# python

## GIS libraries

# Introduction

# The main tasks that any geospatial software can perform are next:

- Data access. Both local and remote data (WMS, GM, OSM).
- Selections and queries
- Manipulation of vector and raster files (creation, edition, etc.).
Spatial analysis operations. Geoprocessing (DEM analysis, reprojection, filters, optimal shortest path, etc.).
- Cartographic production.
- GUI integration (ArcToolBox, PyQt4, QGIS plugins, etc.).

# Python has a lot of libraries that can perform this type of operations. They are mainly used by Open Source software, although a lot of commercial software use them too.

# Introduction

# Some of the Geospatial software that use this libraries are:

- QGIS
- OPENEV
- GRASS
- UDIG
- GVSIG
- OPENJUMP
- GOOGLE EARTH
- NASA WORLD WIND
- ARCGIS

# In the same way, spatial databases also take advantage of them:

- POSTGIS (POSTGRESQL)
- ORACLE SPATIAL
- ARCSDE
- MICROSOFT SQL SERVER
- MYSQL
- SPATIAL LITE

# Introduction

# Some examples of these libraries:

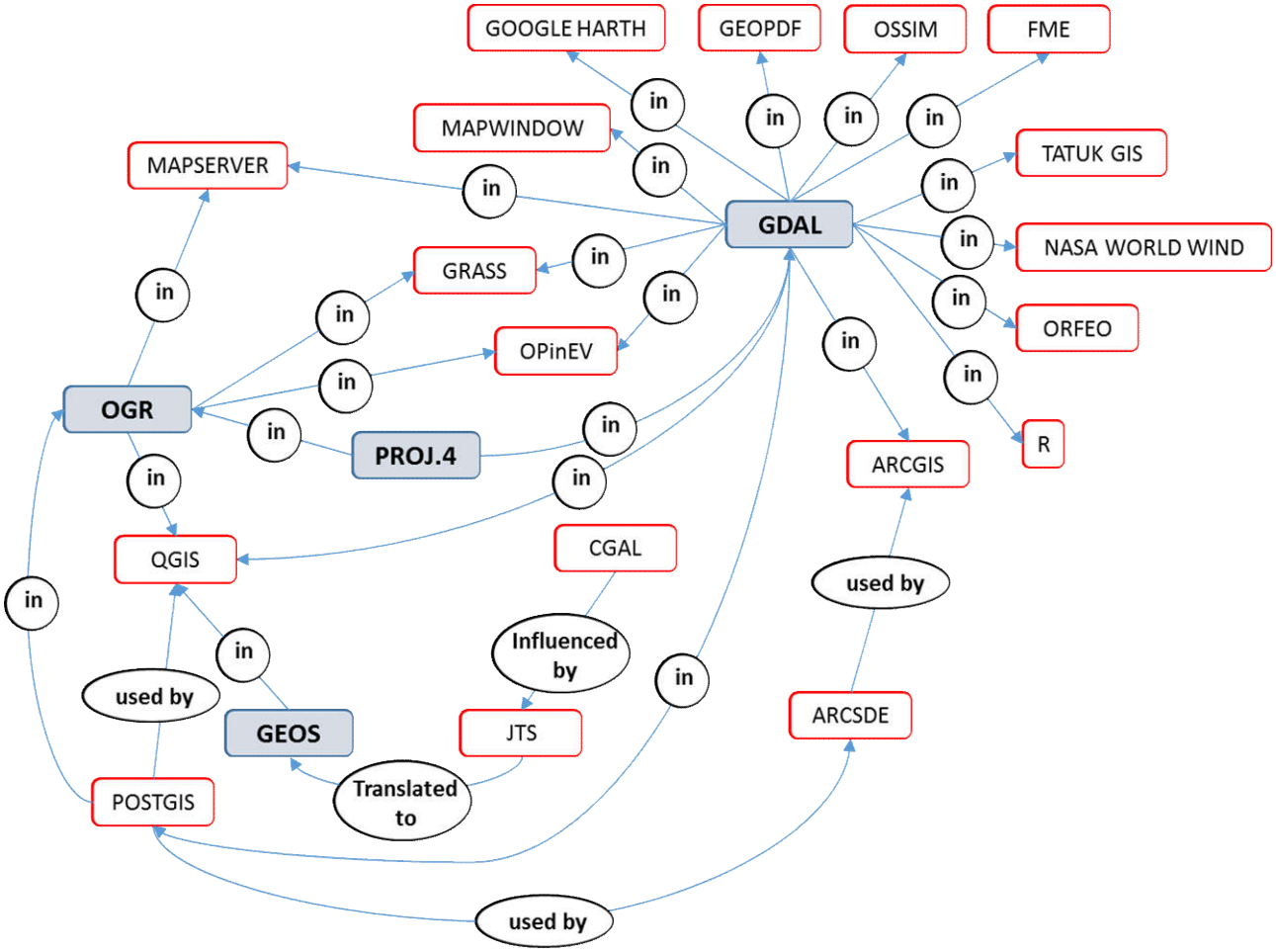
- Data manipulation:
  - GDAL (GeoSpatial Data Abstraction Library). Used for reading and writing raster data. Used in over 80 pieces of software.
  - OGR. Used for vector data (SHP, KML, WTK, CAD, etc.). Over 70 file formats.

- Computational geometry:
  - PROJ4. Specialized library in projections (United States Geological Service).
  - CGAL (Computational Geometry Algorithms Library).
  - JTS (Java Topology Suite).
  - GEOS (Geometry Engine Open Source). Higher impact than JTS. It has python bindings.

- Others:
  - GEOPY. Google Maps geocoding API.
  - SHAPELIB. Shapefile maniputation (also in OGR)
  - LIBLAS. LiDAR data analysis and management.
  - OPENCV. Computer vision library (images, video, stereoscopic vision, camera calibration, object recognition, etc.).
  - **ARCPY**. ArcGIS python library.
  - **PYQGIS**. QGIS python library.

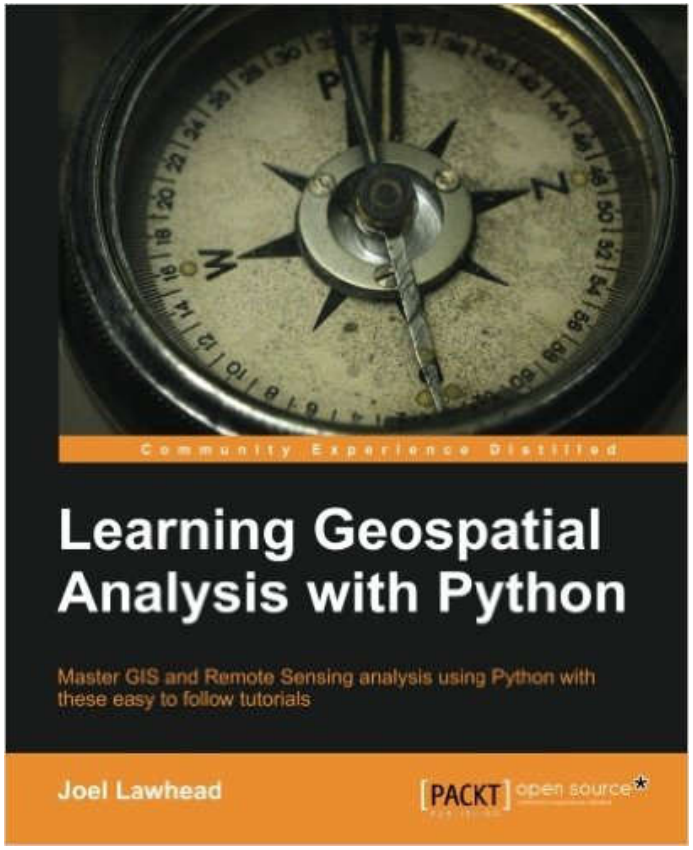
# Introduction

# Software and libraries relationship:



# Introduction

# A good reference book for working with geospatial information using python:



python
ArcGIS

# Arcpy package

# Arcpy

# In order to work with python in ArcGIS, it is necessary to import the **Arcpy** library.

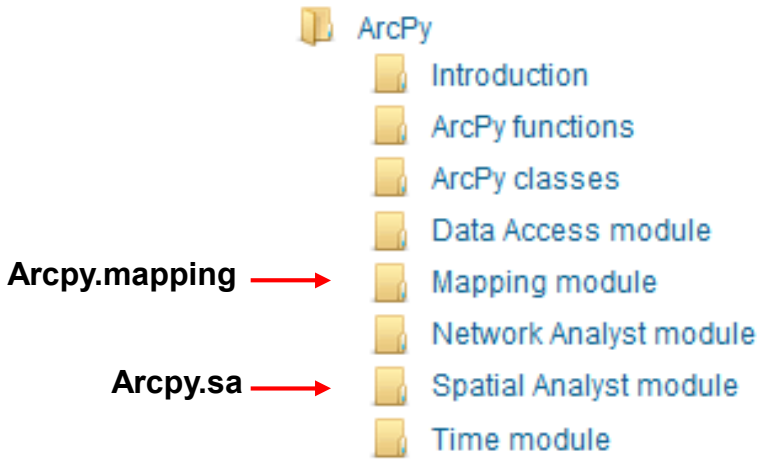
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/000v000000v7000000

# Arcpy provides access to **geoprocessing tools**, as well as functions, classes, and extra modules in order to design both single and complex workflows:

- data description, lists, fields, cursors, geometries, raster, etc.

# some extra modules:

- arcpy.mapping: designed mainly to manipulate the contents of existing map documents (.mxd) and layer files.
- arcpy.sa: spatial analysis module



# Arcpy

## Classes

- Cursor
- Exceptions
- FeatureSets/RecordSets
- Fields
- General
- Geometry
  - Geometry
  - Multipoint
  - Point
  - PointGeometry
  - Polygon
  - Polyline
- Geostatistical Analyst
- Graphing
- Parameter

## Functions

- Cursors
  - InsertCursor
  - SearchCursor
  - UpdateCursor
- Data store
- Describing data
- Environments and settings
- Fields
- Geodatabase administration
- Geometry
- General
- General data functions
- Getting and setting parameters
- Licensing and installation
- Listing data
- Log history
- Messaging and error handling
- Progress dialog
- Publishing
- Raster
- Spatial references and transformations
- Tools and toolboxes

# Arcpy.mapping (what's it used for?)

# Making reports with de project content (datafreme name, layers, reference system, etc.).
# Updating, repparing or replacing data sources.
# Updating symbology with a document closed.
# Searching and replacing text strings of all projects in a folder.
# Saving projects for older versions of ArcMap.
# Updating project metadata.
# Making new data sources in "batch" mode using export commands.
# Publising maps to ArcGIS Server.
# Making maps and atlas in PDF format.

# Arcpy.mapping

## Classes

- DataDrivenPages
- DataFrameTime
- DataFrame
- GraduatedColorsSymbology
- GraduatedSymbolsSymbology
- GraphicElement
- LabelClass
- Layer
- LayerTime
- LegendElement
- MapDocument
- MapsurroundElement
- PDFDocument
- PictureElement
- RasterClassifiedSymbology
- StyleItem
- TableView
- TextElement
- UniqueValuesSymbology

## Functions

### Managing Documents and Layers

- AddLayer
- AddLayerToGroup
- AddTableView
- AnalyzeForMSD
- AnalyzeForSD
- ConvertToMSD
- ConvertWebMapToMapDocument
- CreateGISServerConnectionFile
- CreateMapSDDraft
- DeleteMapService
- InsertLayer
- Layer
- ListBookmarks
- ListBrokenDataSources
- ListDataFrames
- ListLayers
- ListLayoutElements
- ListMapServices
- ListStyleItems
- ListTableViews
- MapDocument
- MoveLayer
- PDFDocumentCreate
- PDFDocumentOpen
- PublishMSDToServer
- RemoveLayer
- RemoveTableView
- TableView
- UpdateLayer

### Exporting and Printing Maps

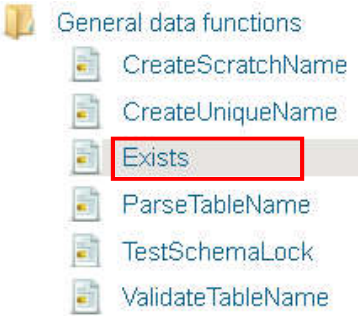
- ExportReport
- ExportToAI
- ExportToBMP
- ExportToEMF
- ExportToEPS
- ExportToGIF
- ExportToJPEG
- ExportToPDF
- ExportToPNG
- ExportToSVG
- ExportToTIFF
- ListPrinterNames
- PrintMap

# Accessing to the project

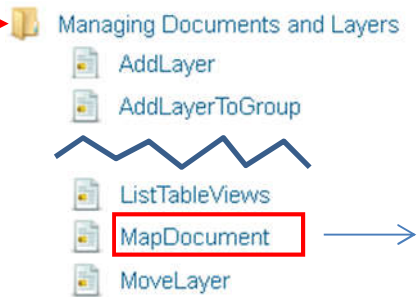
# arcpy.Exists(mxd path): checks if a Project exists
# arcpy.mapping.MapDocument(path): returns the object “Project” and assigns this value to a variable. That way wew can Access to its properties:
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/00s30000000n000000/

- Title (r/w): project title.
- Author (r/w): project author.
- activeView (r/w): active view (either data view or layout view).
- activeDataframe (r): active dataframe

**Arcpy** →




**Arcpy.mapping** →




→ properties

1_access_project.py

# Accessing to the project

```python
#importing modules
import arcpy
import arcpy.mapping as map # renaming module


try:
    #definition of variables
    ruta = r'C:\asignaturas\master\DAS\temario\sesion1\teoria\scripts\castilla_leon.mxd'
    if arcpy.Exists(ruta): #checking if the file exists
        mxd = map.MapDocument(ruta)
    else:
        raise IOError

    # How to use some properties (title, author and summary)
    print "Title: " + mxd.title + " Author:" + mxd.author
    print mxd.summary

    #memory releasing
    del mxd
    # raw_imput: keyboard input. Only makes sense if we run the script on console mode
    raw_input()
except IOError:
    print " One error has occurred: file doesn't exist"
    raw_input()
```

1_access_project.py

# Accessing to the project

```
import arcpy
import arcpy.mapping as map

def printList(lista):
    for l in lista:
        print l

def getName(ruta):
    pass

ruta_mxd = r'C:\asignaturas\master\DAS\temario\sesion1\teoria\scripts\castilla_leon.mxd'
metadatos = []
if arcpy.Exists(ruta_mxd):
    mxd = map.MapDocument(ruta_mxd)
    titulo = mxd.title
    autor = mxd.author
    descrip = mxd.description
    ruta = mxd.filePath
    metadatos.append(titulo)
    metadatos.append(autor)
    metadatos.append(descrip)
    metadatos.append(ruta)
    printList(metadatos)
else:
    print "File doesn't exist"
```

Pay attention on the "file Path" function. change the function "getName" in order to print only the name of the project (without extension)

mxd_metadata.py

# Accessing to the project

# We can run the same script using the python console of ArcMap.
# We need just write the next statements:

```
>>> import os
>>> os.system(script path / parameters')
```

You must write the path without white spaces

# We also can copy the whole code and paste it on the python console

# If a script needs some parameters, put these lines on the ArcMap python console:

```
import sys  # system module
….
ruta = sys.argv[1]  # getting the first parametre
```

2_access_project_python_window.py

# Accessing to the project

```python
#Import modules
import arcpy
import arcpy.mapping as map #change the module name
import sys

try:
    #Definition of variables
    ruta = sys.argv[1] #'Current' doesn't work
    print ruta
    if arcpy.Exists(ruta): #check if path exists
        mxd = map.MapDocument(ruta)
    else:
        raise IOError

    #Using some properties (title, author and summary)
    print "Titulo: " + mxd.title + " Autor:" + mxd.author
    print mxd.summary
    #keyboard input. Only makes sense if we run the script on console mode
    raw_input()
except IOError:
    print "File doesn't exists"
    raw_input() #hace que la ventana de comandos no se cierre para ver el print
```

Python

```
>>> import os
>>> os.system(r'2_access_project_python_window.py E:\DAS\2016-2017\temario_ingles\session1\theory\scripts\test.mxd')
```

C:\Windows\system32\cmd.exe

```
E:\DAS\2016-2017\temario_ingles\session1\theory\scripts\test.mxd
Titulo: ArcMap Map Document Autor:JMP
Test file for ArcMap
```

2_access_project_python_window.py

# Accessing to the project

# If we need deal with the open project, we must use the reserved word «CURRENT» as parameter in the function MapDocument .

Python

```
>>> import arcpy
... import arcpy.mapping as map # renaming module
...
... #Definiction of variables
... ruta = 'CURRENT' #reference to the current project
... mxd = map.MapDocument(ruta)
... #Using some properties (title, author and summary)
... print "Titulo: " + mxd.title + " Autor:" + mxd.author
... print mxd.summary
...
Titulo: ArcMap Map Document Autor:JMP
Test file for ArcMap
>>>
```

3_access_project_current.py

# Accessing to the project

```
# This only works if you copy the whole code into the command line and then press
# enter.

import arcpy
import arcpy.mapping as map # renaming module

#Definiction of variables
ruta = 'CURRENT' #reference to the current project
mxd = map.MapDocument(ruta)
#Using some properties (title, author and summary)
print "Titulo: " + mxd.title + " Autor:" + mxd.author
print mxd.summary
```

3_access_project_current.py

# Refresh of the current map

# Sometimes, when we use some tools, like add a new layer, we need to update the TOC (table of contents) and the active view. In order to do this, Arcpy has two commands: "**RefreshTOC**" and "**RefreshActiveView**“.

# arcpy.mapping.ListDataFrames (MapDocument): returns a list of DataFrame objects that exists within a single map document (.mxd)
# arcpy.mapping.Layer(string): returns a layer object according to the path indicated as parameter.
# arcpy.mapping.AddLayer(DataFrame,Layer): Add a new layer to an specific DataFrame. Add a new layer to an specific DataFrame. After that, we need to refresh both, the TOC and the active view.

4_map_refresh_current.py

# Refresh of the current map

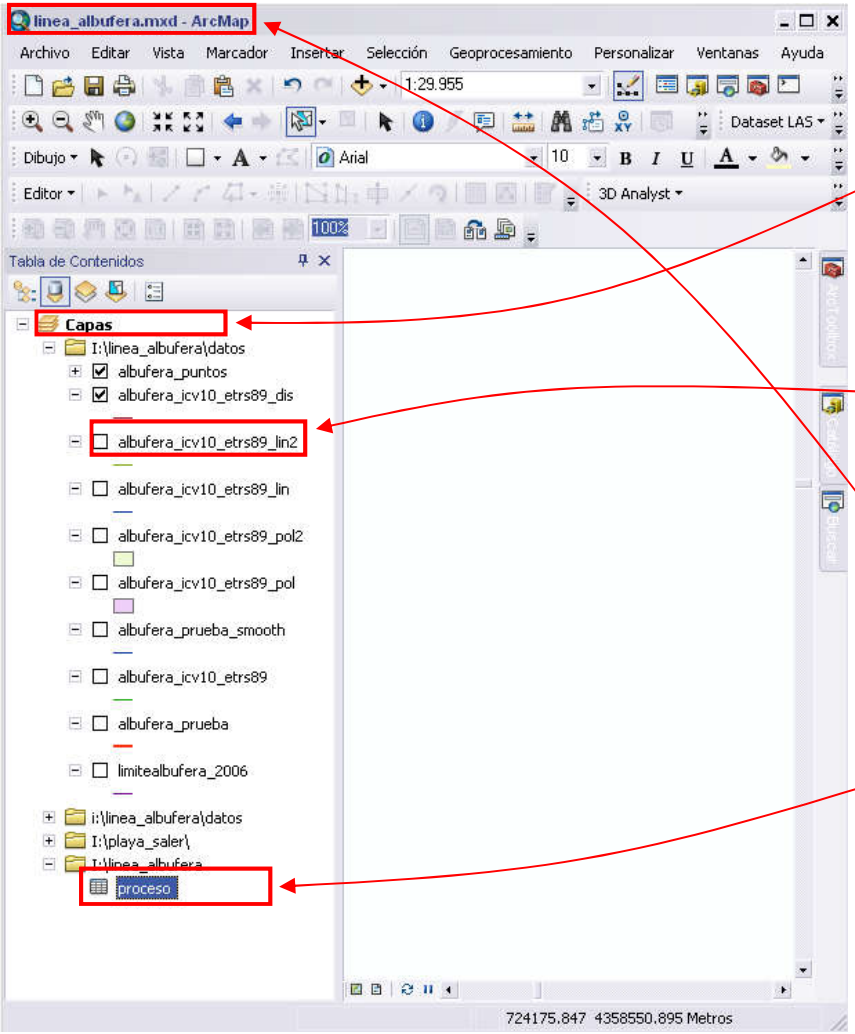
```
import arcpy
import arcpy.mapping as map

#Definiction of variables
ruta = 'CURRENT' #reference to the current project
mxd = map.MapDocument(ruta) #returns the MapDocument
mapa = map.ListDataFrames(mxd)[0] #returns the first dataframe within the MapDocument
#Layer reference accordind a specific path
capa = map.Layer(r'C:\asignaturas\sig avanzado\2014-2015\sesiones\sesion1\practica\castilla-leon\MUNICIPIO.shp')
#Add the new layer
map.AddLayer(mapa,capa)
#Refreshing of the TOC and the active view
arcpy.RefreshActiveView
arcpy.RefreshTOC
```

Currently, if we use the "AddMap" method, refreshing functions are no longer necessaries.

4_map_refresh_current.py

# Arcpy.mapping

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/00s300000032000000/




## Classes

- DataDrivenPages
- DataFrameTime
- DataFrame
- GraduatedColorsSymbology
- GraduatedSymbolsSymbology
- GraphicElement
- LabelClass
- Layer
- LayerTime
- LegendElement
- MapDocument
- MapsurroundElement
- PDFDocument
- PictureElement
- RasterClassifiedSymbology
- StyleItem
- TableView
- TextElement
- UniqueValuesSymbology

## Functions

Managing Documents and Layers

- AddLayer
- AddLayerToGroup
- AddTableView
- AnalyzeForMSD
- AnalyzeForSD
- ConvertToMSD
- ConvertWebMapToMapDocument
- CreateGISServerConnectionFile
- CreateMapSDDraft
- DeleteMapService
- InsertLayer
- Layer
- ListBookmarks
- ListBrokenDataSources
- ListDataFrames
- ListLayers
- ListLayoutElements
- ListMapServices
- ListStyleItems
- ListTableViews
- MapDocument
- MoveLayer
- PDFDocumentCreate
- PDFDocumentOpen
- PublishMSDToServer
- RemoveLayer
- RemoveTableView
- TableView
- UpdateLayer

# DataFrame object

# arcpy.mapping.ListDataFrames (MapDocument): returns a list of DataFrame objects that exists within a single map document (.mxd).

ListDataFrames(mxd)[0] : returns the first DataFrame.

# DataFrame properties:
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/00s300000003000000/

- Name (r/w): data frame's name.
- MapUnits (r): map units.
- Scale (r/w): map scale.

# Methods:

- panToExtent (extent): Pans and centers the data frame extent using a new Extent object without changing the data frame's scale.

  Example: myMap.panToExtent(myLayer. getExtent())
- zoomToSelectedFeatures (): Changes the data frame extent to match the extent of the currently selected features for all layers in a data frame.

5_access_dataframe.py

# DataFrame object

```
#Import modules
import arcpy
import arcpy.mapping as map

try:
    #Definition of varialbes
    ruta = r'E:\DAS\2016-2017\temario_ingles\session1\theory\scripts\test.mxd'
    if arcpy.Exists(ruta): #checks if the file exists
        mxd = map.MapDocument(ruta)
    else:
        raise IOError
    mapa = map.ListDataFrames(mxd)[0] #first dataframe
    #some dataframe's properties (name, units and scale)
    print 'Name of the data frame (0): ' + mapa.name
    print 'Units: ' + mapa.mapUnits
    print 'Scale: 1:' + str(mapa.scale)
except IOError:
    print 'Se ha producido un error: el archivo no existe'

    #Exercise: make a tool (in ArcToolBox) to shift the data frame extent.
    #Hint: look at the EXTENT data frame property
```

make a tool (in ArcToolBox) to shift the data frame extent.
Hint: look at the EXTENT data frame property

5_access_dataframe.py

# DataFrame object

# Some times, if we need to repeat some pieces of code, it is better to reuse it. A good way to do this is making our own library of functions in separate .py files

# This file will store all definitions of the functions we want to use.

# In order to get any function, we just use the "**import**" command.

6_access_dataframe_function.py

# DataFrame object

```python
#Import modules
import arcpy
import arcpy.mapping as map
import language as lang #access to the external module

try:
    #Definition of variables
    ruta = r'E:\DAS\2016-2017\temario_ingles\session1\theory\scripts\test.mxd'
    if arcpy.Exists(ruta):
        mxd = map.MapDocument(ruta)
    else:
        raise IOError
    df = map.ListDataFrames(mxd)[0]
    print 'Unidades: ' + df.mapUnits
    #calling changeUnits function
    print 'Unidades: ' + lang.changeUnits(df.mapUnits)

except IOError:
    print "One error has occurred: file doesn't exist"
```

```python
def changeUnits(a):
    unidades = ''
    if a == 'Meters': unidades = 'Metros'
    if a == 'Kilometers':unidades = 'Kilometros'
    return unidades
```

6_access_dataframe_function.py

# Access to data frame layers

# arcpy.mapping.ListLayers (MapDocument, {filter},{dataframe}): returns a list with layer objects within a map document or a data frame.

ListDataFrames(mxd, “”, df)[0] : first layer within a data frame.

# Layer properties (**layer class**):
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/00s300000008000000/

- name (r/w): name of the layer.
- isFeatureLayer (r): TRUE if a layer is a feature layer.
- isRasterLayer (r): TRUE if layer is a raster layer.
- dataSource (r): returns the complete path for the layer's data source.
- visible (r/w): turns on and turns off a layer.

# Methods:

- getExtent (): returns the extent of a layer. Extent is an object that stores the coordinates of the area that covers a layer.
- save () / saveACopy(): save a layer.

7_access_layers.py

# Access to data frame layers

```
#import modules
import arcpy
import arcpy.mapping as map

try:
    #Definition of variables
    ruta =  r'E:\DAS\2016-2017\temario_ingles\session1\theory\scripts\test.mxd'
    if arcpy.Exists(ruta): #checks if the file exists
        mxd = map.MapDocument(ruta)
    else:
        raise IOError
    df = map.ListDataFrames(mxd)[0] #first data frame
    capa = map.ListLayers(mxd,"",df)[0] #first layer within a data frame
    caja = capa.getExtent() #returns the layer's extent
    #print the name and the coordinates of the extent
    print capa.name, caja.XMin, caja.YMin, caja.XMax, caja.YMax
    #get the name of all layers within the data frame
    for c in map.ListLayers(mxd,"",df):
        print c.name

except IOError:
    print "file doesn't exist"

#EXERCISE. GET THE CENTROID FOR EACH LOADED LAYER
```

Get the centroid coordinates for each layer within this map document.

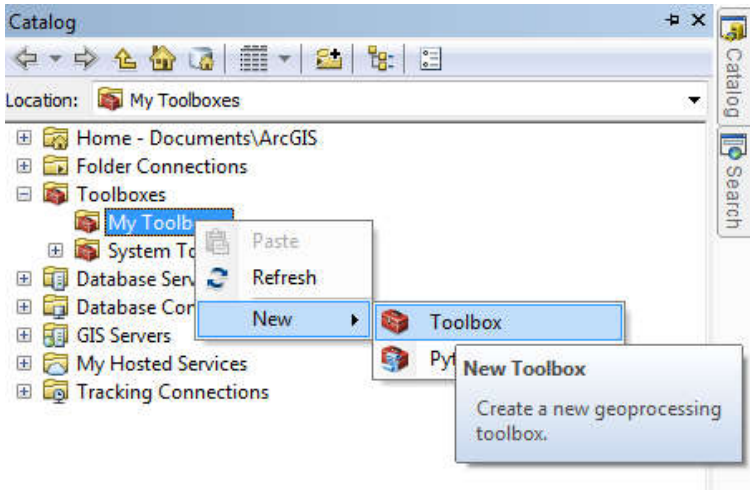
7_access_layers.py

# Running a script using ArcToolBox

# If we need to run a script with "CURRENT" (map document opened), it is better making a new tool with this script linked.
# Just pressing a button we can run the script.
# If we want to write some messages, instead using the print command, we will use the statement **arcpy.AddMessage()**. This allows us to display messages within the results window when the script is running.

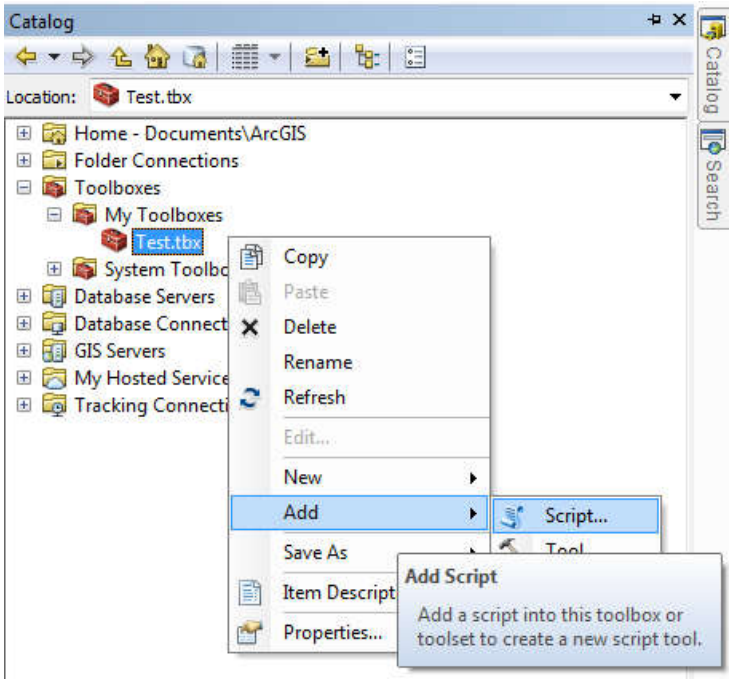
# To create a tool with a linked script:

1. Select New -> Toolbox from «My Toolboxes».



# Running a script using ArcToolBox

2. Add a new script to the custom tool box.
3. Set the name of the tool and the linked script (.py file)
4. Run the tool.




Link the script 7 to a new tool box. The script has to show the messages within the results window. change the code if is necessary.