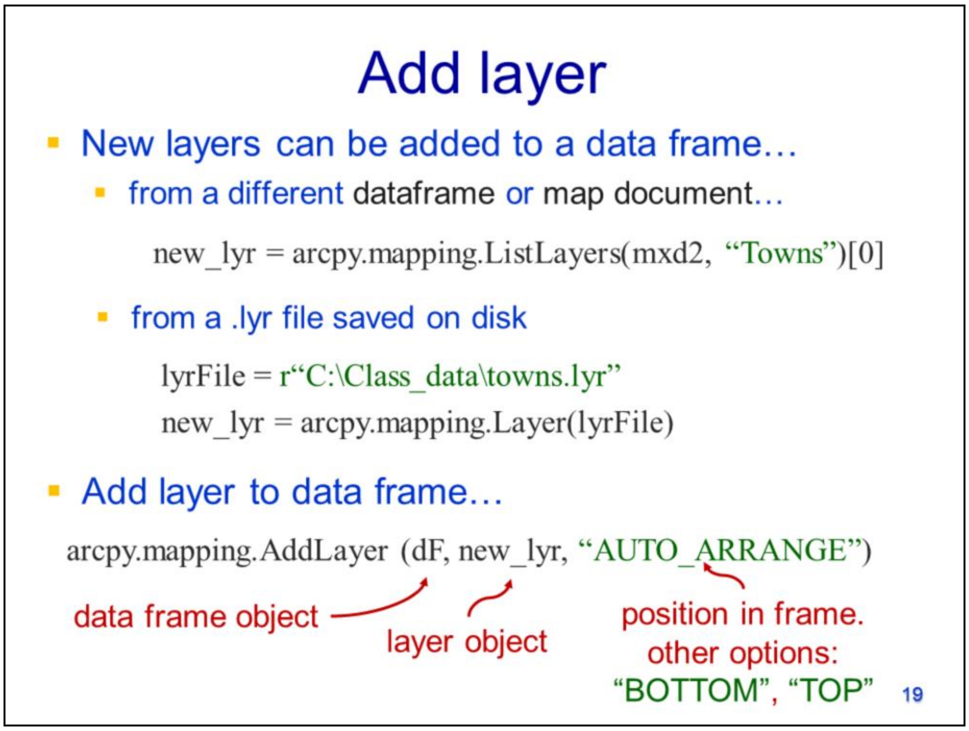
- To get a list of layers with invalid data sources…

```
brokenLst= arcpy.mapping.ListBrokenDataSources(mxd)

>>> brokenLst
[<map layer u'LAKE'>, [<map layer u'middlesexsoils'>, [<map layer u'Towns'>]
```

18

A list of “broken” data sources can be retrieved for a map document object by using the **ListBrokenDataSources** method of the mapping module. This method returns a list of layer objects for which the source dataset cannot be found.



Add layer

- New layers can be added to a data frame…
  - from a different dataframe or map document…

```
new_lyr = arcpy.mapping.ListLayers(mxd2, “Towns”)[0]
```

  - from a .lyr file saved on disk

```
lyrFile = r“C:\Class_data\towns.lyr”
new_lyr = arcpy.mapping.Layer(lyrFile)
```

- Add layer to data frame…

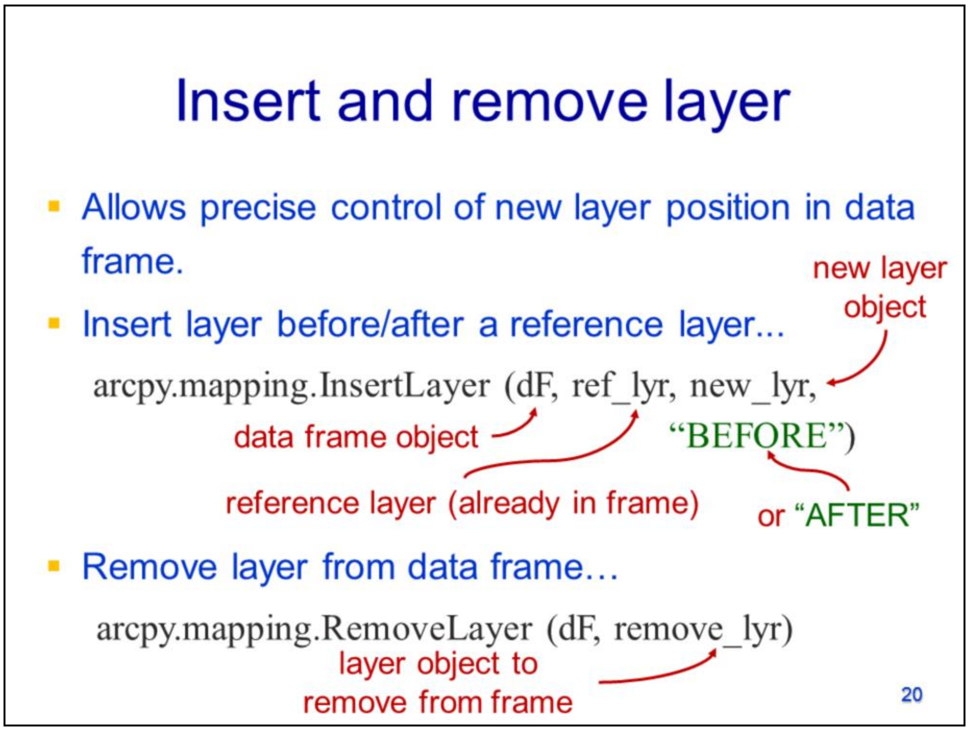
```
arcpy.mapping.AddLayer (dF, new_lyr, “AUTO_ARRANGE”)
```

data frame object
layer object
position in frame.
other options:
“BOTTOM”, “TOP”
19

New layers can be added to a data frame. The layer object, for the new layer can be obtained from a different data frame or a different map document.

The layer object may also be created from a **.lyr** file that has been saved on disk. The mapping module’s Layer method can be used to get a layer object for a saved layer file,

The mapping module’s **AddLayer** method will add the layer object to the specified data frame. The general position of the layer in the data frame can be specified.



Insert and remove layer

- Allows precise control of new layer position in data frame.
- Insert layer before/after a reference layer...

```
arcpy.mapping.InsertLayer (dF, ref_lyr, new_lyr,
                                  "BEFORE")
```

new layer object
data frame object
reference layer (already in frame)
or "AFTER"

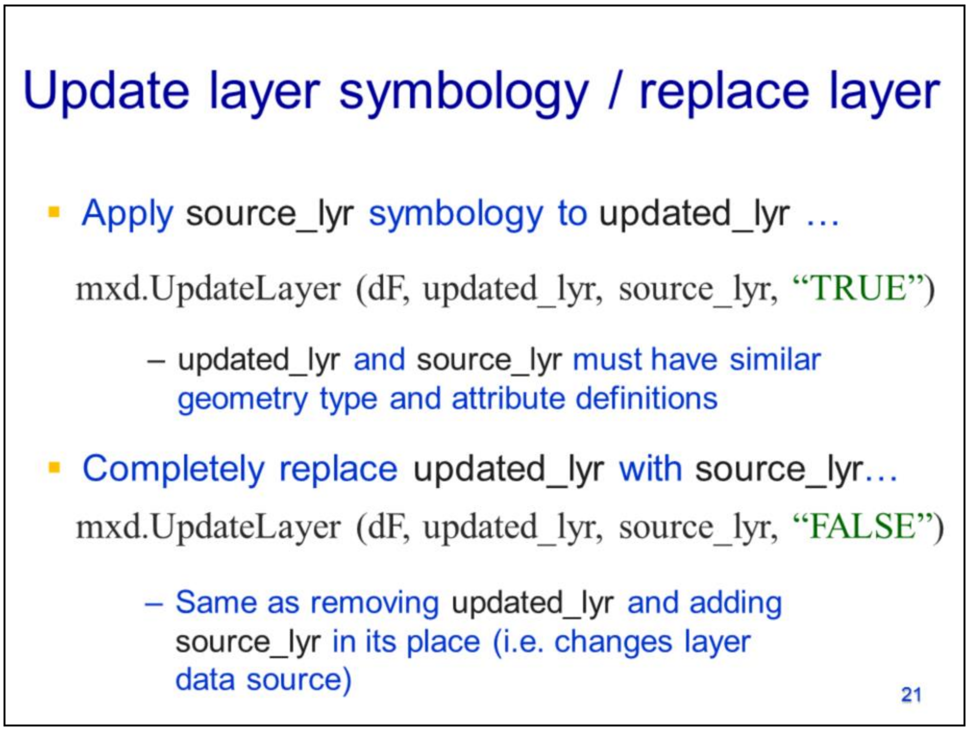
- Remove layer from data frame…

```
arcpy.mapping.RemoveLayer (dF, remove_lyr)
```

layer object to remove from frame
20

The **InsertLayer** method allows precise control over the position of the new layer in the data frame. The position is specified relative to a reference layer that already exists in the data frame. The reference layer is specified in the form of a layer object.

A layer can be removed from the data frame using the **RemoveLayer** method.



Update layer symbology / replace layer

- Apply source_lyr symbology to updated_lyr …

  mxd.UpdateLayer (dF, updated_lyr, source_lyr, “TRUE”)

  – updated_lyr and source_lyr must have similar geometry type and attribute definitions

- Completely replace updated_lyr with source_lyr…

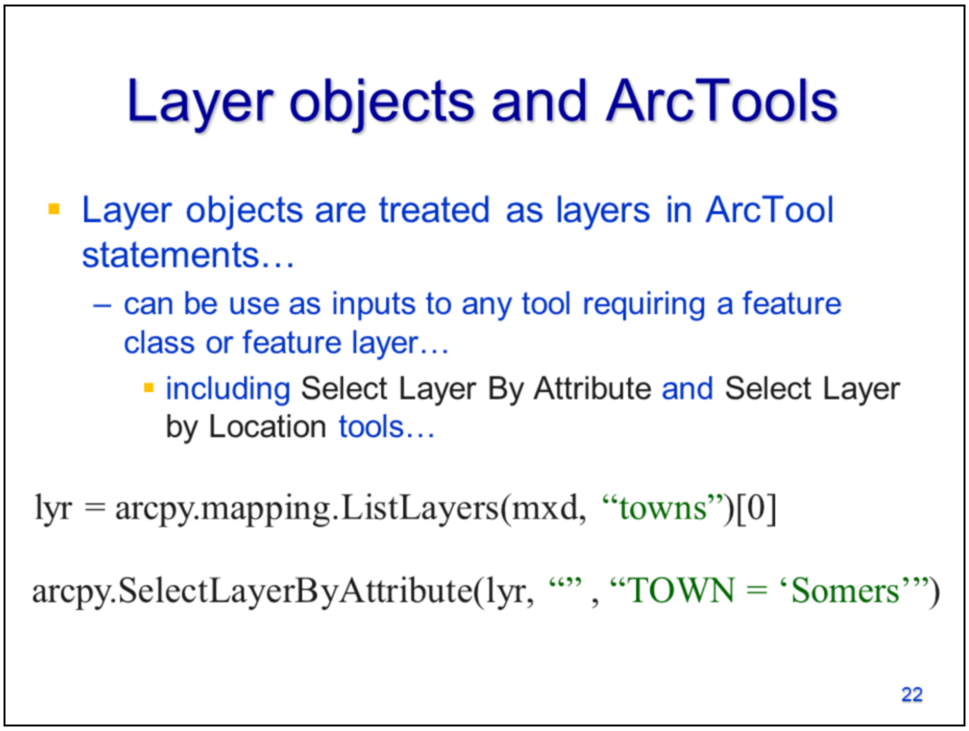
  mxd.UpdateLayer (dF, updated_lyr, source_lyr, “FALSE”)

  – Same as removing updated_lyr and adding source_lyr in its place (i.e. changes layer data source)

21

The **UpdateLayer** method allows the symbology of one layer to be applied to the another layer. The updated_lyr and the source_lyr are both specified as layer objects and must have the same geometry type and attributes. The “TRUE” value in the statement indicates that the updated_lyr’s data source will not be affected.

When a “FALSE” value is used with the **UpdateLayer** tool, the updated_lyr’s data source and symbology will be replaced by the source_lyr. This is equivalent to removing the updated_lyr and adding the source_lyr in its place.



Layer objects and ArcTools

- Layer objects are treated as layers in ArcTool statements…
  - can be use as inputs to any tool requiring a feature class or feature layer…
    - including Select Layer By Attribute and Select Layer by Location tools…

```
lyr = arcpy.mapping.ListLayers(mxd, “towns”)[0]

arcpy.SelectLayerByAttribute(lyr, “” , “TOWN = ‘Somers’”)
```

22

Layer objects can be used as inputs to any ArcTool that accepts a layer as an input – this includes the Select Layer By Attribute and Select Layer By Location tools which only accept layer inputs.

This example shows a layer object being used as an input to the SelectLayerByAttribute tool.