



King AbdulAziz University

Faculty of Environmental Design
Geomatics Department

**GIS Programming
(Level 1)
GEOM 224**

Week 7

**Spatial Data Exploration
with python**

K.Walid



Content

1. Introduction
2. Working with Arcpy. General Operations
 - a. Introduction
 - b. Importing Modules
 - c. Setting up the environment
 - d. Exploring Spatial Data Sets
 - List Datasets
 - List Features Classes
 - List Rasters
 - e. Creating Data Sets
3. Working with Map Document
 - a. Accessing Map Document
 - b. Accessing Data Frames
 - c. Accessing Layers
4. Working with Data Sets
 - a. Describing the Data Set
 - b. Accessing and working with fields



Introduction

- Python is the scripting language for ArcGIS, it's free, widely used and it's a cross platform language
- You can use python with ArcGIS to:
 - ✓ Automate repetitive tasks
 - ✓ Develop workflows that leverage hundreds of tools and functions
 - ✓ Customize Desktop applications
 - ✓ Extend the capabilities of ArcGIS



Working with Arcpy: Introduction

Arcpy is:

- A site package included with ArcGIS that enables interaction with Python
- An access point to the geoprocessing tools
- A package of functions, classes and modules
 - ✓ functions that perform useful tasks and enable automation
(ListFeatureClasses, Describe, SearchCursor)
 - ✓ Classes that can be used to create complex objects
(SpatialReference, Geometry, FieldMap)
 - ✓ Modules that provide specialized functionality
(Mapping, Spatial Analyst, Data Access)



Working with Arcpy: Importing modules

Statement	###Comment
>>>import <u>arcpy</u>	import all arcpy functions and classes
>>>import <u>arcpy.da</u>	import data access module: functions and subclasses to control the edit session and edit operations
>>>import <u>arcpy.sa</u>	import raster module: functions and subclasses, to analyze raster data with the functionality provided by the ArcGIS Spatial Analyst extension
>>>import <u>arcpy.mapping</u>	Import the mapping module :function and sub classes, to manipulate the content of an existing map documents (.mxd) and layer files (.lyr), and to automate exporting and printing.
>>>import <u>arcpy.na</u>	Look by your self to all the existing arcpy modules....
>>>import <u>arcpy.stats</u>	
.....	





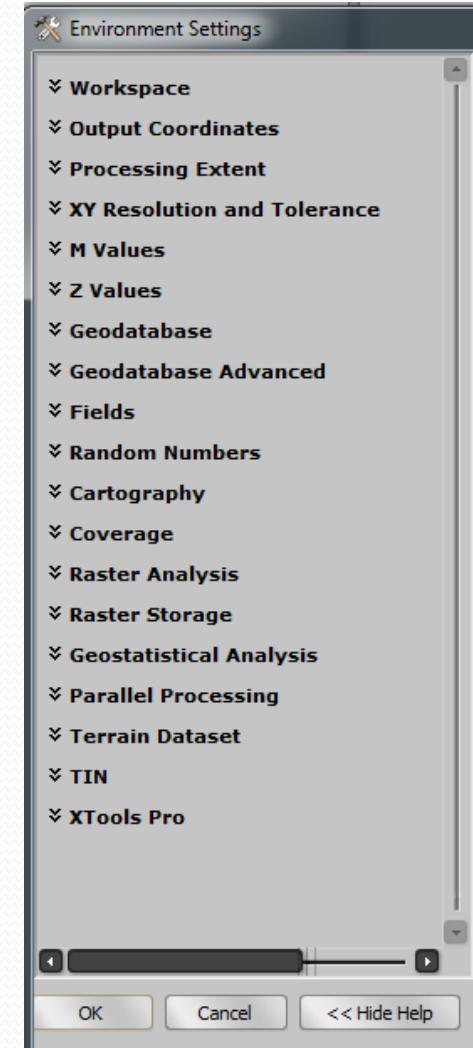
Working with Arcpy: setting up the environment

The geoprocessing environment settings are the default system settings, used by each GP tool.

Literally these parameters are the properties of the

Class arcpy. env

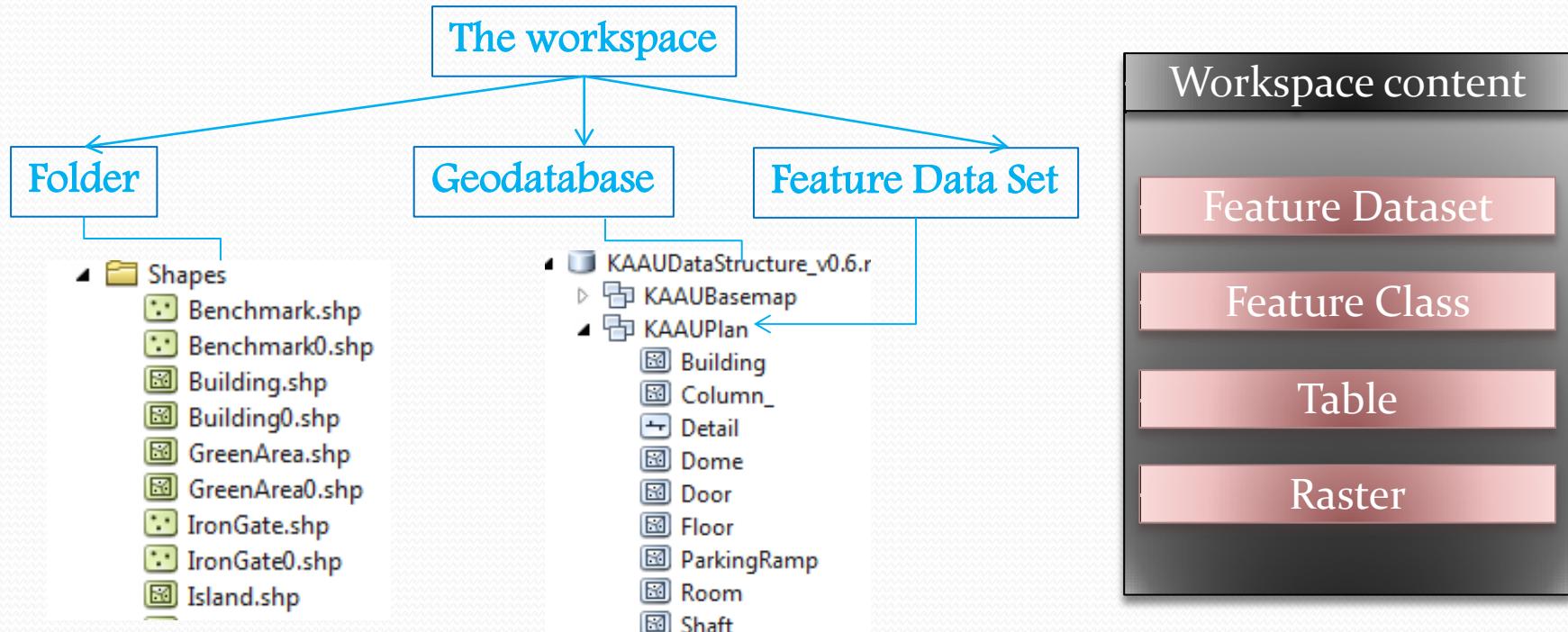
Statement	###Comment
workspace	The default location for geoprocessing tool input and output.
outputCoordinateSystem	Specify the output coordinate system
extent	Specify the extent of the geoprocessing tool
cell size	Define the cell size in raster analysis
.....



Working with Arcpy: setting up the environment

```
arcpy.env.workspace = path #string variable path defining
#physical location to inputs and outputs
```

```
import arcpy #import the arcpy module
arcpy.env.workspace = "C:/data/base.gdb" # Set the workspace environment to local file geodatabase
```





Explore spatial data in the workspace

To explore spatial data in the workspace use the following arcpy functions.

Functions	###Comment
ListDatasets()	Returns a List of all of the datasets in a workspace
ListFeatureClasses() ,	Returns a List of all the feature classes in the workspace
ListFiles()	Returns a List of files in the current workspace based on a query string
ListRasters()	Returns a List of the rasters in the workspace
ListTables()	Returns a List of the tables in the workspace

- ✓ All functions have a **wild_card** parameter to filter filenames
- ✓ All functions have Additional parameters according to the type of data: type of geometry, table format, image size, etc.



ListDatasets()

Syntax

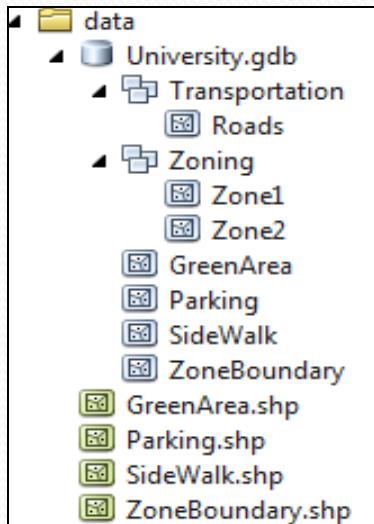
```
ListDatasets ({wild_card}, {feature_type})
```

Parameter	###Comment	Data Type
wild_card	The wild_card limits the results returned. If no wild_card is specified, all values are returned.	string
feature_types	limit the results returned by the wildcard argument. Valid dataset types are: <ul style="list-style-type: none">• Coverage —Only coverages.• Feature —Coverage or geodatabase dataset, depending on the workspace.• GeometricNetwork —Only geometric network datasets.• Mosaic —Only mosaic datasets.• Network —Only network datasets.• ParcelFabric —Only parcel fabric datasets.• Raster —Only raster datasets.• RasterCatalog —Only raster catalog datasets.• Schematic —Only schematic datasets.• Tin —Only TIN datasets.• Topology —Only topology datasets.• All —All datasets in the workspace. This is the default value.	string



ListDatasets()

```
import arcpy #import the arcpy module
arcpy.env.workspace = "c:/University.gdb" # Set the workspace environment to the
#geodatabase University
mylstdatasets = arcpy.ListDatasets("Z*", "Feature") # create a variable list of features datasets
#starting with letter Z
for dataset in datasets:
    print(dataset) # Print to the Interactive window all the feature datasets in the list
```



```
>>>
Zoning
>>>
```



ListFeatureClasses()

Syntax

```
ListFeatureClasses ({wild_card}, {feature_type}, {feature_dataset})
```

Parameter	###Comment	Data Type
wild_card	The wild_card limits the results returned. If no wild_card is specified, all values are returned.	string
feature_types	<p>The feature type to limit the results returned by the wild_card argument. Valid feature types are:</p> <ul style="list-style-type: none">•Annotation —Only annotation feature classes are returned•Dimension —Only dimension feature classes are returned.•Edge —Only edge feature classes are returned.•Junction —Only junction feature classes are returned.•Label — Only label feature classes are returned.•Line —Only line feature classes are returned.•Multipatch —Only multipatch feature classes are returned.•Node —Only node feature classes are returned.•Point —Only point feature classes are returned.•Polygon —Only polygon feature classes are returned.•Polyline —Only line feature classes are returned.•Region —Only region feature classes are returned.•Route —Only route feature classes are returned.•Tic —Only tic feature classes are returned.•All — All datasets in the workspace. This is the default value.	string
feature_dataset	Limits the feature classes returned to the feature dataset, if specified. If blank, only stand-alone feature classes will be returned in the workspace	string



ListFeatureClasses()

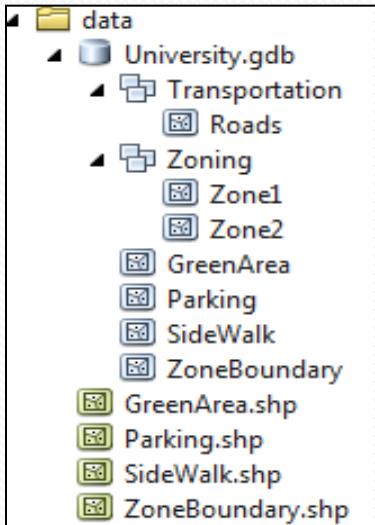
```
import arcpy #import the arcpy module

arcpy.env.workspace = "c:/data" # Set the workspace environment to the folder C:\data

mylstFeatureclasses = arcpy.ListFeatureClasses() # create a variable list of features classes

for fc in mylstFeatureclasses :

    print(fc) # Print to the Interactive window all the feature datasets in the list
```



```
>>>
GreenArea.shp
Parking.shp
SideWalk.shp
ZoneBoundary.shp
>>>
```



ListRasters()

Syntax

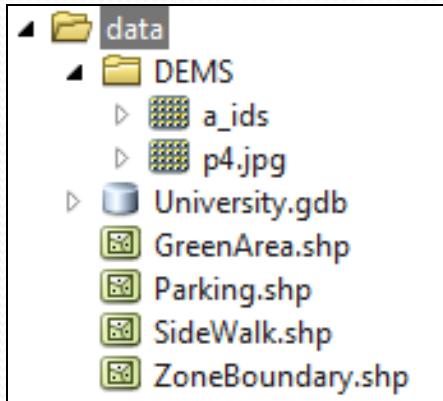
ListRasters ({wild_card}, {raster_type})

Parameter	###Comment	Data Type
wild_card	The wild_card limits the results returned. If no wild_card is specified, all values are returned.	string
raster_type	<p>The raster type to limit the results returned by the wild card argument. Valid raster types are:</p> <p>BMP — Bitmap graphic raster dataset format. GIF — Graphic Interchange Format for raster datasets. IMG — ERDAS IMAGINE raster data format. JP2 — JPEG 2000 raster dataset format. JPG — Joint Photographic Experts Group raster dataset format. PNG — Portable Network Graphics raster dataset format. TIF — Tagged Image File for raster datasets. GRID — Grid data format. All — All supported raster types are returned. This is the default.</p>	string



ListRasters()

```
import arcpy #import the arcpy module
arcpy.env.workspace = "c:/data/DEMS" #Set the workspace environment to the folder
#C:/data/DEMS
mylstRasters = arcpy.ListRasters("*", "GRID") # Get and print a list of GRIDs from the workspace
for raster in rasters:
    print(raster)
```



```
>>>
a_ids
>>>
```



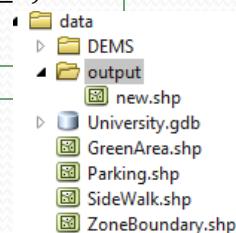
Creating Datasets

To create a geographic dataset you can use the following functions:

- ✓ `CreateFeatureClass_management()`
- ✓ `CreateFeatureDataset_management()`
- ✓ `CreateTable_management()`
- ✓ ...

Workspace Type	<code>CreateFeatureClass</code>	<code>CreateFeatureDataset</code>	<code>CreateTable</code>
Folder	Shapefile(3 files)	-	DBF file
Geodatabase	Feature class	Feature Dataset	Geodatabase table

```
CreateFeatureclass_management (out_path, out_name,{geometry_type},{template},{has_m},{has_z},  
{spatial_reference})
```



```
import arcpy  
from arcpy import env  
env.workspace = "C:/data"  
arcpy.CreateFeatureclass_management("C:/data/output", "new.shp", "POLYGON", "", "DISABLED", "DISABLED", "Parking.shp")
```



Accessing Map Document

A reference to a map document (.mxd) is essential for most map scripting operations

Syntax **mapObj=MapDocument (mxd_path)**

mxd_path:

A string that includes the full path and file name of an existing map document (.mxd) or a string that contains the keyword CURRENT (if you are scripting in ArcGIS):

```
>>>varMxd1= arcpy.mapping.MapDocument(r'C:\data\University.mxd')  
>>>varMxd2= arcpy.mapping.MapDocument('CURRENT')
```



Accessing Map Document

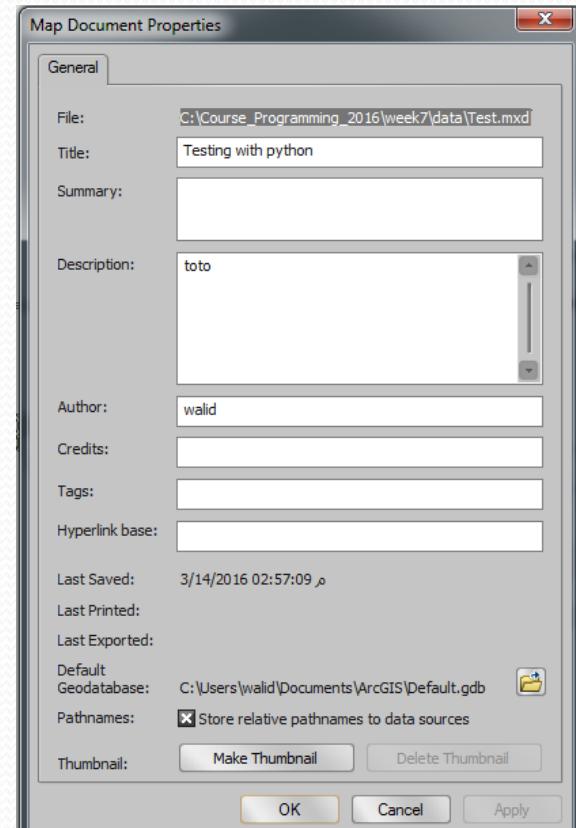
MapDocument properties and methods:

Prperty	###Comment	Data Type
title	Provides the ability to either get or set the map document's title information	String
relativePaths (Read and Write)	Provides the ability to control if a map document stores relative paths to the data sources. A value of True sets relative paths; a value of False sets full paths to the data sources.	Boolean
filePath (Read Only)	Returns a string value that reports the fully qualified map document path and file name	String
author (Read and Write)	Provides the ability to either get or set the map document's author information.	String
.....
Method	###Comment	
Save()	This performs the same operation as File > Save in ArcMap	
SaveACopy()	This performs the same operation as File > Save as...in ArcMap <code>saveACopy (file_name, {version})</code>	
.....		

Accessing Map Document

```
import arcpy
varmxd=arcpy.mapping.MapDocument("C:\\\\data\\\\test.mxd")
print varmxd.filePath
print varmxd.activeView
print varmxd.author
print varmxd.pageSize
print varmxd.title
print varmxd.relativePaths
varmxd1.saveACopy("C:\\\\data\\\\test.mxd","10.3")
```

```
>>>
C:\\data\\test.mxd
Layers
walid
PageSize(width=21.005842011684024, height=29.688426043518756)
Testing with python
True
>>>
```





Accessing DataFrame: ListDataFrames()

The function `ListDataFrames()` returns a list of data frames in a referenced Map Document.

```
Import arcpy
```

```
#import the arcpy module
```

```
varMxd1=arcpy.mapping.MapDocument(r'C:\data\University.mxd')
```

```
#create an object 'varMxd1'map document for the mxd file 'University.mxd'
```

```
dfList=arcpy.mapping.ListDataFrames(varMxd1)
```

```
#Create a variable list 'dfList' containing all dataframes in this map document
```

```
For df in dfList:
```

```
    print df.name, df.spatialReferenc.name
```

```
    #print all data frames names and spatial references
```



```
>>>
```

```
DataFrame1 Ain_el_Abd_UTM_Zone_37N
```

```
DataFrame2 GCS_Ain_el_Abd_1970
```

```
>>>
```



Accessing Layers: ListLayers()

The function ListLayers() returns a list of layers in a referenced Map Document.

```
Import arcpy
```

```
#import the arcpy module
```

```
varMxd1=arcpy.mapping.MapDocument(r'C:\data\University.mxd')
```

```
#create an object 'varMxd1'map document for the mxd file 'University.mxd'
```

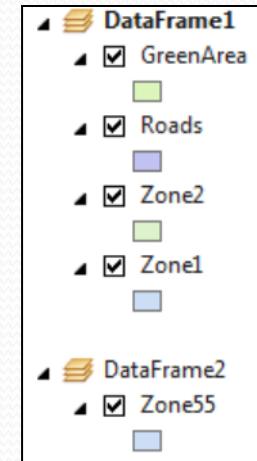
```
lyList=arcpy.mapping.ListLayers(varMxd1)
```

```
#Create a variable list 'lyList' containing all Layers in this map document
```

```
For ly in lyList:
```

```
    print ly.name,
```

```
#print all layer's names
```



```
>>>  
GreenArea  
Roads  
Zone2  
Zone1  
Zone55  
>>>
```



Describing Spatial Dataset: Describe()

The `Describe()` function determines the properties of a dataset (type, fields, indexes, etc.) in order to process it in an appropriate way. `Describe()` returns an object of class `Describe`. All these objects have common properties (`baseName`, `catalogPath`, `children`, `dataType`, `extension`, `file`, `name`, `path`, etc.) and the properties corresponding to their data type (GDB table, a file, etc.).

Import arcpy

```
varMxd1=arcpy.mapping.MapDocument(r'C:\data\University.mxd')
```

```
lyList=arcpy.mapping.ListLayers(varMxd1)
```

For ly in lyList:

```
desc=arcpy.Describe(ly)
```

```
#Create an object describe 'desc' for each layer
```

```
print desc.shapetype
```

```
#print the shape type for each object desc
```



Accessing fields

Fields in the table store the geometry and attribute information for the features

The function `ListFields()` returns a list of fields names of a layer

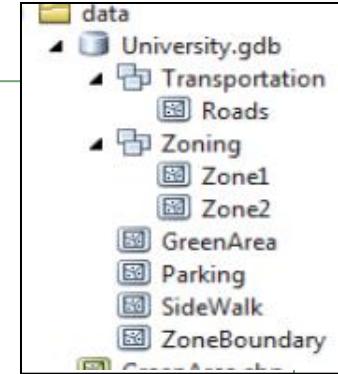
```
import arcpy

# Create an object Feature class to the layer 'Gates' in 'KAU.gdb'
featureClass = "C:\\\\Data\\\\University.gdb\\\\Parking"

# Create a list of field of the 'Parking' Layer
fieldList = arcpy.ListFields(featureClass)

# Loop through each field in the list and print the name and the type
for fld in fieldList:
    print fld.name, fld.type
```

```
>>>
OBJECTID OID
Shape Geometry
DataSource String
CreationUs String
DateCreate Date
DateModifi Date
LastUser String
....
>>>
```



Field Name
OBJECTID
Shape
DataSource
CreationUs
DateCreate
DateModifi
LastUser
ParkingNo
ParkingNam
ParkingOwn
Subtype
Capacity
BaseElevat



Working with fields

To manipulate fields and fields values use the geoprocessing functions

Function	###Comment
AddField_management()	AddField_management (in_table,field_name, field_type, {field_precision}, {field_scale}, {field_length}, {field_alias}, {field_is_nullable}, {field_is_required}, {field_domain})
DeleteField_management()	DeleteField_management (in_table, drop_field)
CalculateField_management()	CalculateField_management (in_table, field, expression, {expression_type}, {code_block})



Working with fields

```
import arcpy
Arcpy.env.workspace = "C:/KAU.gdb"
arcpy.AddField_management("Faculty", "st_numb", "INTEGER", 9, "", "", "stNB", "NULLABLE", "REQUIRED")
```

```
import arcpy
arcpy.env.workspace = "C:/data"
arcpy.CopyFeatures_management("Roads.shp", "C:/output/Roads_copy.shp")
arcpy.DeleteField_management("C:/output/Roads_copy.shp", ["Road_NAME", "LABEL", "CLASS"])
```

```
# Add fields
arcpy.AddField_management("parcels", "xCentroid","DOUBLE",18, 11)
arcpy.AddField_management("parcels", "yCentroid", "DOUBLE", 18, 11)
# Calculate centroid
arcpy.CalculateField_management(("parcels", "xCentroid", "!SHAPE.CENTROID.X!","PYTHON_9.3"))
arcpy.CalculateField_management(inFeatures, "yCentroid", "!SHAPE.CENTROID.Y!","PYTHON_9.3")
```