python ArcGIS

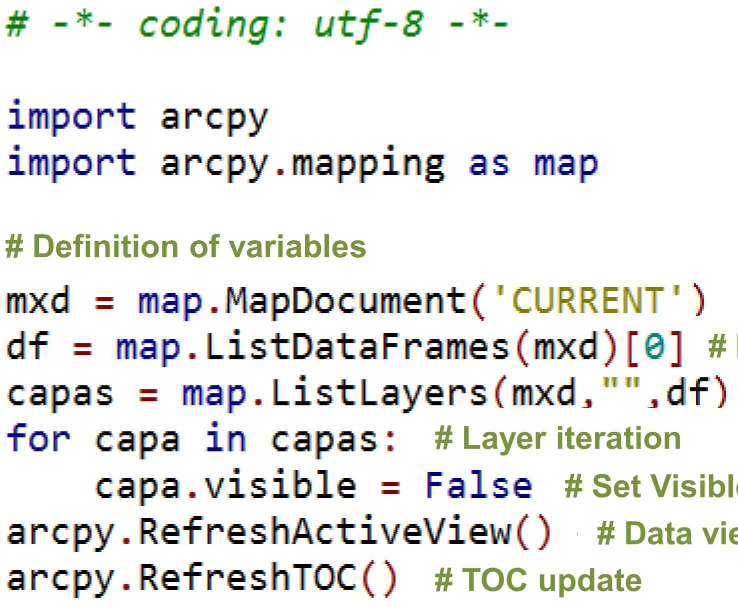
Vector layers

Layer visibility

# "Visible" is a read/write property (boolean values).
# If we are working in an open project, we must update both the data view and the TOC. In this case, we will use the next functions from Arcpy:

- arcpy.RefreshActiveView()
- arcpy.RefreshTOC()

Layer visibility

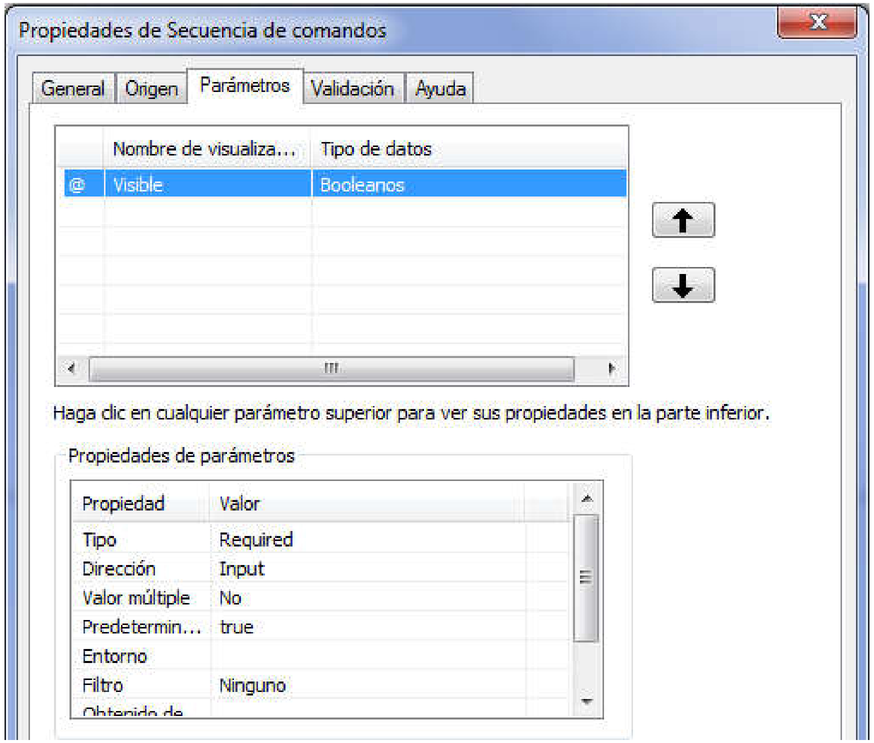


```
# -*- coding: utf-8 -*-

import arcpy
import arcpy.mapping as map

# Definition of variables
mxd = map.MapDocument('CURRENT')
df = map.ListDataFrames(mxd)[0] # Dataframe 0
capas = map.ListLayers(mxd,"",df) # List of layers
for capa in capas:  # Layer iteration
    capa.visible = False  # Set Visible property to off
arcpy.RefreshActiveView()  # Data view update
arcpy.RefreshTOC()  # TOC update
```

Layer visibility

# It is much more interesting to use some parameters by mean ArcToolBox.

# To do this, firstly we will add a new script. After that, from the parameters tab, we will append a new parameter with some properties (name, data type, etc.). In Arcpy, the correct way to access to a parameter is using the function: Arcpy.GetParameter(i), where "i" means the order index of each parameter.



	Nombre de visualiza...	Tipo de datos
@	Visible	Booleanos

Haga clic en cualquier parámetro superior para ver sus propiedades en la parte inferior.

Propiedades de parámetros

Propiedad	Valor
Tipo	Required
Dirección	Input
Valor múltiple	No
Predetermin...	true
Entorno	
Filtro	Ninguno
Obtenido de	

Layer visibility

```
import arcpy
import arcpy.mapping as map #cambio el nombre del modulo

#Definicion de variables
ruta_mxd = 'CURRENT'
estado = arcpy.GetParameter(0) #parametro de tipo booleano (devuelve 0-1)
arcpy.AddMessage(estado) #muestra el valor de estado en la caja de mensajes
mxd = map.MapDocument(ruta_mxd) #acceso al proyecto
df = map.ListDataFrames(mxd)[0] #acceso al primer mapa de la lista (dataframe)
#Acceso a todas las capas y cambio de su estado de visibilidad
for c in map.ListLayers(mxd,"",df):
    c.visible = estado

#Refresco de la vista y de la TOC
arcpy.RefreshActiveView()
arcpy.RefreshTOC()

#liberacion de memoria
del mxd
del df
del c
```

Change the code in a suitable way in order to toggle the visibility of each layer

1_layers_on_off.py

Changing the layer order in the TOC

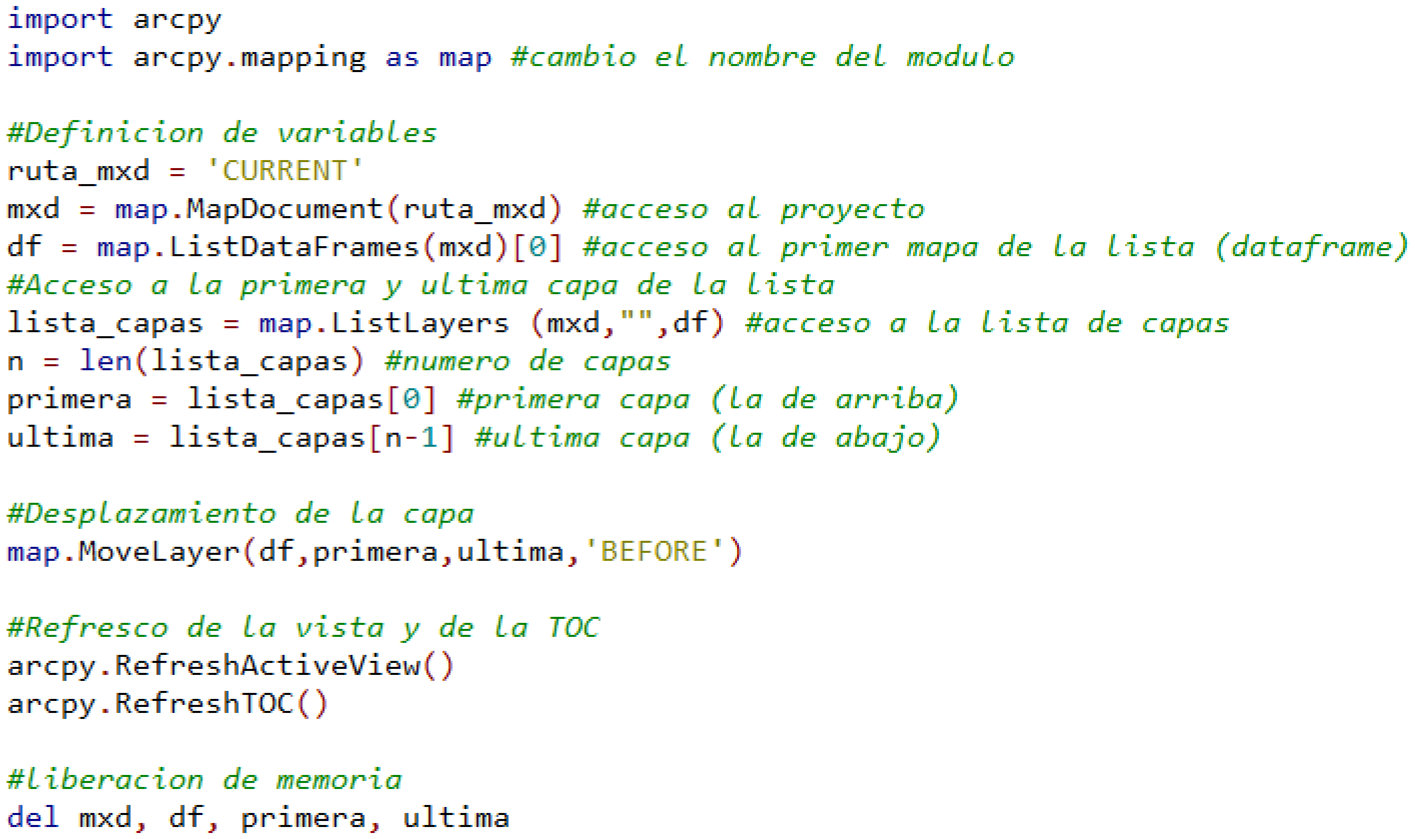
# One of the functions in Arcpy.mapping is "MoveLayer". This function allows to move the layer order in the TOC.

# arcpy.mapping.MoveLayer(dataframe, reference_layer, move_layer,{insert_position}):

- dataframe: current dataframe
- reference_layer: reference position
- move_layer: layer to be moved (current position)
- insert_position: “BEFORE” o “AFTER”. Insert the move_layer before or afther the reference layer

2_move_layer.py

Changing the layer order in the TOC



```
import arcpy
import arcpy.mapping as map #cambio el nombre del modulo

#Definicion de variables
ruta_mxd = 'CURRENT'
mxd = map.MapDocument(ruta_mxd) #acceso al proyecto
df = map.ListDataFrames(mxd)[0] #acceso al primer mapa de la lista (dataframe)
#Acceso a la primera y ultima capa de la lista
lista_capas = map.ListLayers (mxd,"",df) #acceso a la lista de capas
n = len(lista_capas) #numero de capas
primera = lista_capas[0] #primera capa (la de arriba)
ultima = lista_capas[n-1] #ultima capa (la de abajo)

#Desplazamiento de la capa
map.MoveLayer(df,primera,ultima,'BEFORE')

#Refresco de la vista y de la TOC
arcpy.RefreshActiveView()
arcpy.RefreshTOC()

#Liberacion de memoria
del mxd, df, primera, ultima
```

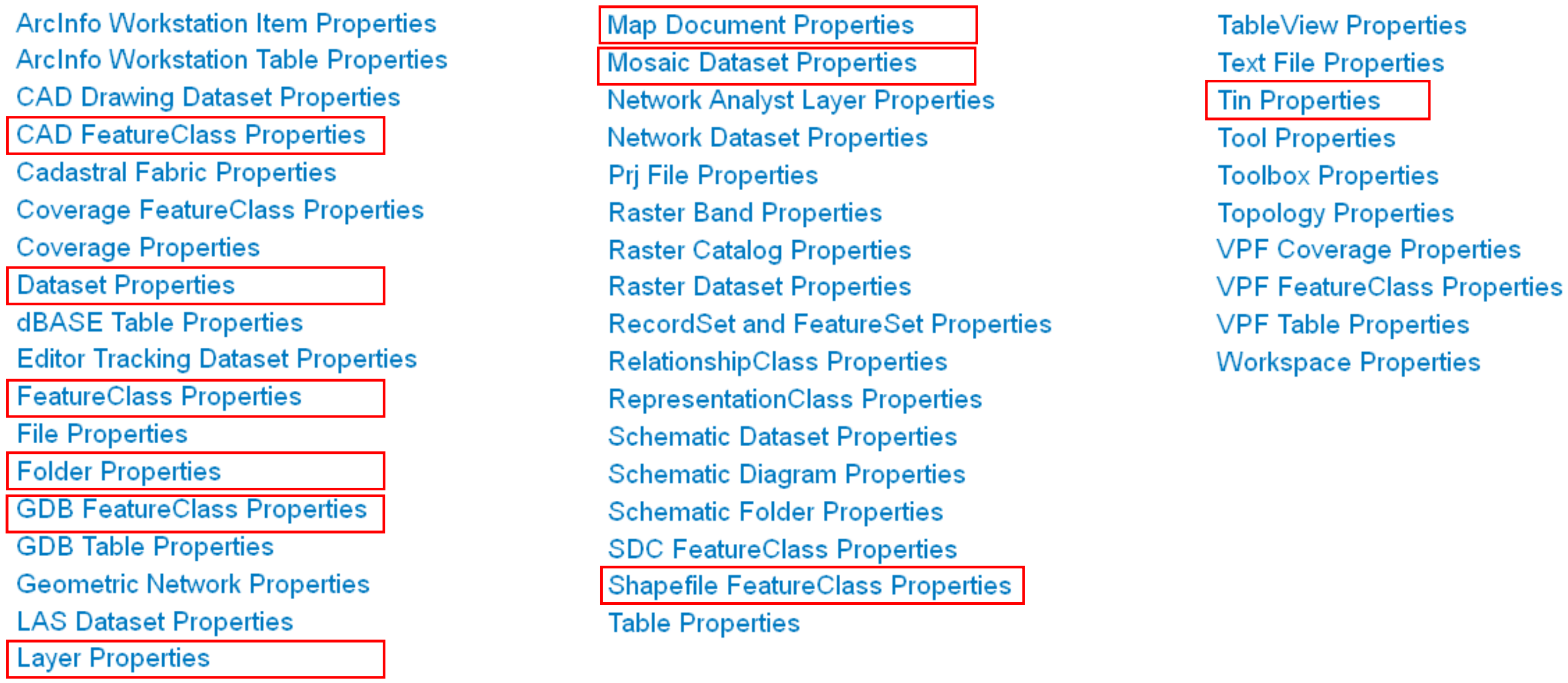
2_move_layer.py

Arcpy.describe

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018v00000066000000/

# Arcpy provides a useful function to describe data: Arcpy.Describe. This function retrieves different types of properties depending on the type of object to be described.

# arcpy.describe (data source).



ArcInfo Workstation Item Properties
ArcInfo Workstation Table Properties
CAD Drawing Dataset Properties
CAD FeatureClass Properties
Cadastral Fabric Properties
Coverage FeatureClass Properties
Coverage Properties
Dataset Properties
dBASE Table Properties
Editor Tracking Dataset Properties
FeatureClass Properties
File Properties
Folder Properties
GDB FeatureClass Properties
GDB Table Properties
Geometric Network Properties
LAS Dataset Properties
Layer Properties
Map Document Properties
Mosaic Dataset Properties
Network Analyst Layer Properties
Network Dataset Properties
Prj File Properties
Raster Band Properties
Raster Catalog Properties
Raster Dataset Properties
RecordSet and FeatureSet Properties
RelationshipClass Properties
RepresentationClass Properties
Schematic Dataset Properties
Schematic Diagram Properties
Schematic Folder Properties
SDC FeatureClass Properties
Shapefile FeatureClass Properties
Table Properties
TableView Properties
Text File Properties
Tin Properties
Tool Properties
Toolbox Properties
Topology Properties
VPF Coverage Properties
VPF FeatureClass Properties
VPF Table Properties
Workspace Properties

Arcpy.describe

# To describe an object "layer", we will use the next syntax:

```
desc = arcpy.Describe(layer.dataSource)
```

# Some properties are common to all data (e.g. a shapefile):
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018v00000013000000/

```
Name: PROVINCIA.shp
DataType: ShapeFile
CatalogPath: I:\tutorial_gvsig\carto\datos\castilla-leon\PROVINCIA.shp
BaseName: PROVINCIA
Extension: shp
DataElementType: DEShapeFile
File: PROVINCIA.shp
```

3_describe_layers.py

Arcpy.describe

# If we want to make sure whether an object has or not a property, we can use the function "**hasattr**".

```
desc = arcpy.Describe(layer.dataSource)
if hasattr(desc, “name”):
        print ‘Name: ‘ + desc.name
```

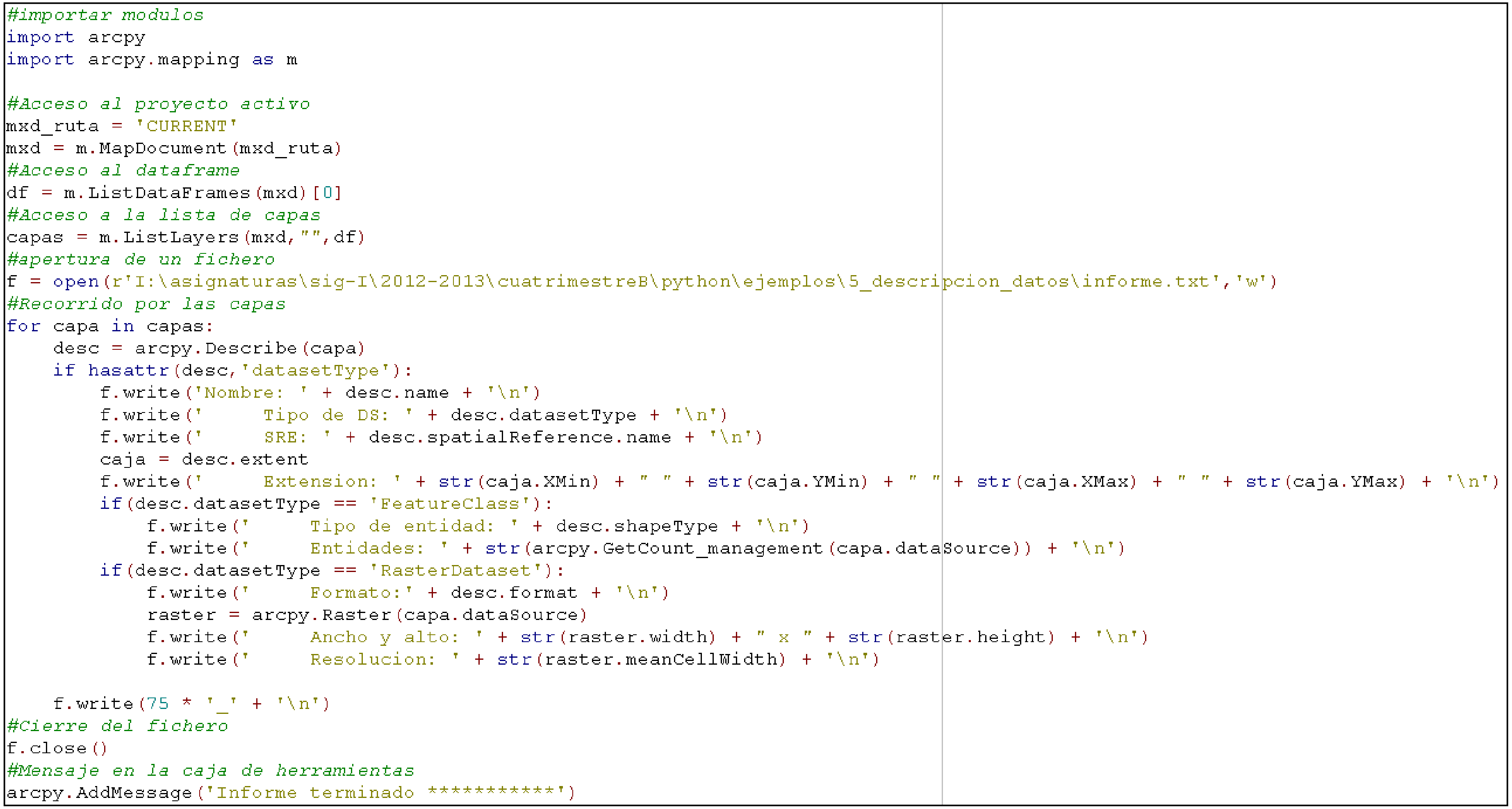
# The function "describe" returns different properties for each object. In the event of a shapefile, we can know its type of geometry in this way:

```
layer0 = ListDataFrames(mxd, “”, df)[0]
desc = arcpy.Describe(layer0)
type_geom = desc.featureClass.shapeType
```

3_describe_layers.py

Arcpy.describe

Example 1



```python
#importar modulos
import arcpy
import arcpy.mapping as m

#Acceso al proyecto activo
mxd_ruta = 'CURRENT'
mxd = m.MapDocument(mxd_ruta)
#Acceso al dataframe
df = m.ListDataFrames(mxd)[0]
#Acceso a la lista de capas
capas = m.ListLayers(mxd,"",df)
#apertura de un fichero
f = open(r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\5_descripcion_datos\informe.txt','w')
#Recorrido por las capas
for capa in capas:
    desc = arcpy.Describe(capa)
    if hasattr(desc,'datasetType'):
        f.write('Nombre: ' + desc.name + '\n')
        f.write('     Tipo de DS: ' + desc.datasetType + '\n')
        f.write('     SRE: ' + desc.spatialReference.name + '\n')
        caja = desc.extent
        f.write('     Extension: ' + str(caja.XMin) + " " + str(caja.YMin) + " " + str(caja.XMax) + " " + str(caja.YMax) + '\n')
        if(desc.datasetType == 'FeatureClass'):
            f.write('     Tipo de entidad: ' + desc.shapeType + '\n')
            f.write('     Entidades: ' + str(arcpy.GetCount_management(capa.dataSource)) + '\n')
        if(desc.datasetType == 'RasterDataset'):
            f.write('     Formato:' + desc.format + '\n')
            raster = arcpy.Raster(capa.dataSource)
            f.write('     Ancho y alto: ' + str(raster.width) + " x " + str(raster.height) + '\n')
            f.write('     Resolucion: ' + str(raster.meanCellWidth) + '\n')

    f.write(75 * '_' + '\n')
#Cierre del fichero
f.close()
#Mensaje en la caja de herramientas
arcpy.AddMessage('Informe terminado ***********')
```

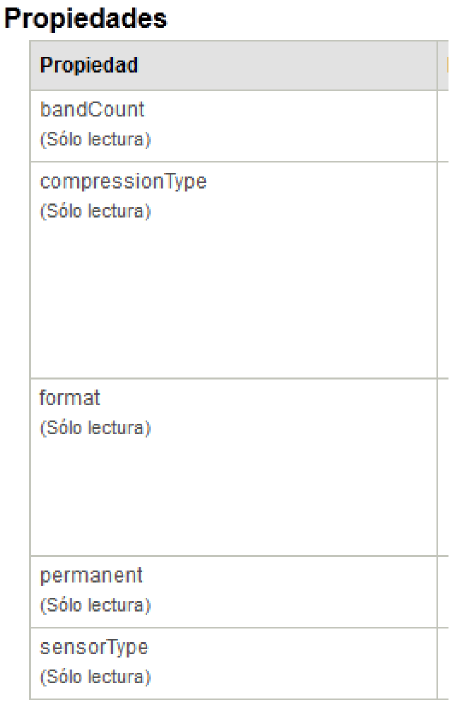
3_describe_layers.py

Arcpy.describe

Example 2

# Sometimes, we can describe a datasource in other ways. The "Raster" class from Arcpy, has a lot of properties that we can use.

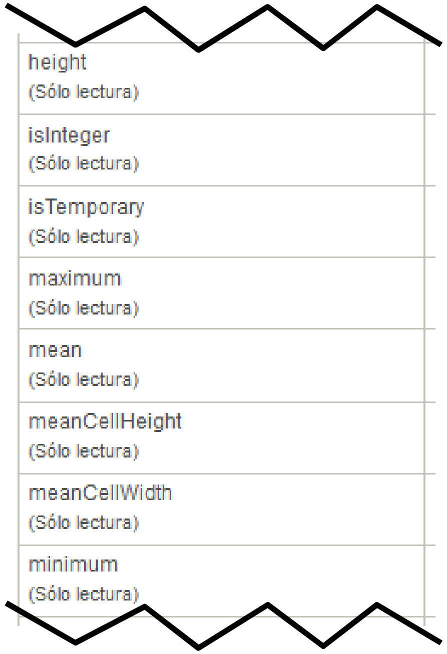
Raster DataSet properties



Propiedades

Propiedad
bandCount (Sólo lectura)
compressionType (Sólo lectura)
format (Sólo lectura)
permanent (Sólo lectura)
sensorType (Sólo lectura)

Raster properties



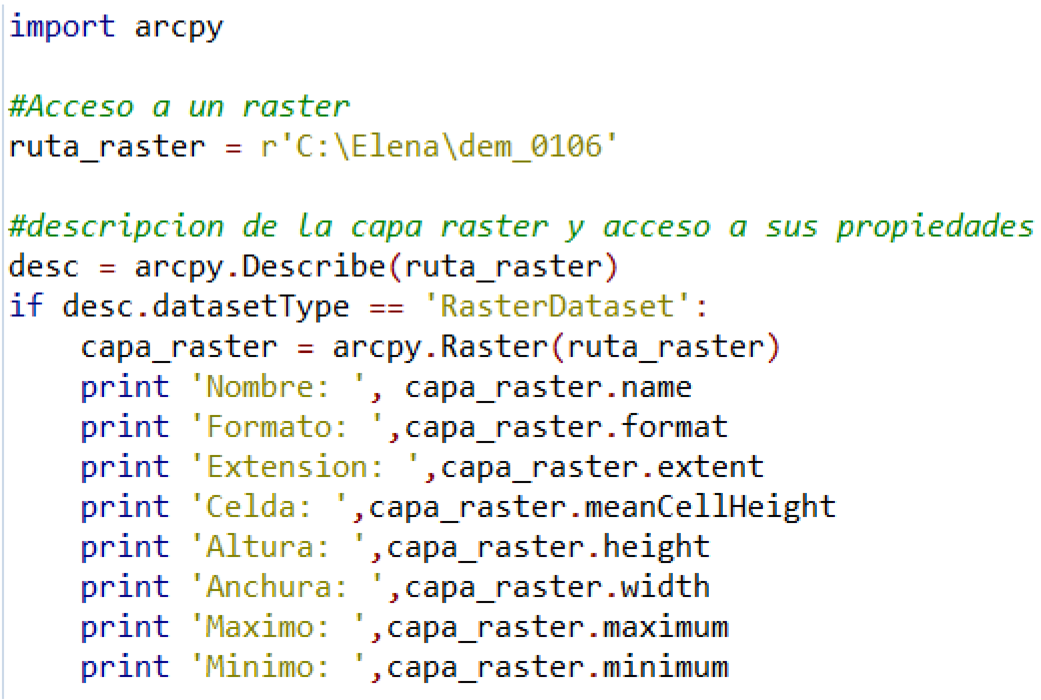
height (Sólo lectura)
isInteger (Sólo lectura)
isTemporary (Sólo lectura)
maximum (Sólo lectura)
mean (Sólo lectura)
meanCellHeight (Sólo lectura)
meanCellWidth (Sólo lectura)
minimum (Sólo lectura)

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018v0000000t000000
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018z00000051000000

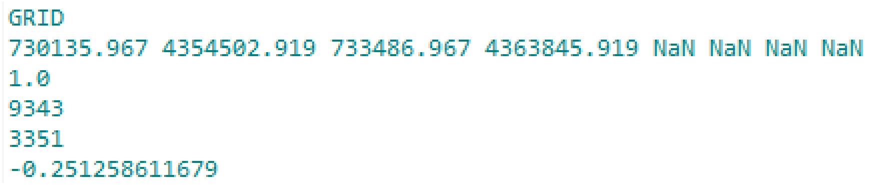
Arcpy.describe

Example 2

# Sometimes, we can describe a datasource in other ways. The "Raster" class from Arcpy, has a lot of properties that we can use.



```
import arcpy

#Acceso a un raster
ruta_raster = r'C:\Elena\dem_0106'

#descripcion de la capa raster y acceso a sus propiedades
desc = arcpy.Describe(ruta_raster)
if desc.datasetType == 'RasterDataset':
    capa_raster = arcpy.Raster(ruta_raster)
    print 'Nombre: ', capa_raster.name
    print 'Formato: ',capa_raster.format
    print 'Extension: ',capa_raster.extent
    print 'Celda: ',capa_raster.meanCellHeight
    print 'Altura: ',capa_raster.height
    print 'Anchura: ',capa_raster.width
    print 'Maximo: ',capa_raster.maximum
    print 'Minimo: ',capa_raster.minimum
```

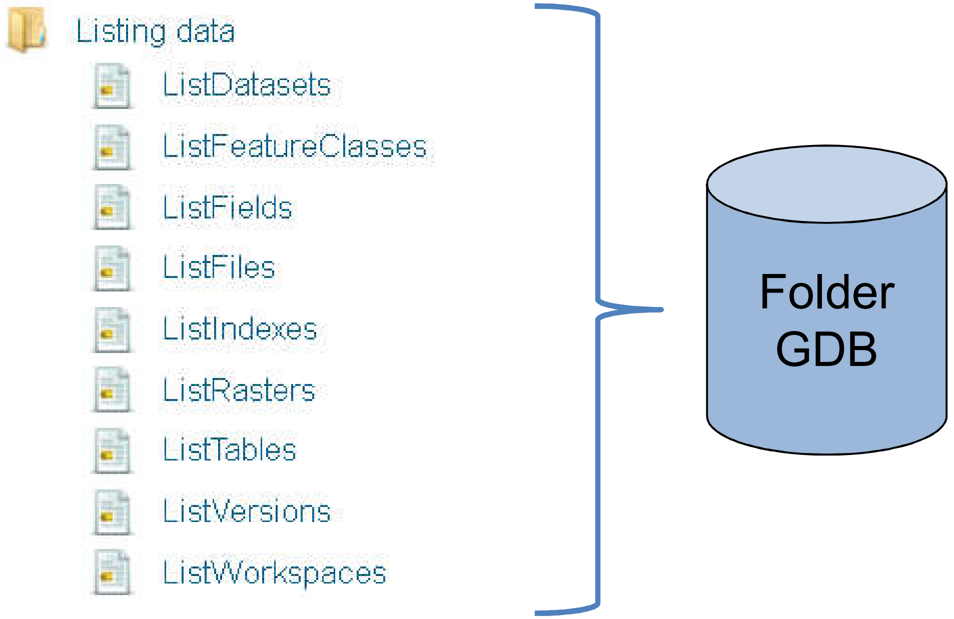


```
GRID
730135.967 4354502.919 733486.967 4363845.919 NaN NaN NaN NaN
1.0
9343
3351
-0.251258611679
```

Listing data.

# Arcpy has several useful functions to obtain a list of different type of data:

- arcpy.**ListDataSets** ({wild card},{feature type}). It returns a list of all data within a workspace (folder, GDB, etc.). We can set up the workspace using the property **arcpy.env.workspace**.

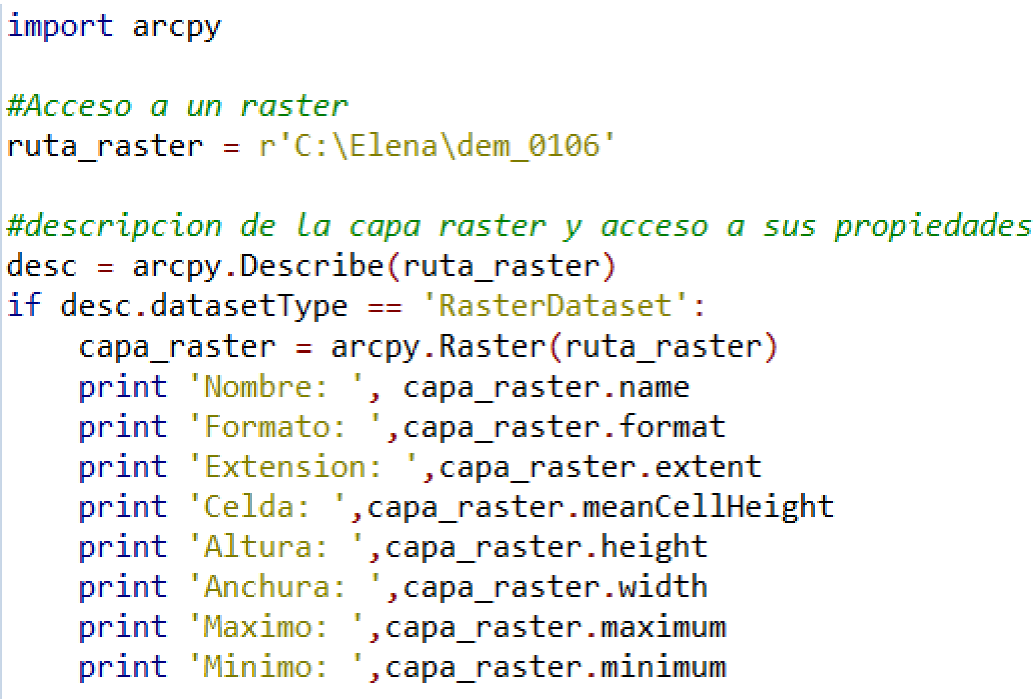


Listing data

- ListDatasets
- ListFeatureClasses
- ListFields
- ListFiles
- ListIndexes
- ListRasters
- ListTables
- ListVersions
- ListWorkspaces

Folder
GDB

Listing data.



```
import arcpy

#Acceso a un raster
ruta_raster = r'C:\Elena\dem_0106'

#descripcion de la capa raster y acceso a sus propiedades
desc = arcpy.Describe(ruta_raster)
if desc.datasetType == 'RasterDataset':
    capa_raster = arcpy.Raster(ruta_raster)
    print 'Nombre: ', capa_raster.name
    print 'Formato: ',capa_raster.format
    print 'Extension: ',capa_raster.extent
    print 'Celda: ',capa_raster.meanCellHeight
    print 'Altura: ',capa_raster.height
    print 'Anchura: ',capa_raster.width
    print 'Maximo: ',capa_raster.maximum
    print 'Minimo: ',capa_raster.minimum
```

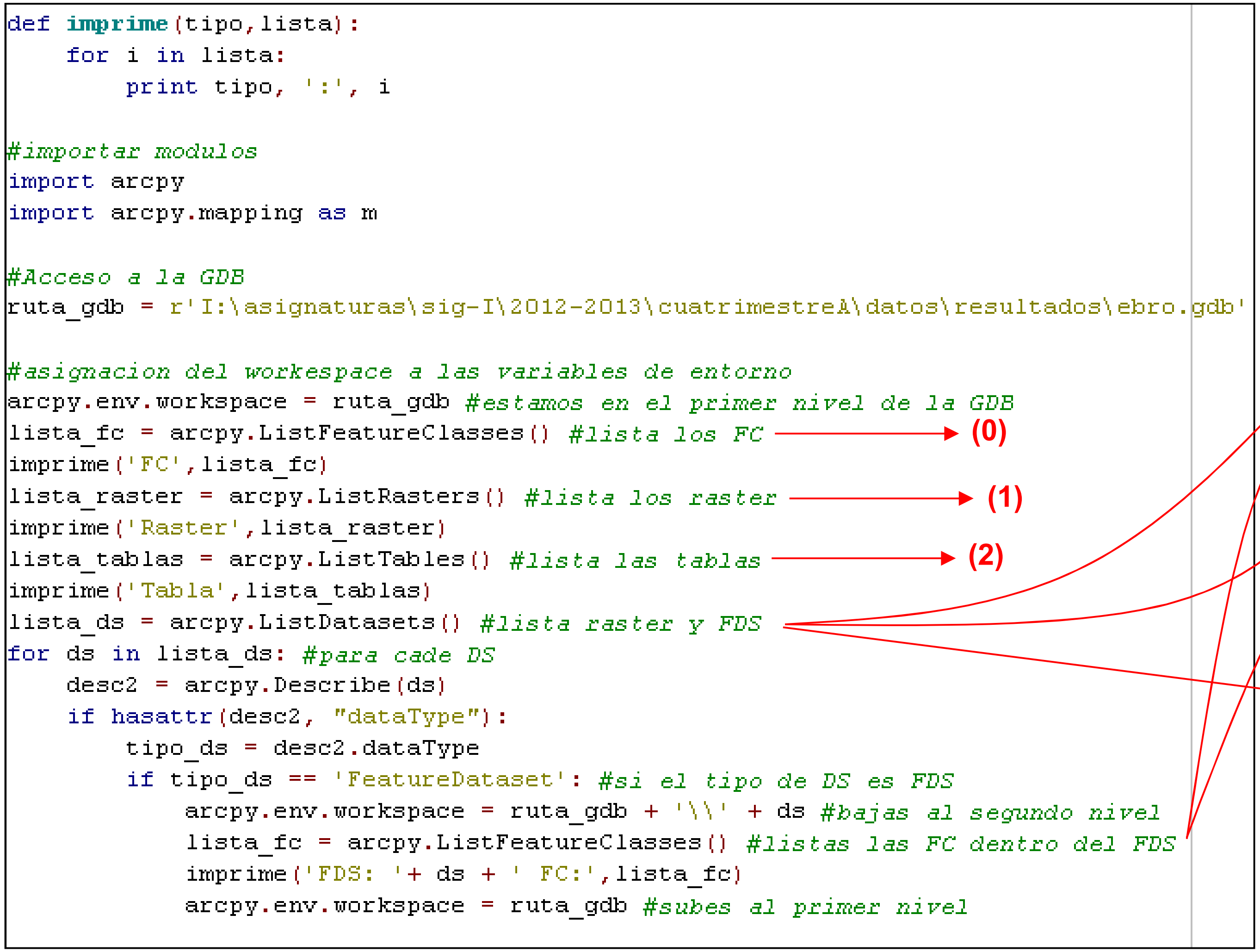
Change this code in order to do the same with all raster dataset within a folder. You have to find out the suitable method.

Suggestion: find out information in the next links ("current workspace" and "list rasters").

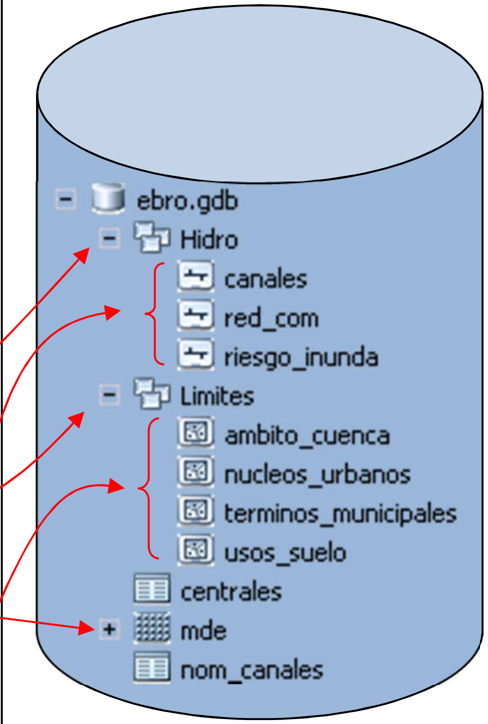
http://resources.arcgis.com/en/help/main/10.1/index.html#/na/001w00000002000000/

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018v0000003w000000/

GDB content.



```
def imprime(tipo,lista):
    for i in lista:
        print tipo, ':', i

#importar modulos
import arcpy
import arcpy.mapping as m

#Acceso a la GDB
ruta_gdb = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreA\datos\resultados\ebro.gdb'

#asignacion del workspace a las variables de entorno
arcpy.env.workspace = ruta_gdb #estamos en el primer nivel de la GDB
lista_fc = arcpy.ListFeatureClasses() #lista los FC
imprime('FC',lista_fc)
lista_raster = arcpy.ListRasters() #lista los raster
imprime('Raster',lista_raster)
lista_tablas = arcpy.ListTables() #lista las tablas
imprime('Tabla',lista_tablas)
lista_ds = arcpy.ListDatasets() #lista raster y FDS
for ds in lista_ds: #para cade DS
    desc2 = arcpy.Describe(ds)
    if hasattr(desc2, "dataType"):
        tipo_ds = desc2.dataType
        if tipo_ds == 'FeatureDataset': #si el tipo de DS es FDS
            arcpy.env.workspace = ruta_gdb + '\\' + ds #bajas al segundo nivel
            lista_fc = arcpy.ListFeatureClasses() #listas las FC dentro del FDS
            imprime('FDS: '+ ds + ' FC:',lista_fc)
            arcpy.env.workspace = ruta_gdb #subes al primer nivel
```

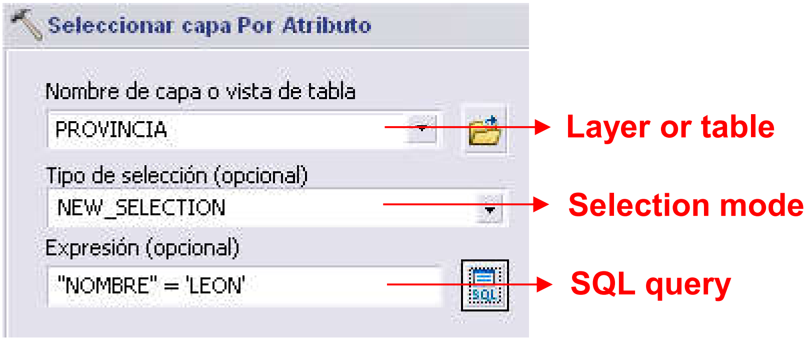


# Selection by attribute

# Selection by attribute has the same behavior that "Selection -> Select by attribute option. In ArcToolBox, SelectLayerByAttribute is the suitable tool. We can call this tool in taht way: **arcpy.SelectLayerByAttribute_management**.

# This tool needs a layer or table as input. So, if we just have a path, firstly we need make a layer from this path (FeatureLayer class). **Arcpy.MakeFeatureLayer_Management** will be the more suitable function in this case.

In arcpy, the name of a tool has the next structure: tool name, plus underscore, plus the toolbox alias where the tool is included.



Seleccionar capa Por Atributo

Nombre de capa o vista de tabla: PROVINCIA

Tipo de selección (opcional): NEW_SELECTION

Expresión (opcional): "NOMBRE" = 'LEON'

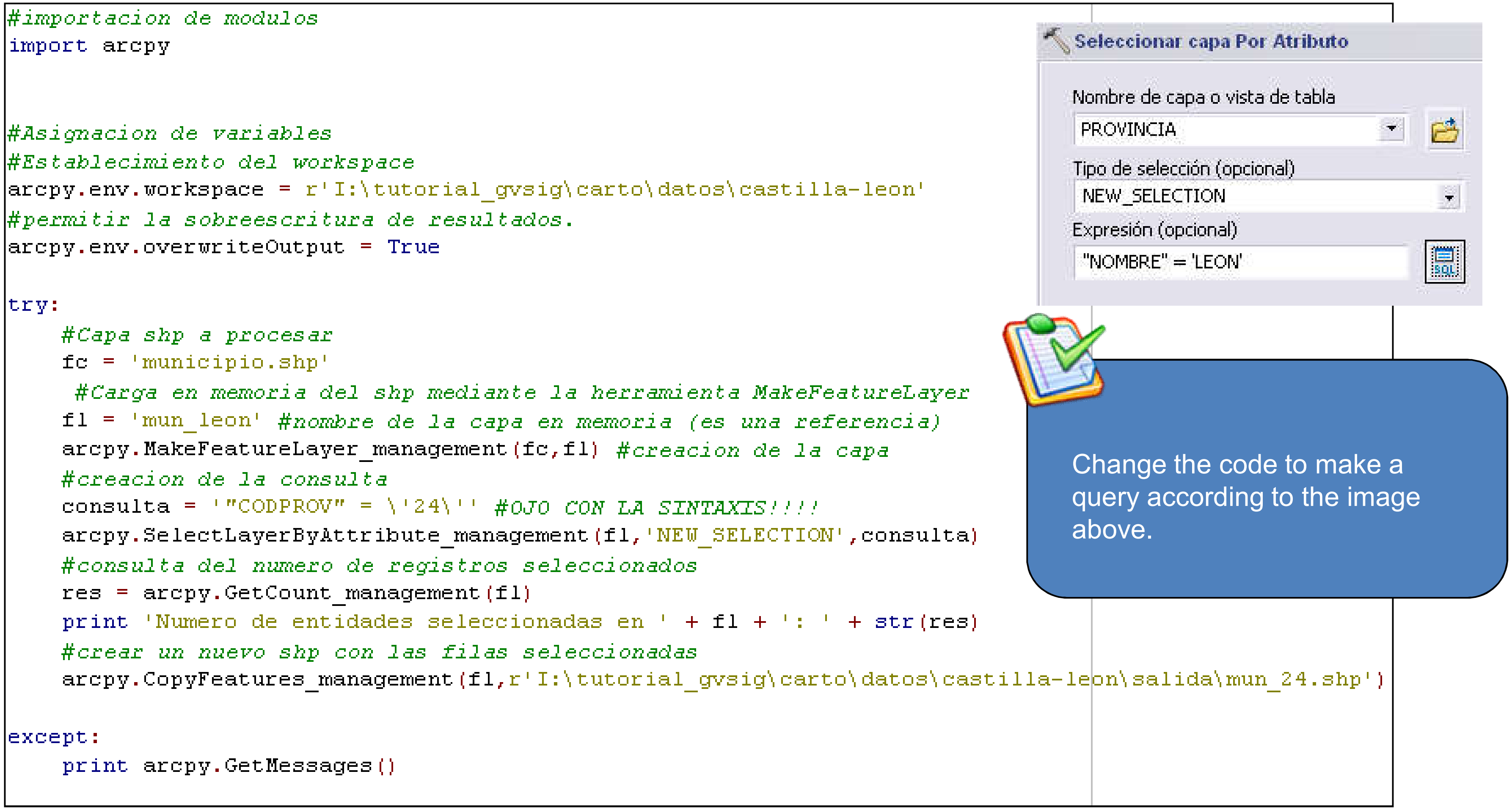
Layer or table

Selection mode

SQL query

4_select_by_attribute.py

Selection by attribute



Seleccionar capa Por Atributo

Nombre de capa o vista de tabla: PROVINCIA

Tipo de selección (opcional): NEW_SELECTION

Expresión (opcional): "NOMBRE" = 'LEON'

```
#importacion de modulos
import arcpy


#Asignacion de variables
#Establecimiento del workspace
arcpy.env.workspace = r'I:\tutorial_gvsig\carto\datos\castilla-leon'
#permitir la sobreescritura de resultados.
arcpy.env.overwriteOutput = True

try:
    #Capa shp a procesar
    fc = 'municipio.shp'
     #Carga en memoria del shp mediante la herramienta MakeFeatureLayer
    fl = 'mun_leon' #nombre de la capa en memoria (es una referencia)
    arcpy.MakeFeatureLayer_management(fc,fl) #creacion de la capa
    #creacion de la consulta
    consulta = '"CODPROV" = \'24\'' #OJO CON LA SINTAXIS!!!!
    arcpy.SelectLayerByAttribute_management(fl,'NEW_SELECTION',consulta)
    #consulta del numero de registros seleccionados
    res = arcpy.GetCount_management(fl)
    print 'Numero de entidades seleccionadas en ' + fl + ': ' + str(res)
    #crear un nuevo shp con las filas seleccionadas
    arcpy.CopyFeatures_management(fl,r'I:\tutorial_gvsig\carto\datos\castilla-leon\salida\mun_24.shp')

except:
    print arcpy.GetMessages()
```

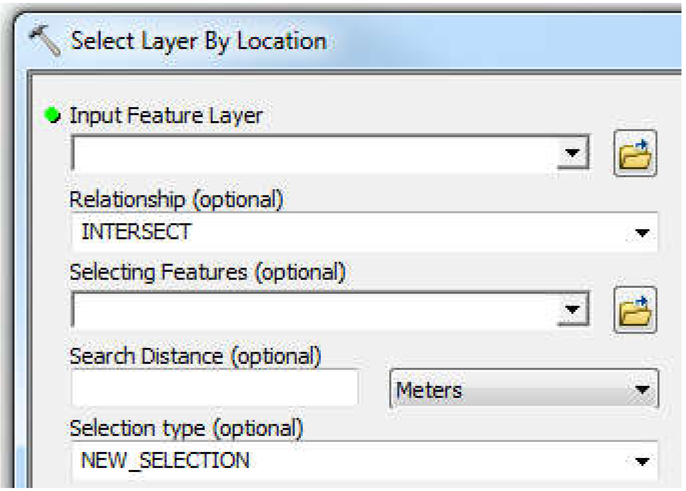
Change the code to make a query according to the image above.

4_select_by_attribute.py

# Selection by location

# Selection by location has the same behavior that "Selection -> Select by location option. In ArcToolBox, SelectLayerByLocation is the suitable tool. We can call this tool in taht way: **arcpy.SelectLayerByLocation_management**.

# This tool needs a layer or table as input. So, if we just have a path, firstly we need make a layer from this path (FeatureLayer class). **Arcpy.MakeFeatureLayer_Management** will be the more suitable function in this case.



Select Layer By Location

Input Feature Layer

Relationship (optional)
INTERSECT

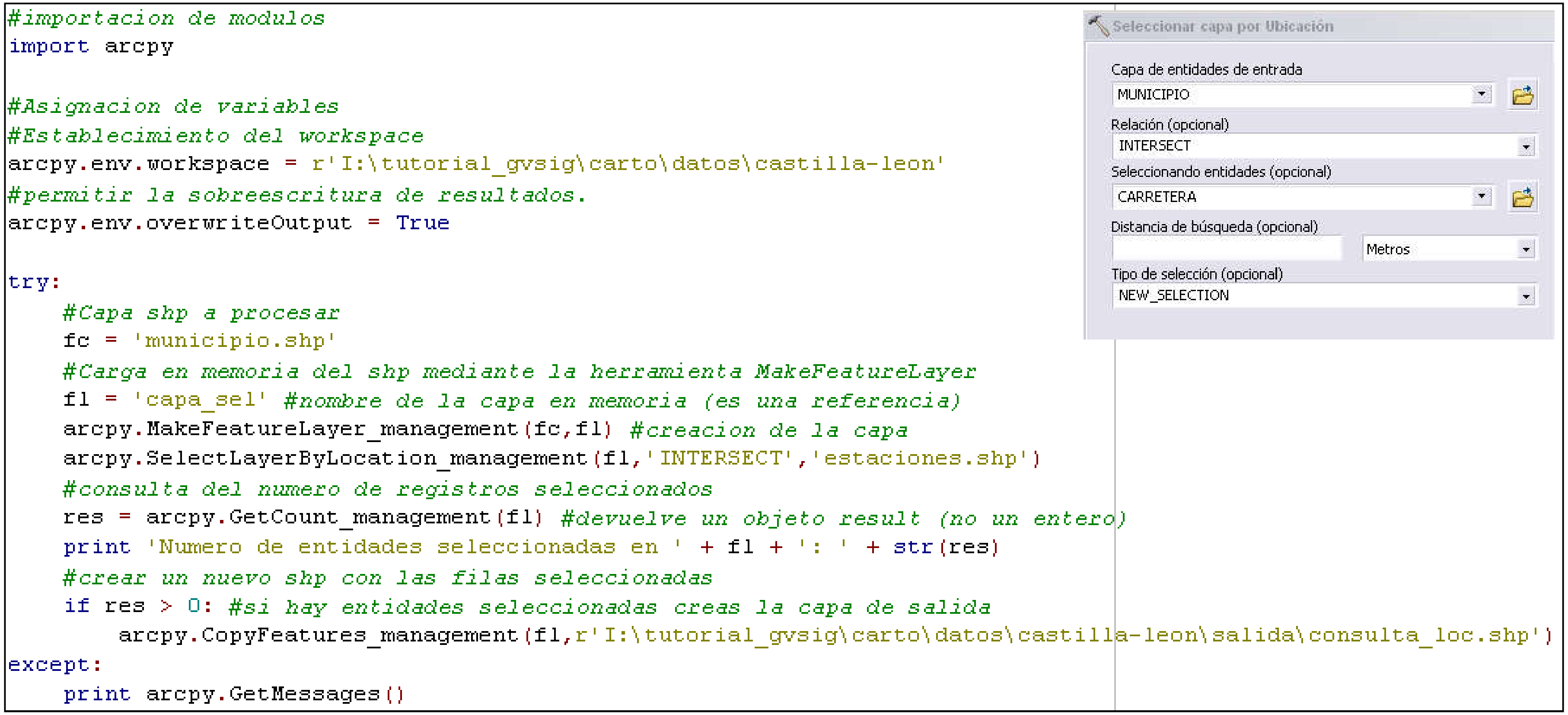
Selecting Features (optional)

Search Distance (optional)
Meters

Selection type (optional)
NEW_SELECTION

5_select_by_location.py

Selection by location



```
#importacion de modulos
import arcpy

#Asignacion de variables
#Establecimiento del workspace
arcpy.env.workspace = r'I:\tutorial_gvsig\carto\datos\castilla-leon'
#permitir la sobreescritura de resultados.
arcpy.env.overwriteOutput = True

try:
    #Capa shp a procesar
    fc = 'municipio.shp'
    #Carga en memoria del shp mediante la herramienta MakeFeatureLayer
    fl = 'capa_sel' #nombre de la capa en memoria (es una referencia)
    arcpy.MakeFeatureLayer_management(fc,fl) #creacion de la capa
    arcpy.SelectLayerByLocation_management(fl,'INTERSECT','estaciones.shp')
    #consulta del numero de registros seleccionados
    res = arcpy.GetCount_management(fl) #devuelve un objeto result (no un entero)
    print 'Numero de entidades seleccionadas en ' + fl + ': ' + str(res)
    #crear un nuevo shp con las filas seleccionadas
    if res > 0: #si hay entidades seleccionadas creas la capa de salida
        arcpy.CopyFeatures_management(fl,r'I:\tutorial_gvsig\carto\datos\castilla-leon\salida\consulta_loc.shp')
except:
    print arcpy.GetMessages()
```

Seleccionar capa por Ubicación

Capa de entidades de entrada: MUNICIPIO

Relación (opcional): INTERSECT

Seleccionando entidades (opcional): CARRETERA

Distancia de búsqueda (opcional): Metros

Tipo de selección (opcional): NEW_SELECTION

5_select_by_location.py

Data access. Cursors

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018z0000009r000000/

# A cursor is a data access object that can be used either to iterate through the set of rows in a table or to insert new rows into a table. Cursors have three forms: search, insert, or update. Cursors are commonly used to read and update attributes.

# There are three types of cursors:

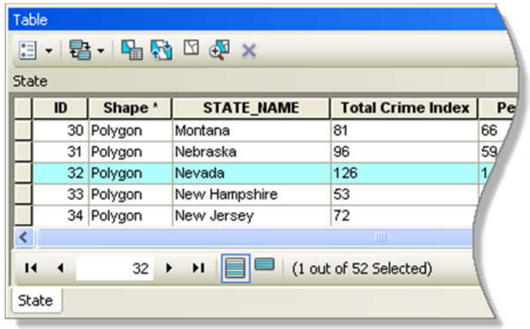
- **search** (read).
- **update** (read / write).
- **insert** (insert a new row).

# We can iterate through a table linked to a feature class or through a external table.

# The **arcpy** module provides the suitable functions to manage cursors.

# To iterate, we can use any standard loop (for or while)

Data access. Cursors



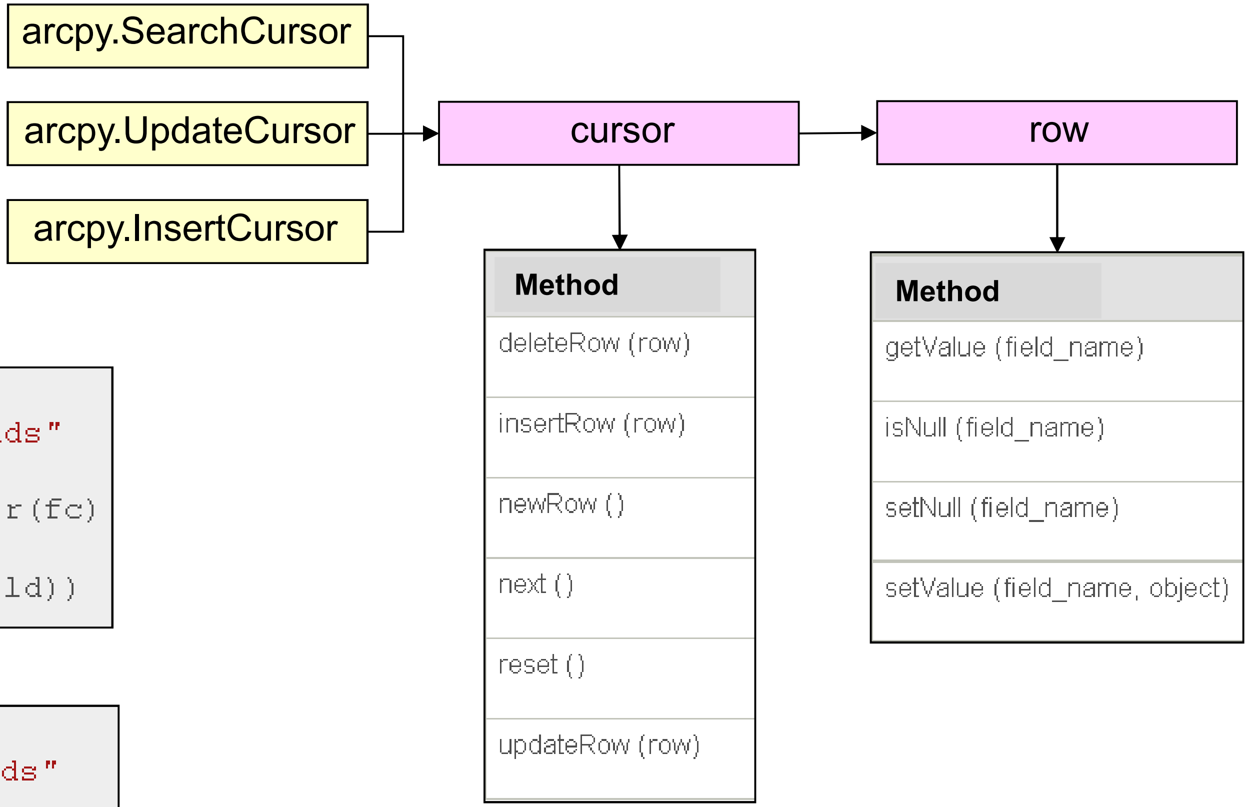
Table

State

ID	Shape *	STATE_NAME	Total Crime Index	Pe
30	Polygon	Montana	81	66
31	Polygon	Nebraska	96	59
32	Polygon	Nevada	126	1
33	Polygon	New Hampshire	53	
34	Polygon	New Jersey	72	

32 (1 out of 52 Selected)

State



Method
deleteRow (row)
insertRow (row)
newRow ()
next ()
reset ()
updateRow (row)

Method
getValue (field_name)
isNull (field_name)
setNull (field_name)
setValue (field_name, object)

For loop iteration

```
import arcpy
fc = "c:/data/base.gdb/roads"
field = "StreetName"
cursor = arcpy.SearchCursor(fc)
for row in cursor:
    print(row.getValue(field))
```

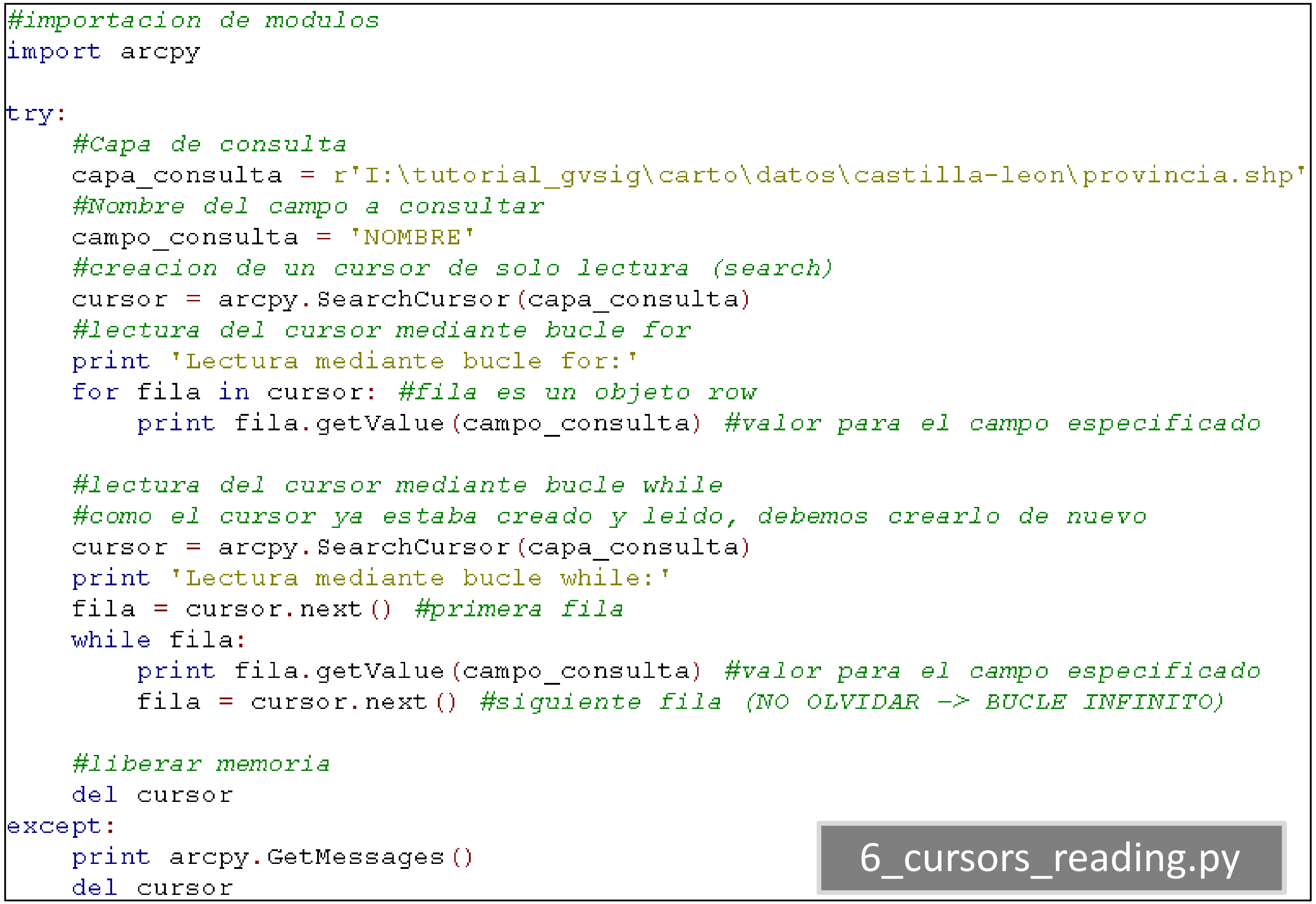
While loop iteration

```
import arcpy
fc = "c:/data/base.gdb/roads"
field = "StreetName"
cursor = arcpy.SearchCursor(fc)
row = cursor.next()
while row:
    print(row.getValue(field))
    cursor.next()
```

# SearchCursor

http://resources.arcgis.com/en/help/main/10.1/index.html#//018v00000050000000

# SearchCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})



```
#importacion de modulos
import arcpy

try:
    #Capa de consulta
    capa_consulta = r'I:\tutorial_gvsig\carto\datos\castilla-leon\provincia.shp'
    #Nombre del campo a consultar
    campo_consulta = 'NOMBRE'
    #creacion de un cursor de solo lectura (search)
    cursor = arcpy.SearchCursor(capa_consulta)
    #lectura del cursor mediante bucle for
    print 'Lectura mediante bucle for:'
    for fila in cursor: #fila es un objeto row
        print fila.getValue(campo_consulta) #valor para el campo especificado

    #lectura del cursor mediante bucle while
    #como el cursor ya estaba creado y leido, debemos crearlo de nuevo
    cursor = arcpy.SearchCursor(capa_consulta)
    print 'Lectura mediante bucle while:'
    fila = cursor.next() #primera fila
    while fila:
        print fila.getValue(campo_consulta) #valor para el campo especificado
        fila = cursor.next() #siguiente fila (NO OLVIDAR -> BUCLE INFINITO)

    #liberar memoria
    del cursor
except:
    print arcpy.GetMessages()
    del cursor
```

6_cursors_reading.py

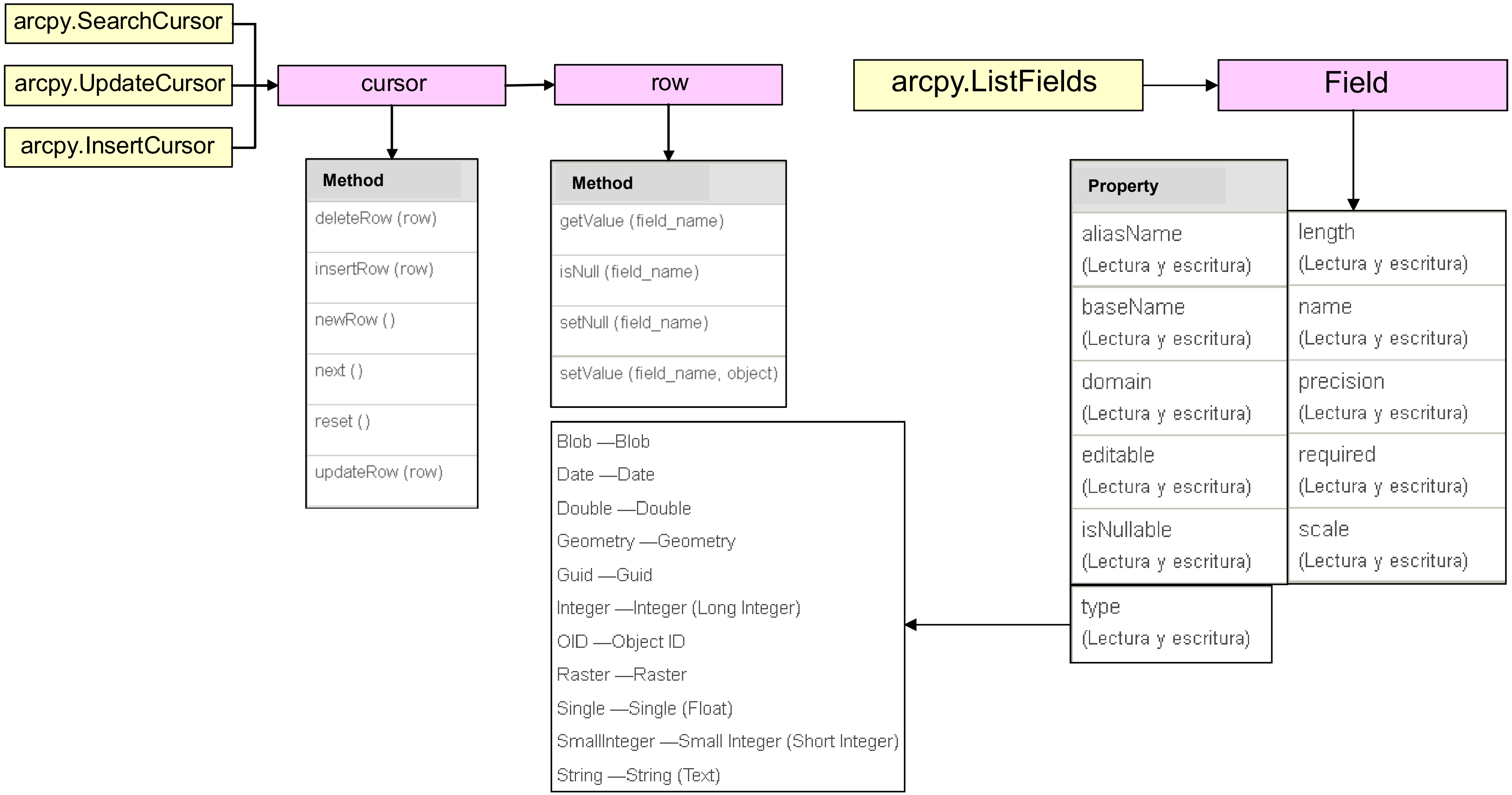
SearchCursor

Taken as template the previous code, make a script to print the result of the next query: name of all municipalities belonging to Castilla-Leon province, having more than 100.000 inhabitants in the year 1995.

SearchCursor and field access

# **ListFields** funtion returns a list with all fields in a table.

# arcpy.ListFields (dataset, {wild_card}, {field_type})



arcpy.SearchCursor

arcpy.UpdateCursor

arcpy.InsertCursor

cursor

Method
deleteRow (row)
insertRow (row)
newRow ()
next ()
reset ()
updateRow (row)

row

Method
getValue (field_name)
isNull (field_name)
setNull (field_name)
setValue (field_name, object)

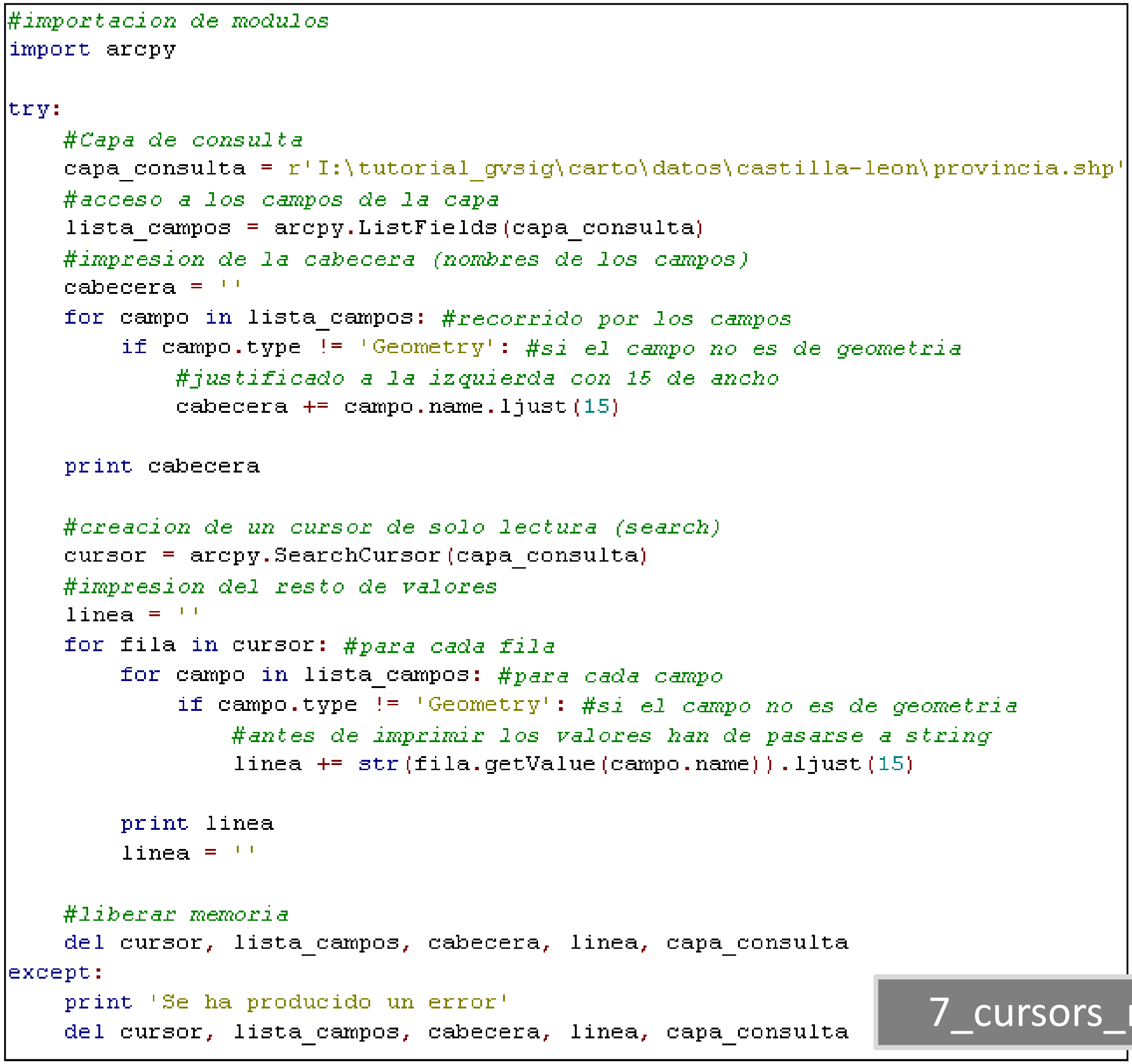
Blob —Blob
Date —Date
Double —Double
Geometry —Geometry
Guid —Guid
Integer —Integer (Long Integer)
OID —Object ID
Raster —Raster
Single —Single (Float)
SmallInteger —Small Integer (Short Integer)
String —String (Text)

arcpy.ListFields

Field

Property	
aliasName (Lectura y escritura)	length (Lectura y escritura)
baseName (Lectura y escritura)	name (Lectura y escritura)
domain (Lectura y escritura)	precision (Lectura y escritura)
editable (Lectura y escritura)	required (Lectura y escritura)
isNullable (Lectura y escritura)	scale (Lectura y escritura)
type (Lectura y escritura)	

SearchCursor and field access



```
#importacion de modulos
import arcpy

try:
    #Capa de consulta
    capa_consulta = r'I:\tutorial_gvsig\carto\datos\castilla-leon\provincia.shp'
    #acceso a los campos de la capa
    lista_campos = arcpy.ListFields(capa_consulta)
    #impresion de la cabecera (nombres de los campos)
    cabecera = ''
    for campo in lista_campos: #recorrido por los campos
        if campo.type != 'Geometry': #si el campo no es de geometria
            #justificado a la izquierda con 15 de ancho
            cabecera += campo.name.ljust(15)

    print cabecera

    #creacion de un cursor de solo lectura (search)
    cursor = arcpy.SearchCursor(capa_consulta)
    #impresion del resto de valores
    linea = ''
    for fila in cursor: #para cada fila
        for campo in lista_campos: #para cada campo
            if campo.type != 'Geometry': #si el campo no es de geometria
                #antes de imprimir los valores han de pasarse a string
                linea += str(fila.getValue(campo.name)).ljust(15)

        print linea
        linea = ''

    #liberar memoria
    del cursor, lista_campos, cabecera, linea, capa_consulta
except:
    print 'Se ha producido un error'
    del cursor, lista_campos, cabecera, linea, capa_consulta
```

7_cursors_reading_fields.py

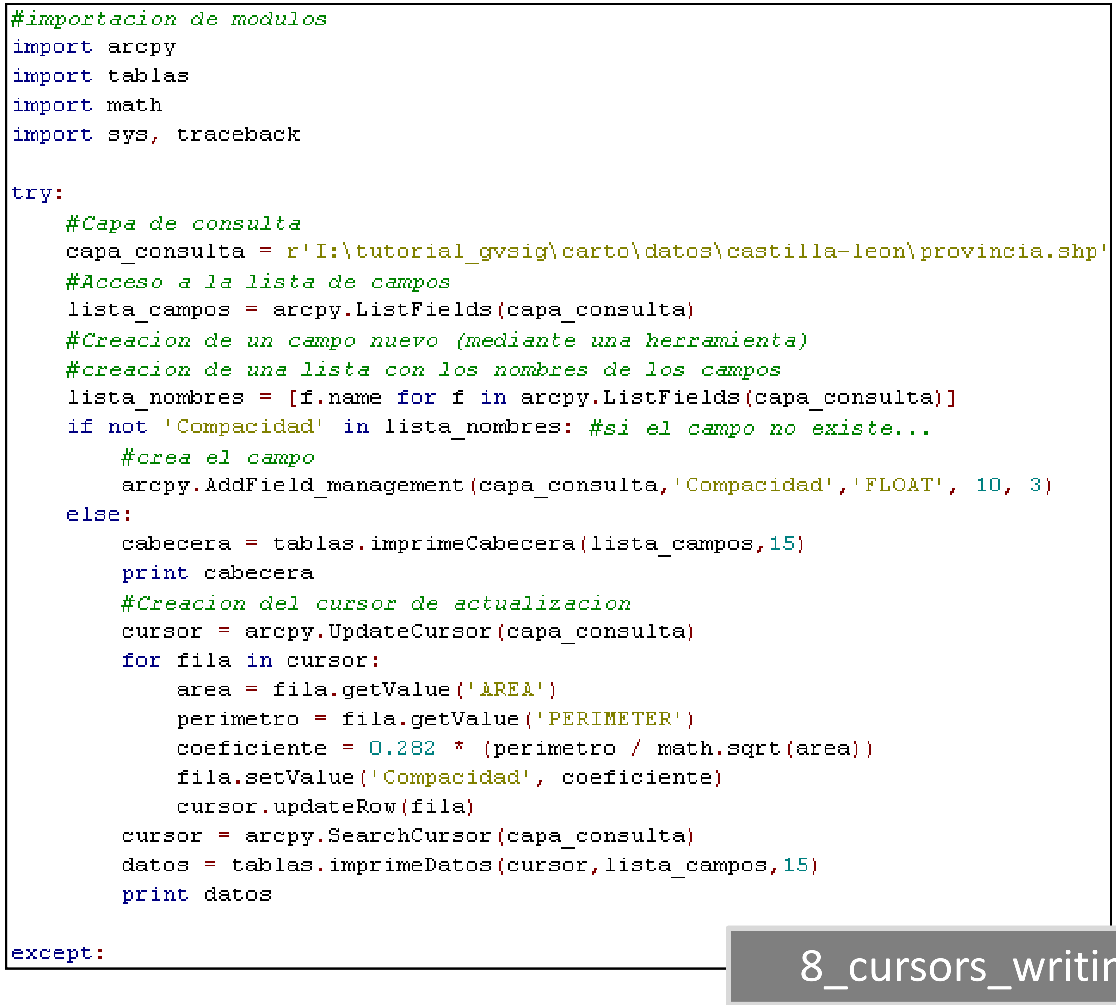
SearchCursor and field access

Create a csv file (using semicolon ";" as separator) from the data in the table of the previous exercise. This way, we are able to export our data to other software.

# UpdateCursor

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018v00000064000000/

# UpdateCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})



```
#importacion de modulos
import arcpy
import tablas
import math
import sys, traceback

try:
    #Capa de consulta
    capa_consulta = r'I:\tutorial_gvsig\carto\datos\castilla-leon\provincia.shp'
    #Acceso a la lista de campos
    lista_campos = arcpy.ListFields(capa_consulta)
    #Creacion de un campo nuevo (mediante una herramienta)
    #creacion de una lista con los nombres de los campos
    lista_nombres = [f.name for f in arcpy.ListFields(capa_consulta)]
    if not 'Compacidad' in lista_nombres: #si el campo no existe...
        #crea el campo
        arcpy.AddField_management(capa_consulta,'Compacidad','FLOAT', 10, 3)
    else:
        cabecera = tablas.imprimeCabecera(lista_campos,15)
        print cabecera
        #Creacion del cursor de actualizacion
        cursor = arcpy.UpdateCursor(capa_consulta)
        for fila in cursor:
            area = fila.getValue('AREA')
            perimetro = fila.getValue('PERIMETER')
            coeficiente = 0.282 * (perimetro / math.sqrt(area))
            fila.setValue('Compacidad', coeficiente)
            cursor.updateRow(fila)
        cursor = arcpy.SearchCursor(capa_consulta)
        datos = tablas.imprimeDatos(cursor,lista_campos,15)
        print datos

except:
```

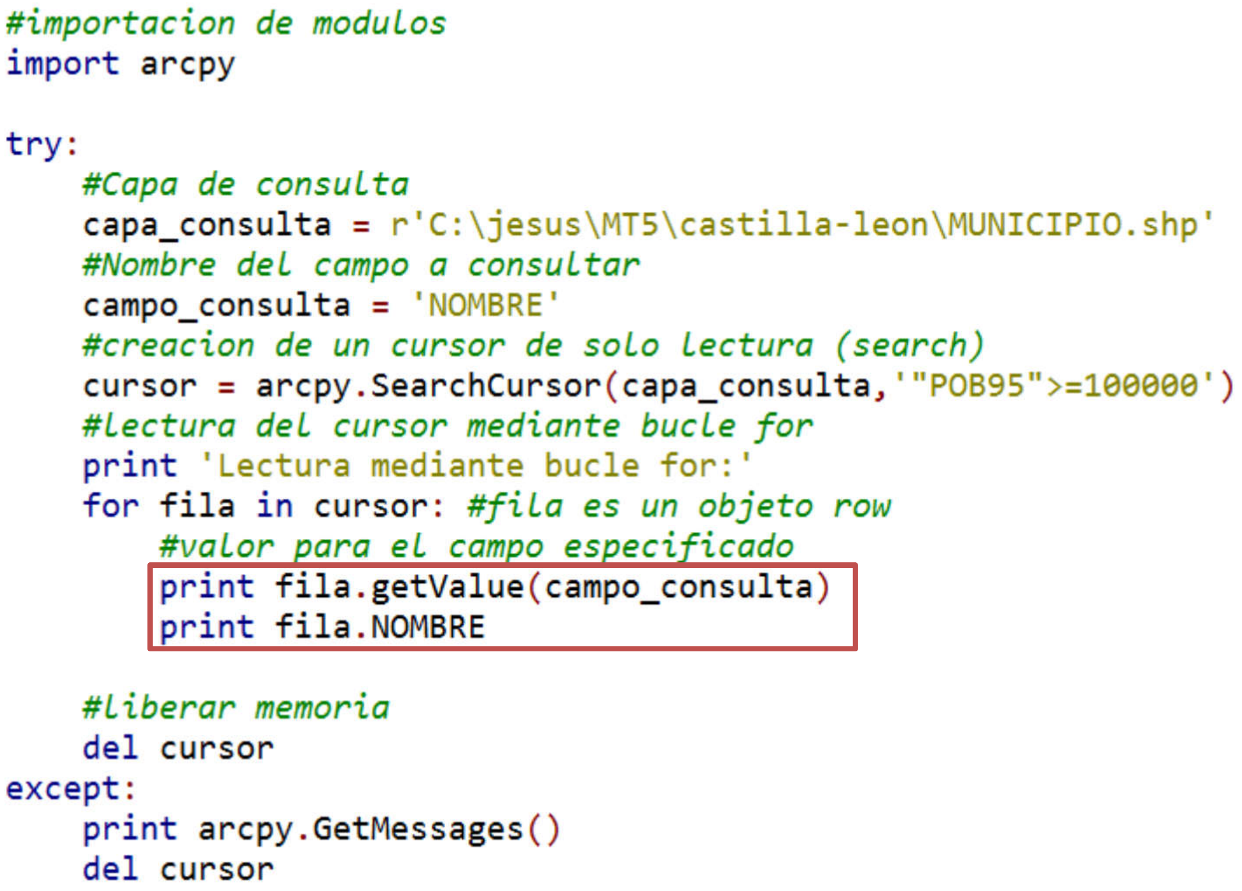
8_cursors_writing.py

UpdateCursor

Using the previous code as a basis, add a new field to the table "Municipios" and compute the population density in the year 1995 (inhabitants per square kilometer)

Peculiarities of the cursors

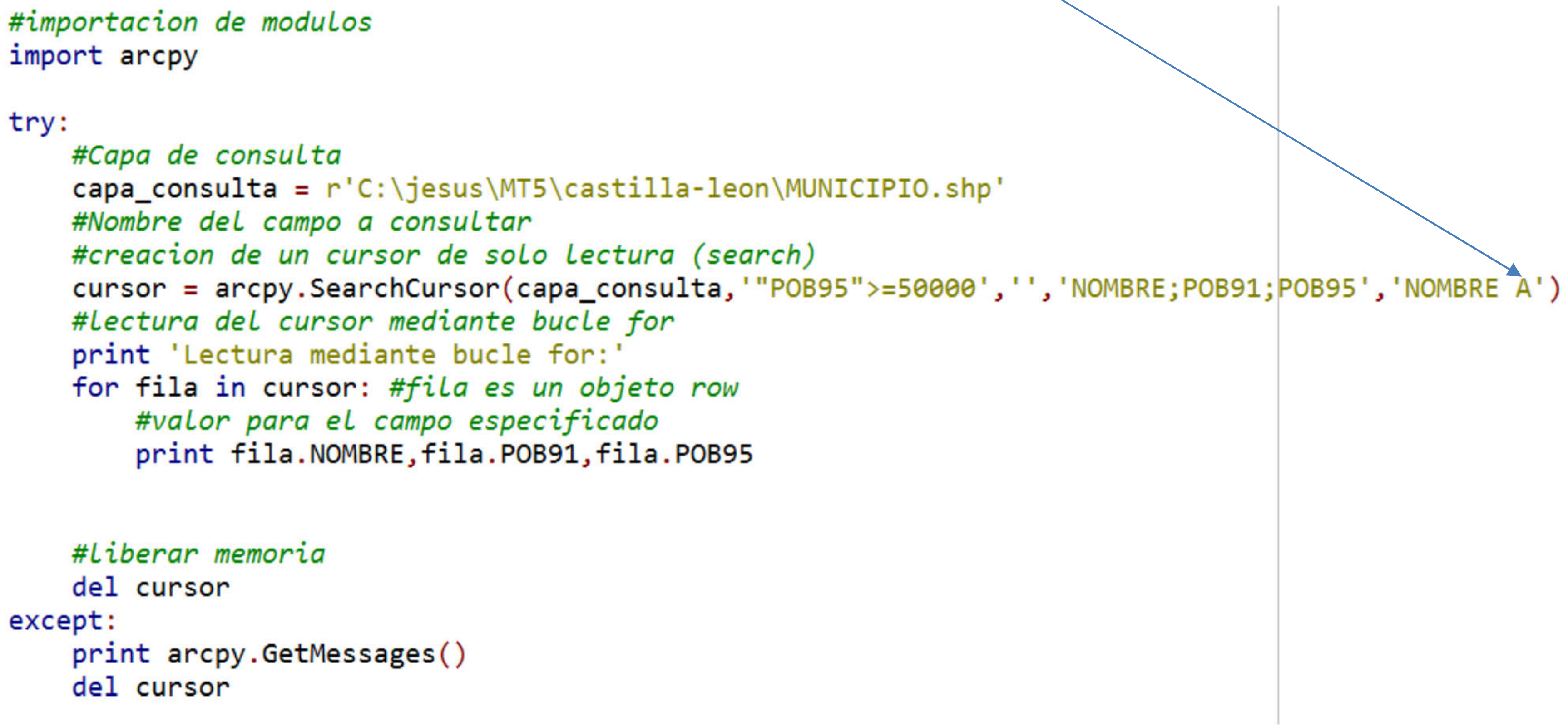
# To get the value of a field we can use two techniques: using the getValue function or deal the name of the field as a property (names without white spaces).



```
#importacion de modulos
import arcpy

try:
    #Capa de consulta
    capa_consulta = r'C:\jesus\MT5\castilla-leon\MUNICIPIO.shp'
    #Nombre del campo a consultar
    campo_consulta = 'NOMBRE'
    #creacion de un cursor de solo lectura (search)
    cursor = arcpy.SearchCursor(capa_consulta,'"POB95">=100000')
    #lectura del cursor mediante bucle for
    print 'Lectura mediante bucle for:'
    for fila in cursor: #fila es un objeto row
        #valor para el campo especificado
        print fila.getValue(campo_consulta)
        print fila.NOMBRE

    #liberar memoria
    del cursor
except:
    print arcpy.GetMessages()
    del cursor
```

Peculiarities of the cursors

# The search cursor has several useful modifiers: for example, {fields} only returns a subset of fields and ({sort_fields} along with "A" ascending or "D" descending"}) let us sort the rows of the cursor by one or more fields.



```
#importacion de modulos
import arcpy

try:
    #Capa de consulta
    capa_consulta = r'C:\jesus\MT5\castilla-leon\MUNICIPIO.shp'
    #Nombre del campo a consultar
    #creacion de un cursor de solo lectura (search)
    cursor = arcpy.SearchCursor(capa_consulta,'"POB95">=50000','','NOMBRE;POB91;POB95','NOMBRE A')
    #lectura del cursor mediante bucle for
    print 'Lectura mediante bucle for:'
    for fila in cursor: #fila es un objeto row
        #valor para el campo especificado
        print fila.NOMBRE,fila.POB91,fila.POB95


    #liberar memoria
    del cursor
except:
    print arcpy.GetMessages()
    del cursor
```

Peculiarities of the cursors

# There are other module that provides more powerful tools in order to extract information from data sources. This module is called arcpy.da (arcpy data access). In this module, cursors have higher performance and there are several specialized functions, like edition control, domain and subtypes operations and conversion between tables or feature classes and numpy arrays.

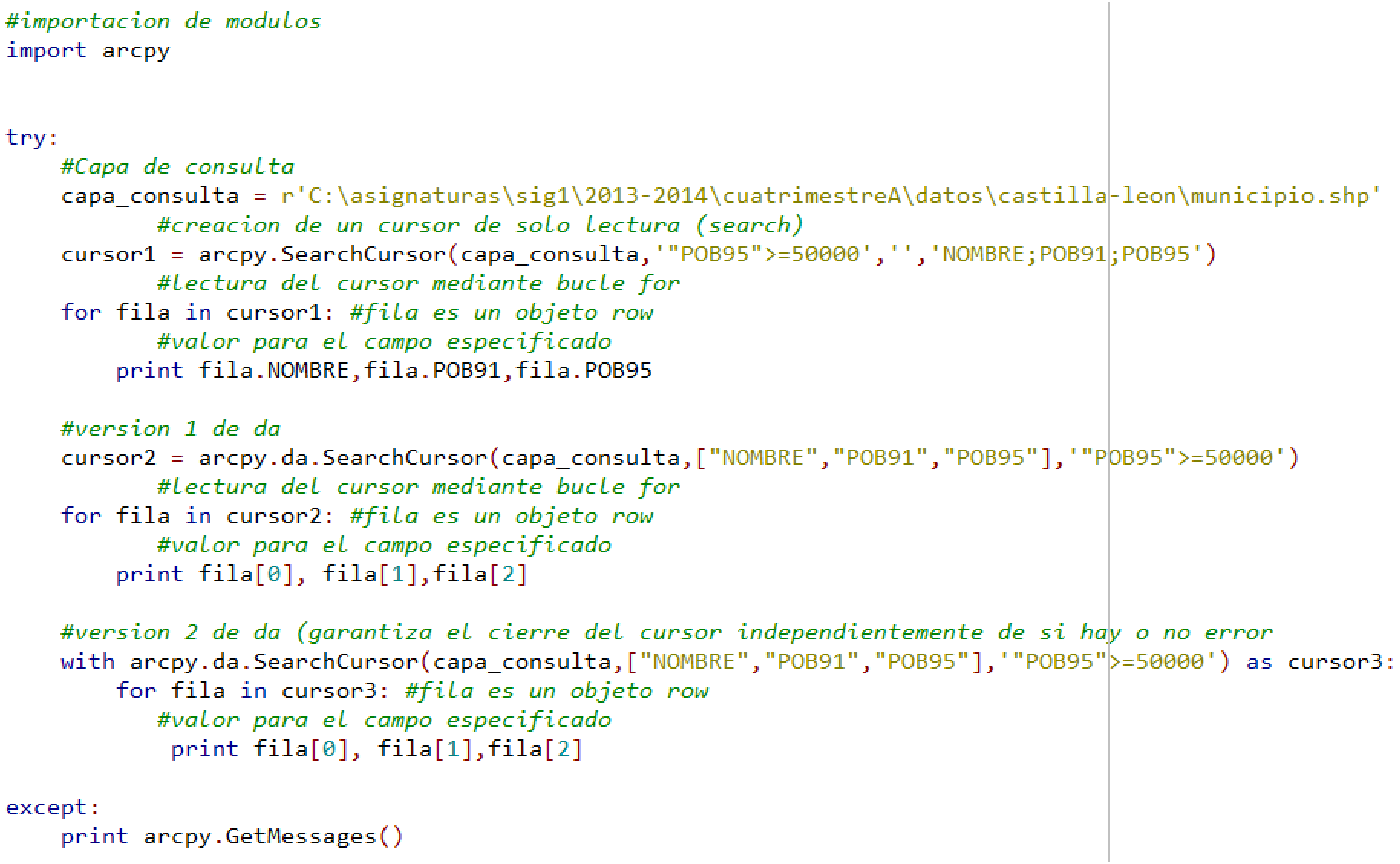
# Differences:

arcpy:
SearchCursor (dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})

arcpy.da
SearchCursor (in_table, field_names, {where_clause}, {spatial_reference}, {explode_to_points}, {sql_clause})

Only geodatabases

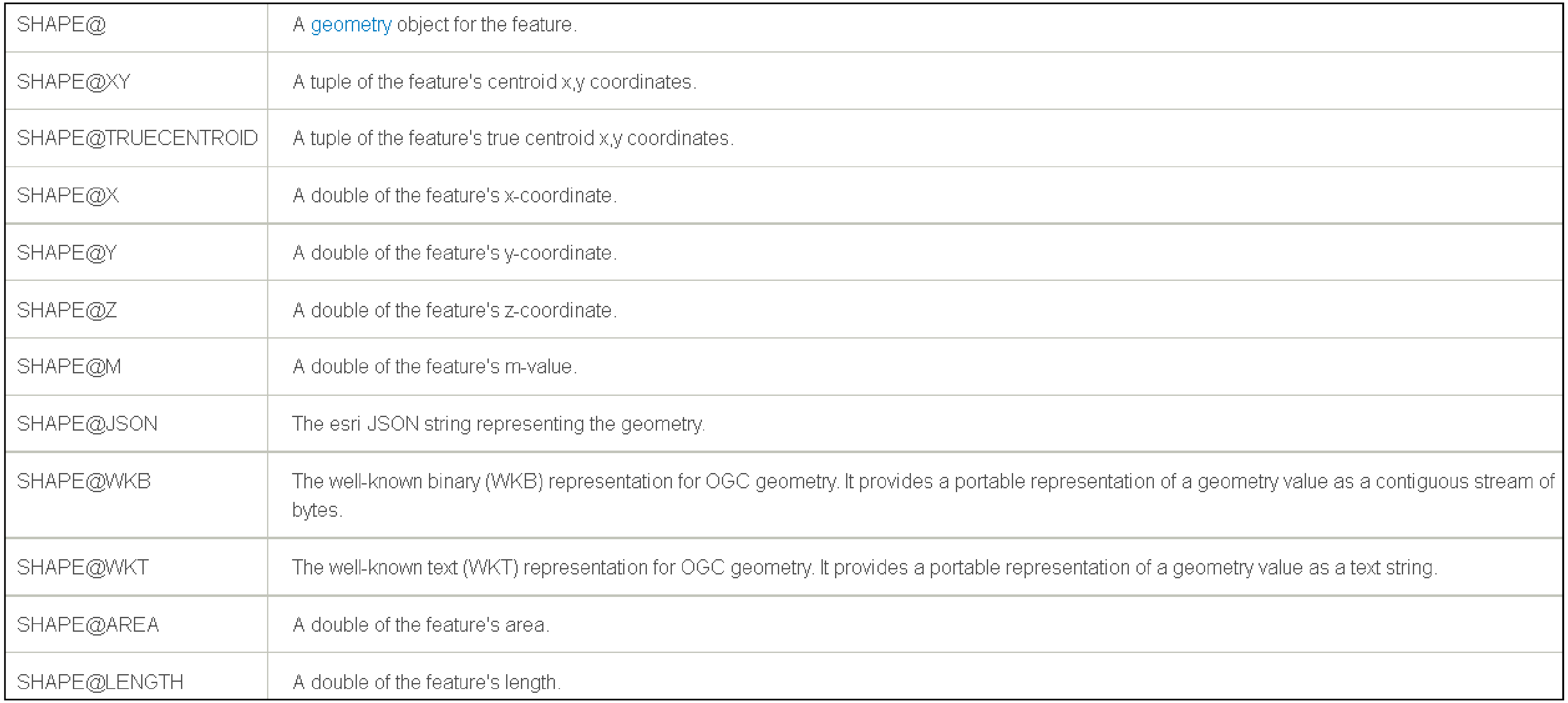
Peculiarities of the cursors



```
#importacion de modulos
import arcpy


try:
    #Capa de consulta
    capa_consulta = r'C:\asignaturas\sig1\2013-2014\cuatrimestreA\datos\castilla-leon\municipio.shp'
           #creacion de un cursor de solo lectura (search)
    cursor1 = arcpy.SearchCursor(capa_consulta,'"POB95">=50000','','NOMBRE;POB91;POB95')
           #lectura del cursor mediante bucle for
    for fila in cursor1: #fila es un objeto row
           #valor para el campo especificado
        print fila.NOMBRE,fila.POB91,fila.POB95

    #version 1 de da
    cursor2 = arcpy.da.SearchCursor(capa_consulta,["NOMBRE","POB91","POB95"],'"POB95">=50000')
           #lectura del cursor mediante bucle for
    for fila in cursor2: #fila es un objeto row
           #valor para el campo especificado
        print fila[0], fila[1],fila[2]

    #version 2 de da (garantiza el cierre del cursor independientemente de si hay o no error
    with arcpy.da.SearchCursor(capa_consulta,["NOMBRE","POB91","POB95"],'"POB95">=50000') as cursor3:
        for fila in cursor3: #fila es un objeto row
           #valor para el campo especificado
            print fila[0], fila[1],fila[2]

except:
    print arcpy.GetMessages()
```

Peculiarities of the cursors

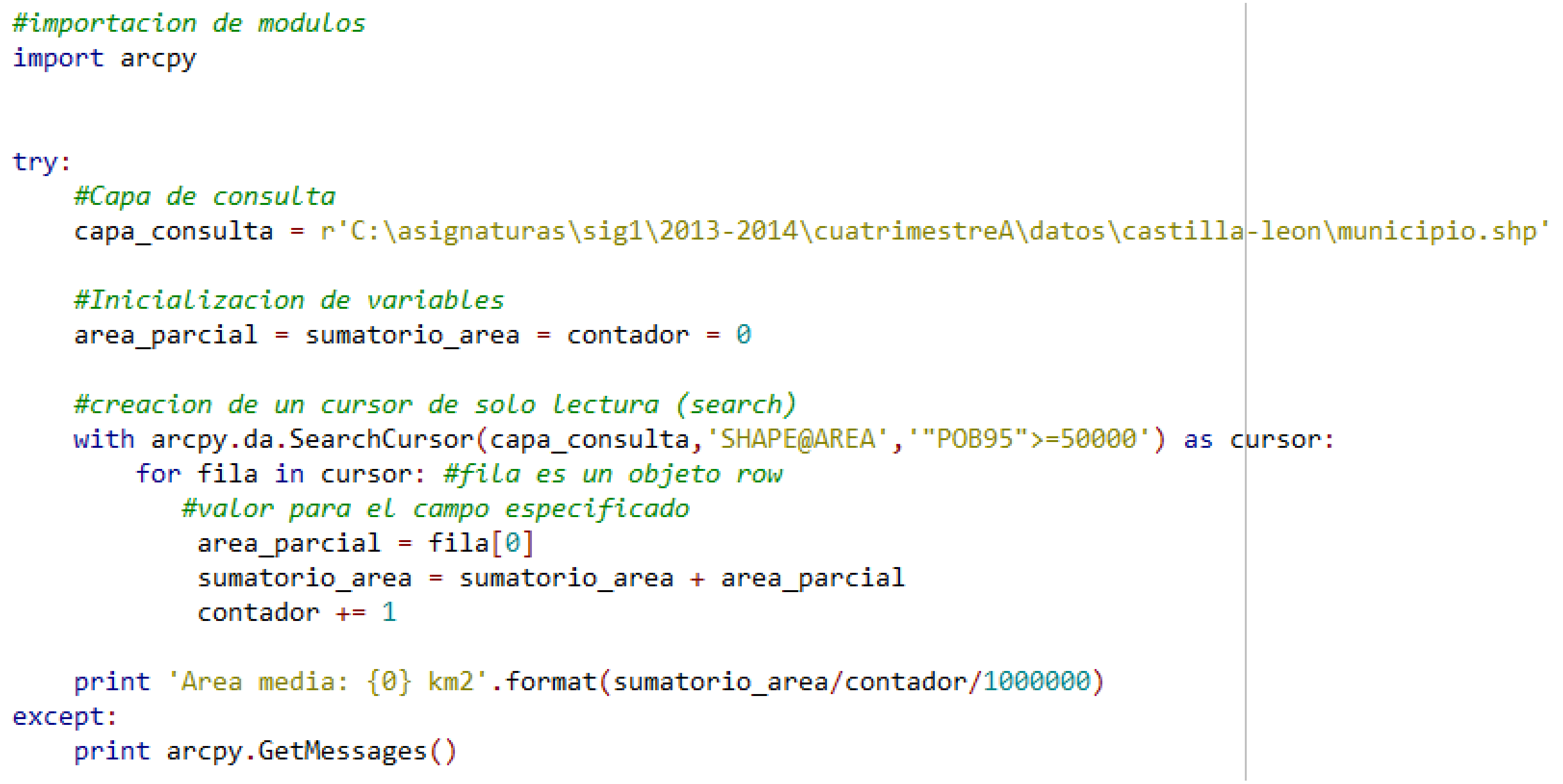
# One important advantage of arcpy.da is that we can get geometric information without process any geometry previously (slower process). We can achieve this using names of special fields (tokens).



SHAPE@	A geometry object for the feature.
SHAPE@XY	A tuple of the feature's centroid x,y coordinates.
SHAPE@TRUECENTROID	A tuple of the feature's true centroid x,y coordinates.
SHAPE@X	A double of the feature's x-coordinate.
SHAPE@Y	A double of the feature's y-coordinate.
SHAPE@Z	A double of the feature's z-coordinate.
SHAPE@M	A double of the feature's m-value.
SHAPE@JSON	The esri JSON string representing the geometry.
SHAPE@WKB	The well-known binary (WKB) representation for OGC geometry. It provides a portable representation of a geometry value as a contiguous stream of bytes.
SHAPE@WKT	The well-known text (WKT) representation for OGC geometry. It provides a portable representation of a geometry value as a text string.
SHAPE@AREA	A double of the feature's area.
SHAPE@LENGTH	A double of the feature's length.

Peculiarities of the cursors

# One important advantage of arcpy.da is that we can get geometric information without process any geometry previously (slower process). We can achieve this using names of special fields (tokens).



```
#importacion de modulos
import arcpy


try:
    #Capa de consulta
    capa_consulta = r'C:\asignaturas\sig1\2013-2014\cuatrimestreA\datos\castilla-leon\municipio.shp'

    #Inicializacion de variables
    area_parcial = sumatorio_area = contador = 0

    #creacion de un cursor de solo lectura (search)
    with arcpy.da.SearchCursor(capa_consulta,'SHAPE@AREA','"POB95">=50000') as cursor:
        for fila in cursor: #fila es un objeto row
            #valor para el campo especificado
            area_parcial = fila[0]
            sumatorio_area = sumatorio_area + area_parcial
            contador += 1

    print 'Area media: {0} km2'.format(sumatorio_area/contador/1000000)
except:
    print arcpy.GetMessages()
```

# InsertCursor

http://resources.arcgis.com/en/help/main/10.1/index.html#//018v0000002z000000

# This cursor allows to insert new rows into a table.

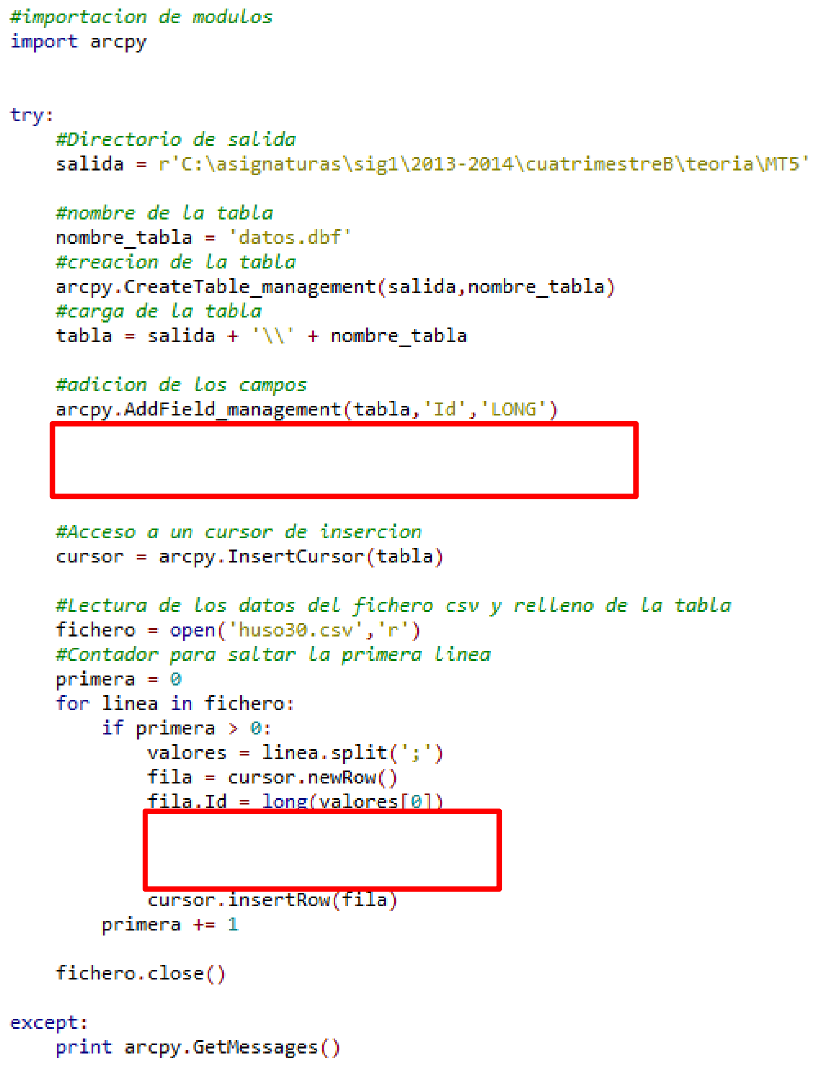
# InsertCursor (dataset, {spatial_reference})

# To insert a new row we must follow the next steps:

- make a new insert cursor
- create a new empty row (using the method newRow())
- populate the different fields using a suitable value
- insert the row into the cursor

```
cursor = arcpy.InsertCursor(table)
row = cursor.newRow()
row.field_name = value
cursor.insertCursor(row)
```

InsertCursor



```
#importacion de modulos
import arcpy


try:
    #Directorio de salida
    salida = r'C:\asignaturas\sig1\2013-2014\cuatrimestreB\teoria\MT5'

    #nombre de la tabla
    nombre_tabla = 'datos.dbf'
    #creacion de la tabla
    arcpy.CreateTable_management(salida,nombre_tabla)
    #carga de la tabla
    tabla = salida + '\\' + nombre_tabla

    #adicion de los campos
    arcpy.AddField_management(tabla,'Id','LONG')



    #Acceso a un cursor de insercion
    cursor = arcpy.InsertCursor(tabla)

    #Lectura de los datos del fichero csv y relleno de la tabla
    fichero = open('huso30.csv','r')
    #Contador para saltar la primera linea
    primera = 0
    for linea in fichero:
        if primera > 0:
            valores = linea.split(';')
            fila = cursor.newRow()
            fila.Id = long(valores[0])



            cursor.insertRow(fila)
        primera += 1

    fichero.close()

except:
    print arcpy.GetMessages()
```

This script makes a dbf table from a csv file. Fill in the gaps with a suitable code.

InsertCursor

Write a second version of the script, but this time, using the arcpy.da module

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018w0000000t000000/

Geometries

http://resources.arcgis.com/en/help/main/10.1/index.html#/na/018z00000070000000/

# Arcpy module is able to handle several types of geometric objects: Geometry, Multipoint, Point, PointGeometry, Polygon and Polyline.

# Sometimes it is necessary to manipulate geometries, either to get its properties or to modify its coordinates. We can do that using cursors.

# The row object has a property call "shape". This property returns a reference to a geometry object.

```
cursor = arcpy.SearchCursor(input_layer)
for row in cursor:
    polygon = row.shape
```

# There is another way to do the same thing. The arcpy.da module provides shortcuts to get geometric properties in a faster an more efficient form.

```
cursor = arcpy.da.SearchCursor(input_layer,'SHAPE@AREA')
for row in cursor:
   area = row[0]
```

Geometries

# There are a big amount of topological operators that can be very useful.

- Boundary ()
- Buffer (distance)
- Clip (extent)
- Contains (second geometry)
- ConvexHull ()
- Crosses (second geometry)
- Difference (second geometry)
- Disjoint (second geometry)
- DistanceTo (second geometry)
- Equals (second geometry)
- getArea (type)
- getLength (measure type)
- getPart ({index})
- Intersect (second geometry)
- Overlaps (second geometry)
- PositionAlongLine (value,{percentage})
- ProjectAs(spatial reference, {transform name})
- SymmetricDifference (second geometry)
- Touches (second geometry)
- Union (second geometry)
- Within (second geometry)

# Points

# Points are a type of geometry that can be used to build other type of geometry, as lines or polygons.

# To build a new point you can use the method "Point()" that is into arcpy module: arcpy.Point({X}, {Y}, {Z}, {M}, {ID})

# Examples:

```
#empty point
point = arcpy.Point()
```

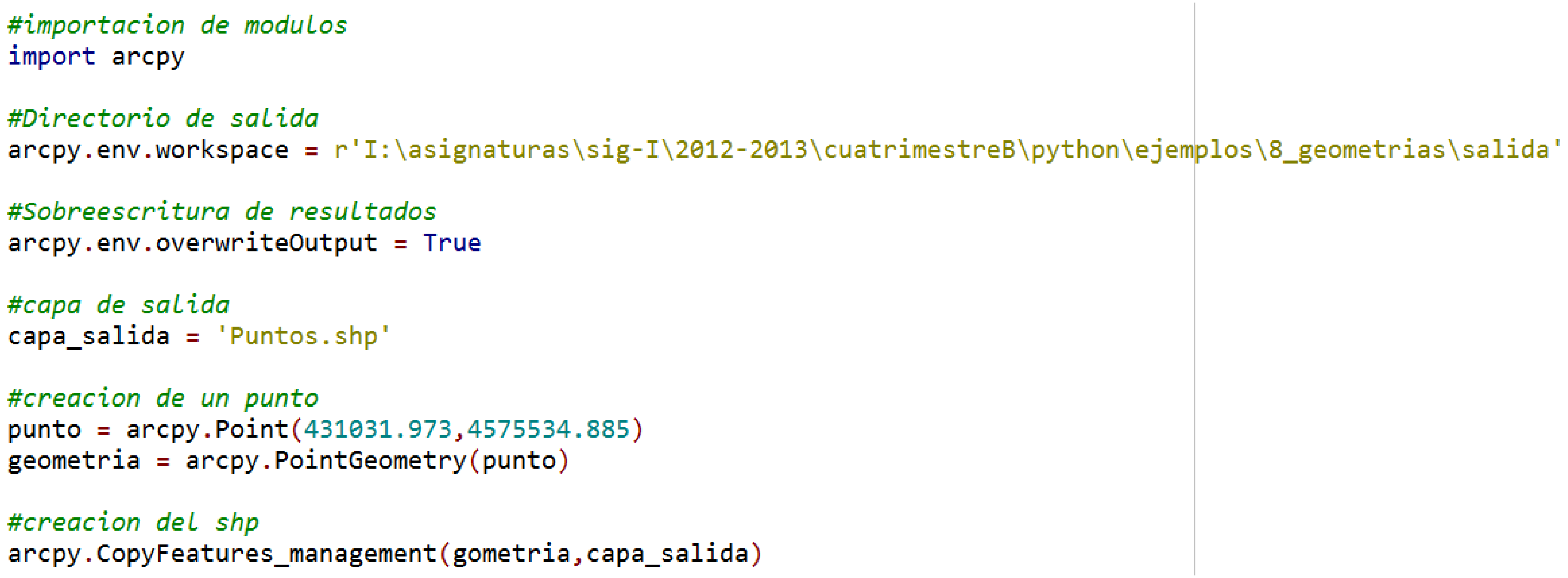
```
#Point from a pair of coordinates
point = arcpy.Point(431031.973,4575534.885)
```

# Geometries can be used as input parameters for a ArcToolBox tool. In such a case, the parameter can be a simple geometry or an array of geometries.

```
point = arcpy.Point(431031.973,4575534.885)
geometry = arcpy.PointGeometry(point) #Point must become into geometryPoint before
arcpy.CopyFeatures_management(geometry,output_layer)
```

9_make_point.py

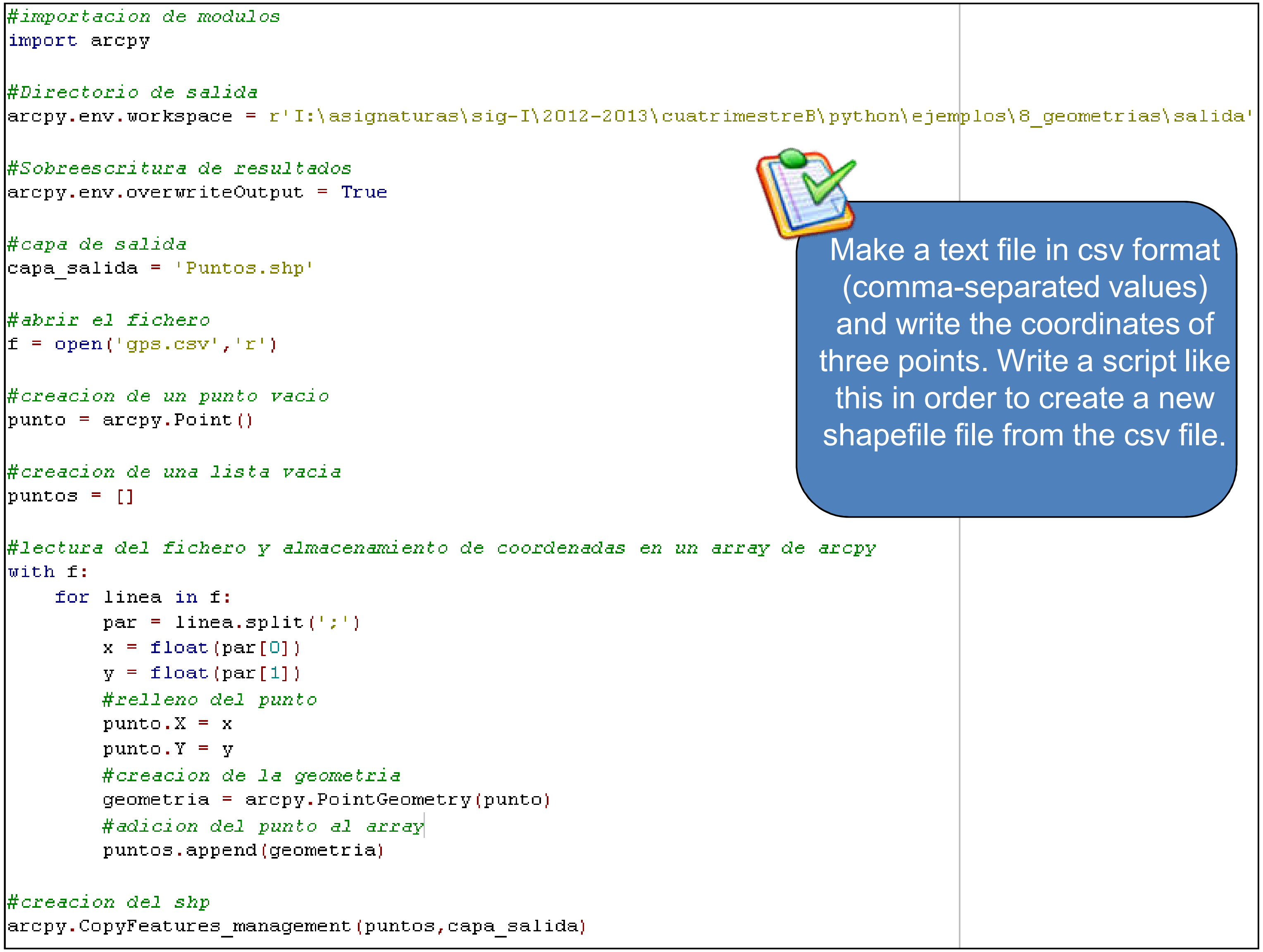
Points



```
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#capa de salida
capa_salida = 'Puntos.shp'

#creacion de un punto
punto = arcpy.Point(431031.973,4575534.885)
geometria = arcpy.PointGeometry(punto)

#creacion del shp
arcpy.CopyFeatures_management(gometria,capa_salida)
```

9_make_point.py

Points



```
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#capa de salida
capa_salida = 'Puntos.shp'

#abrir el fichero
f = open('gps.csv','r')

#creacion de un punto vacio
punto = arcpy.Point()

#creacion de una lista vacia
puntos = []

#lectura del fichero y almacenamiento de coordenadas en un array de arcpy
with f:
    for linea in f:
        par = linea.split(';')
        x = float(par[0])
        y = float(par[1])
        #relleno del punto
        punto.X = x
        punto.Y = y
        #creacion de la geometria
        geometria = arcpy.PointGeometry(punto)
        #adicion del punto al array
        puntos.append(geometria)

#creacion del shp
arcpy.CopyFeatures_management(puntos,capa_salida)
```

Make a text file in csv format (comma-separated values) and write the coordinates of three points. Write a script like this in order to create a new shapefile file from the csv file.

# Polyline

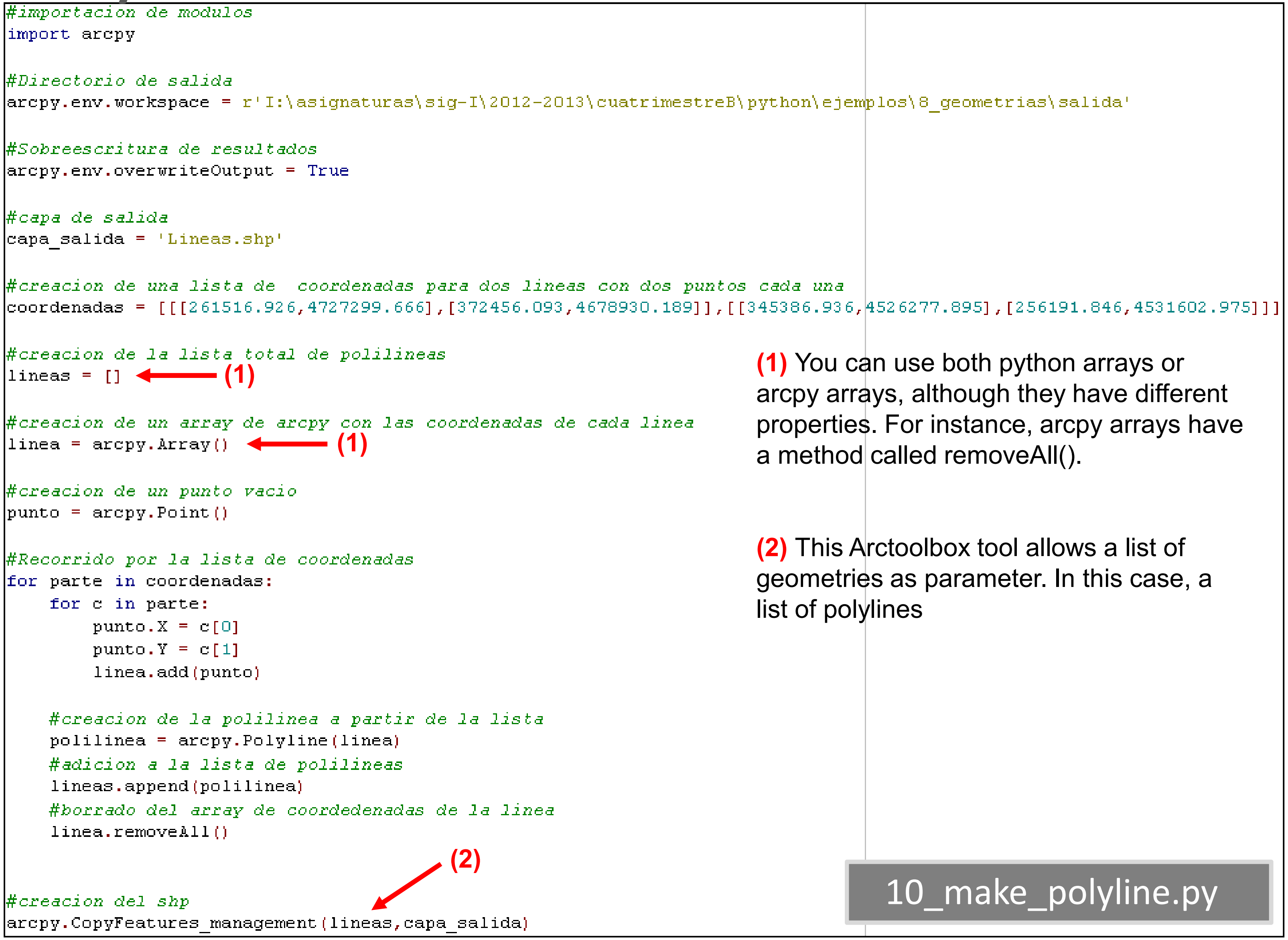
# Polylines are built from a list of points.

# To build a new polyline you can use the method "Polyline()" that is into arcpy module: arcpy.Polyline (inputs, {spatial_reference}, {has_z}, {has_m})

# There are also a big amount of topological operators and properties that can be very useful.

- Area
- Centroid
- Extent
- FirstPoint
- HullRectangle
- IsMultipart
- LabelPoint
- LastPoint
- Length
- Length3D
- PartCount
- PointCount
- TrueCentroid
- Type

Polyline



```python
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#capa de salida
capa_salida = 'Lineas.shp'

#creacion de una lista de  coordenadas para dos lineas con dos puntos cada una
coordenadas = [[[261516.926,4727299.666],[372456.093,4678930.189]],[[345386.936,4526277.895],[256191.846,4531602.975]]]

#creacion de la lista total de polilineas
lineas = []

#creacion de un array de arcpy con las coordenadas de cada linea
linea = arcpy.Array()

#creacion de un punto vacio
punto = arcpy.Point()

#Recorrido por la lista de coordenadas
for parte in coordenadas:
    for c in parte:
        punto.X = c[0]
        punto.Y = c[1]
        linea.add(punto)

    #creacion de la polilinea a partir de la lista
    polilinea = arcpy.Polyline(linea)
    #adicion a la lista de polilineas
    lineas.append(polilinea)
    #borrado del array de coordedenadas de la linea
    linea.removeAll()


#creacion del shp
arcpy.CopyFeatures_management(lineas,capa_salida)
```

(1)

(1)

(2)

(1) You can use both python arrays or arcpy arrays, although they have different properties. For instance, arcpy arrays have a method called removeAll().

(2) This Arctoolbox tool allows a list of geometries as parameter. In this case, a list of polylines

10_make_polyline.py

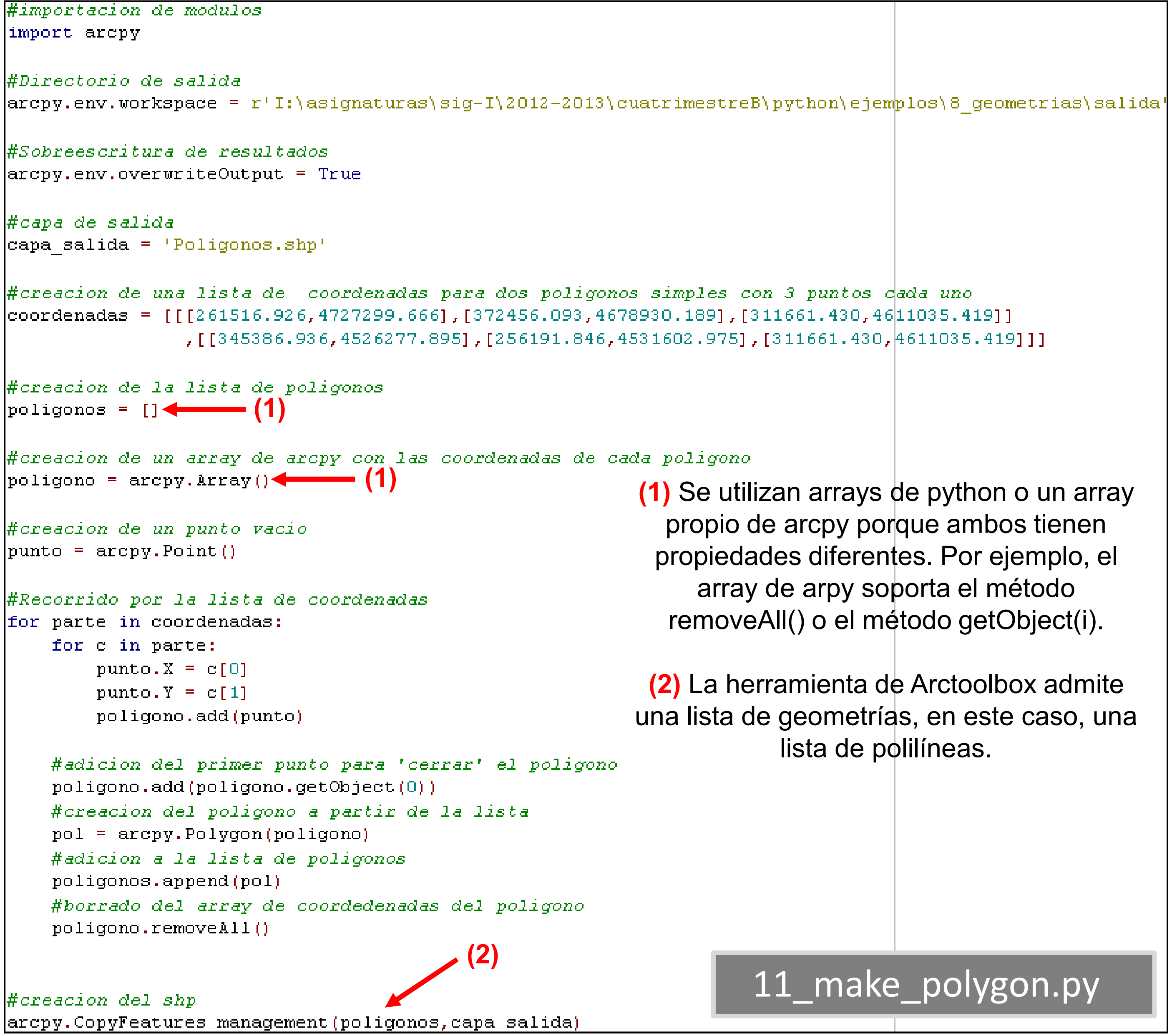
# Polygons

# Polygons have a structure similar to polylines, but the first point is duplicated and inserted at the end of the list, in order to close the polygon.

# They can be simple or complex, with one or more rings (polygons with holes). To access each ring we must count its parts (partCount() and getPart()).

# To build a new polygon you can use the method "Polygon()" that is into arcpy module: arcpy.Polygon (inputs, {spatial_reference}, {has_z}, {has_m})

Polygons



```
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#capa de salida
capa_salida = 'Poligonos.shp'

#creacion de una lista de  coordenadas para dos poligonos simples con 3 puntos cada uno
coordenadas = [[[261516.926,4727299.666],[372456.093,4678930.189],[311661.430,4611035.419]]
                ,[[345386.936,4526277.895],[256191.846,4531602.975],[311661.430,4611035.419]]]

#creacion de la lista de poligonos
poligonos = []

#creacion de un array de arcpy con las coordenadas de cada poligono
poligono = arcpy.Array()

#creacion de un punto vacio
punto = arcpy.Point()

#Recorrido por la lista de coordenadas
for parte in coordenadas:
    for c in parte:
        punto.X = c[0]
        punto.Y = c[1]
        poligono.add(punto)

    #adicion del primer punto para 'cerrar' el poligono
    poligono.add(poligono.getObject(0))
    #creacion del poligono a partir de la lista
    pol = arcpy.Polygon(poligono)
    #adicion a la lista de poligonos
    poligonos.append(pol)
    #borrado del array de coordedenadas del poligono
    poligono.removeAll()


#creacion del shp
arcpy.CopyFeatures_management(poligonos,capa_salida)
```

(1) Se utilizan arrays de python o un array propio de arcpy porque ambos tienen propiedades diferentes. Por ejemplo, el array de arpy soporta el método removeAll() o el método getObject(i).

(2) La herramienta de Arctoolbox admite una lista de geometrías, en este caso, una lista de polilíneas.

11_make_polygon.py

Accessing to geometries

# You can use two ways: cursors from arcpy or cursors from arcpy.da

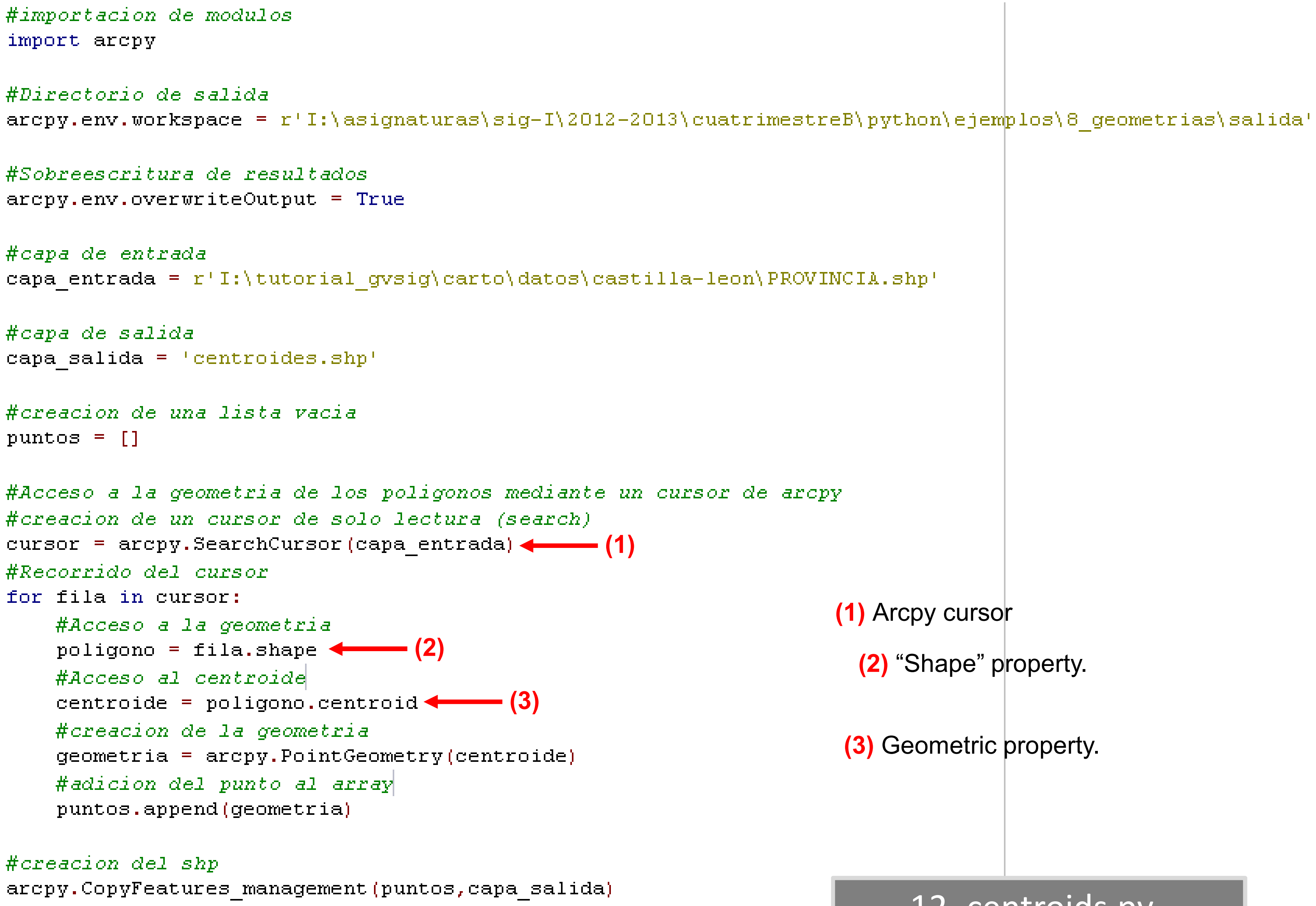
# In the first case, once a cursor has been built, we will use the property "shape", that belongs to the class "row".

```
cursor = arcpy.SearchCursor(input_layer)
for row in cursor:
      polygon = row.shape
```

# In the second case, we will use a combination between cursors from arcpy.da module and tokens (faster and more efficient).

```
cursor = arcpy.da.SearchCursor(input_layer,'SHAPE@AREA')
for row in cursor:
   area = row[0]
```

Centroids computation



```
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#capa de entrada
capa_entrada = r'I:\tutorial_gvsig\carto\datos\castilla-leon\PROVINCIA.shp'

#capa de salida
capa_salida = 'centroides.shp'

#creacion de una lista vacia
puntos = []

#Acceso a la geometria de los poligonos mediante un cursor de arcpy
#creacion de un cursor de solo lectura (search)
cursor = arcpy.SearchCursor(capa_entrada)
#Recorrido del cursor
for fila in cursor:
    #Acceso a la geometria
    poligono = fila.shape
    #Acceso al centroide
    centroide = poligono.centroid
    #creacion de la geometria
    geometria = arcpy.PointGeometry(centroide)
    #adicion del punto al array
    puntos.append(geometria)

#creacion del shp
arcpy.CopyFeatures_management(puntos,capa_salida)
```

(1) Arcpy cursor

(2) “Shape” property.

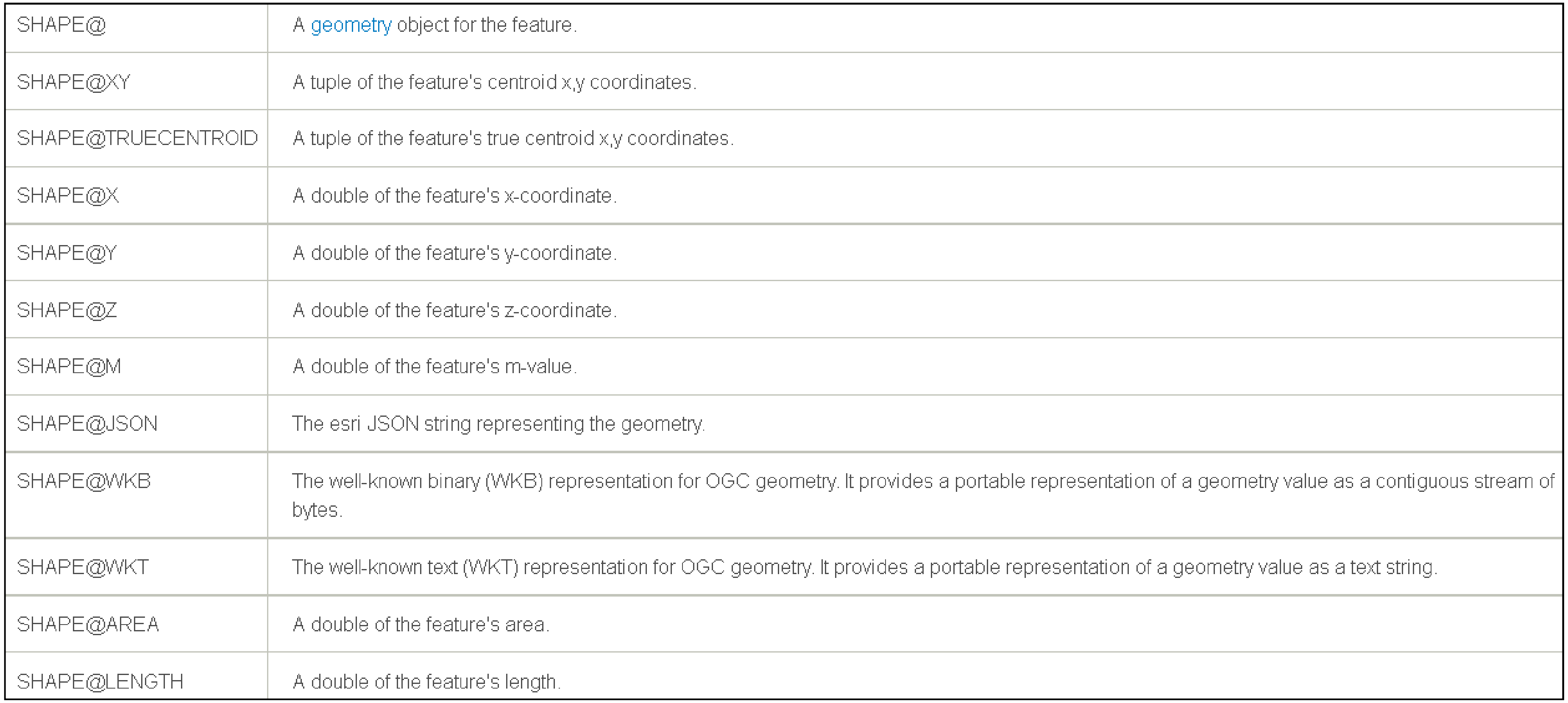
(3) Geometric property.

12_centroids.py

Tokens

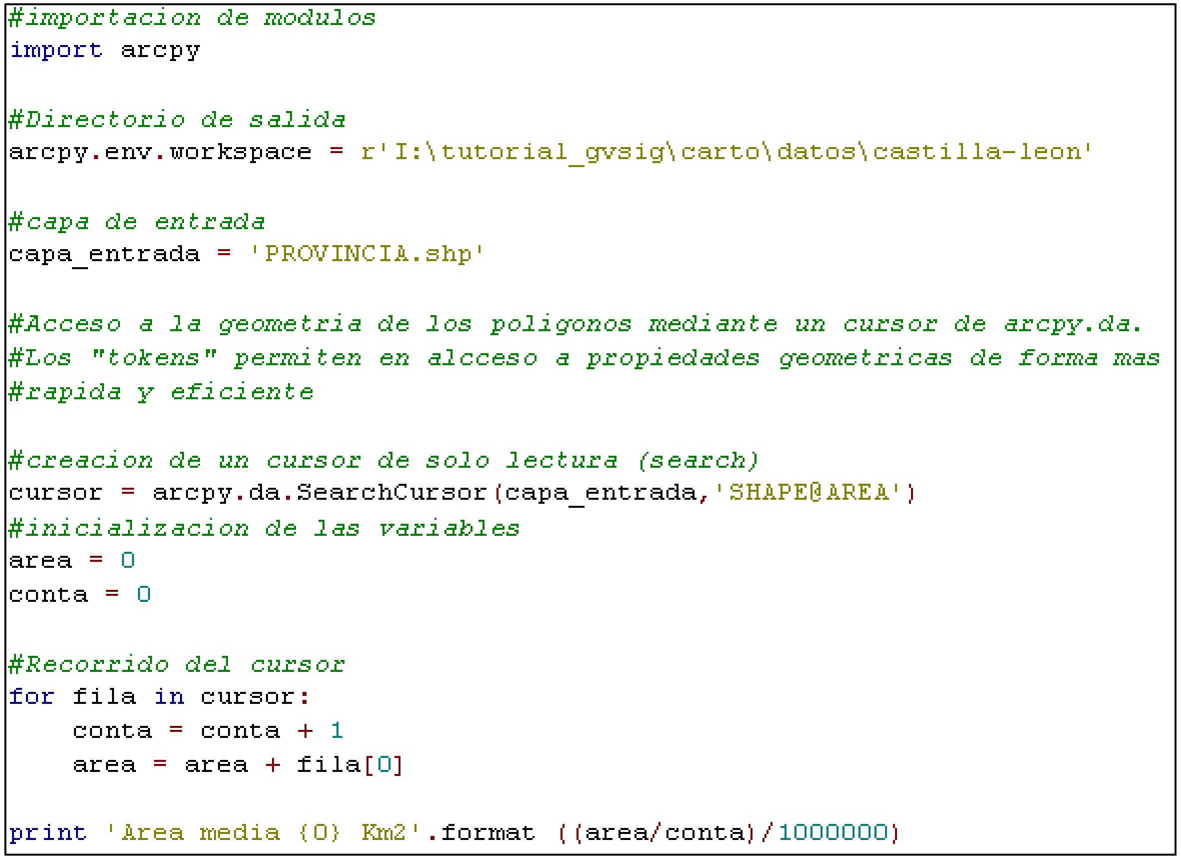
http://resources.arcgis.com/en/help/main/10.1/index.html#//002z0000001t000000

# They are shortcuts to geometric properties. This method is faster than get firstly a geometry and then extract its properties.



SHAPE@	A geometry object for the feature.
SHAPE@XY	A tuple of the feature's centroid x,y coordinates.
SHAPE@TRUECENTROID	A tuple of the feature's true centroid x,y coordinates.
SHAPE@X	A double of the feature's x-coordinate.
SHAPE@Y	A double of the feature's y-coordinate.
SHAPE@Z	A double of the feature's z-coordinate.
SHAPE@M	A double of the feature's m-value.
SHAPE@JSON	The esri JSON string representing the geometry.
SHAPE@WKB	The well-known binary (WKB) representation for OGC geometry. It provides a portable representation of a geometry value as a contiguous stream of bytes.
SHAPE@WKT	The well-known text (WKT) representation for OGC geometry. It provides a portable representation of a geometry value as a text string.
SHAPE@AREA	A double of the feature's area.
SHAPE@LENGTH	A double of the feature's length.

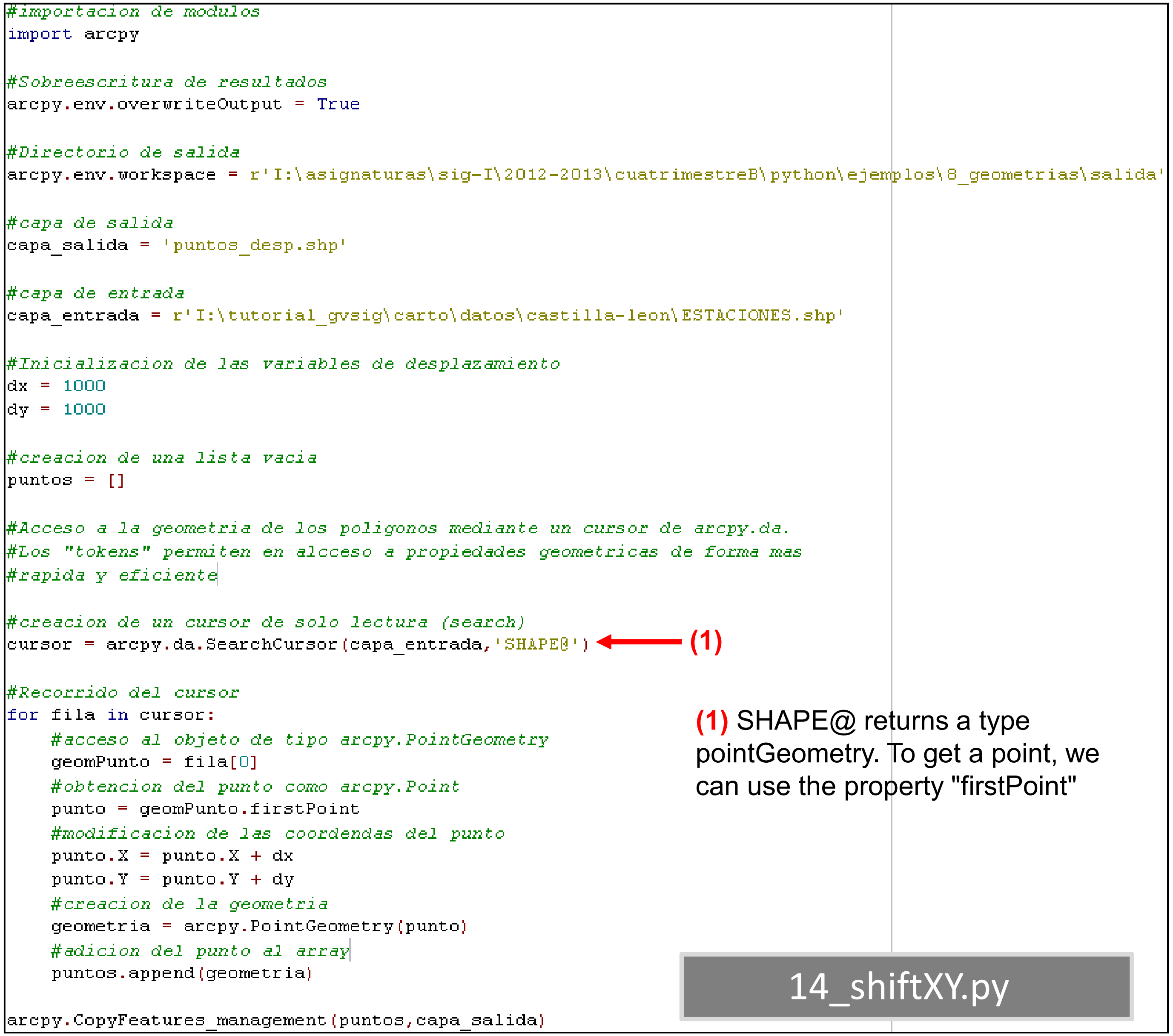
Tokens. Average area computation



```
#importacion de modulos
import arcpy

#Directorio de salida
arcpy.env.workspace = r'I:\tutorial_gvsig\carto\datos\castilla-leon'

#capa de entrada
capa_entrada = 'PROVINCIA.shp'

#Acceso a la geometria de los poligonos mediante un cursor de arcpy.da.
#Los "tokens" permiten en alcceso a propiedades geometricas de forma mas
#rapida y eficiente

#creacion de un cursor de solo lectura (search)
cursor = arcpy.da.SearchCursor(capa_entrada,'SHAPE@AREA')
#inicializacion de las variables
area = 0
conta = 0

#Recorrido del cursor
for fila in cursor:
    conta = conta + 1
    area = area + fila[0]

print 'Area media {0} Km2'.format ((area/conta)/1000000)
```

13_average_area.py

Tokens: shifting a layer



```
#importacion de modulos
import arcpy

#Sobreescritura de resultados
arcpy.env.overwriteOutput = True

#Directorio de salida
arcpy.env.workspace = r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\python\ejemplos\8_geometrias\salida'

#capa de salida
capa_salida = 'puntos_desp.shp'

#capa de entrada
capa_entrada = r'I:\tutorial_gvsig\carto\datos\castilla-leon\ESTACIONES.shp'

#Inicializacion de las variables de desplazamiento
dx = 1000
dy = 1000

#creacion de una lista vacia
puntos = []

#Acceso a la geometria de los poligonos mediante un cursor de arcpy.da.
#Los "tokens" permiten en alcceso a propiedades geometricas de forma mas
#rapida y eficiente

#creacion de un cursor de solo lectura (search)
cursor = arcpy.da.SearchCursor(capa_entrada,'SHAPE@')

#Recorrido del cursor
for fila in cursor:
    #acceso al objeto de tipo arcpy.PointGeometry
    geomPunto = fila[0]
    #obtencion del punto como arcpy.Point
    punto = geomPunto.firstPoint
    #modificacion de las coordendas del punto
    punto.X = punto.X + dx
    punto.Y = punto.Y + dy
    #creacion de la geometria
    geometria = arcpy.PointGeometry(punto)
    #adicion del punto al array
    puntos.append(geometria)

arcpy.CopyFeatures_management(puntos,capa_salida)
```

(1)

(1) SHAPE@ returns a type pointGeometry. To get a point, we can use the property "firstPoint"

14_shiftXY.py

Tokens

http://resources.arcgis.com/en/help/main/10.1/index.html#//002z0000001t000000

Review the exercise "5_centroids.py". Make a new version, but this time using some tokens

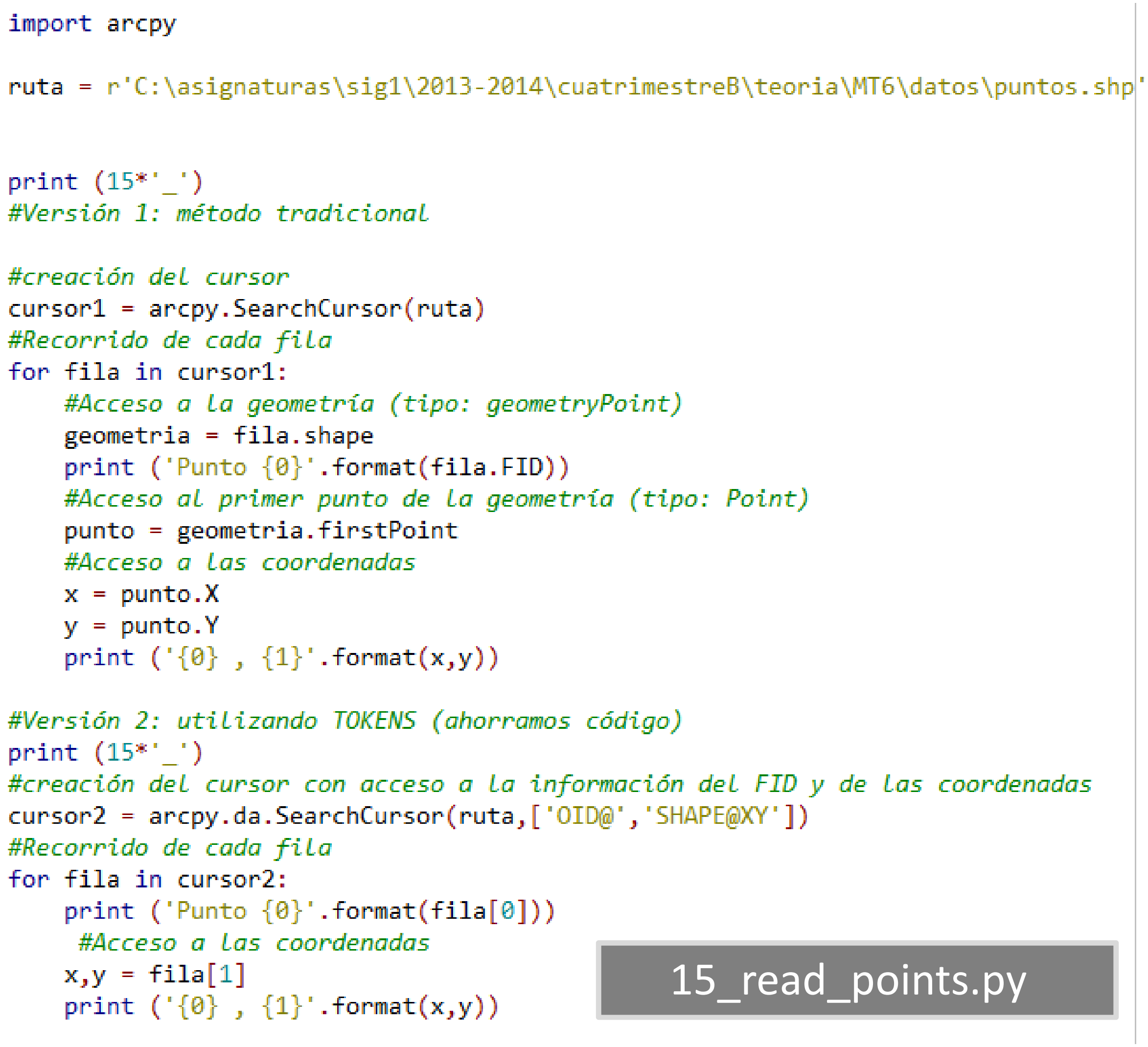
Reading geometries

# So far, we know how to read a geometry. Now we will to learn how to get its coordinates, no matter if we deal with points, polylines or polygons.

# We can use both normal cursors or cursors along with tokens.

# In the event of polylines or polygons with parts, we will have to iterate through each part in order to extract its coordinates

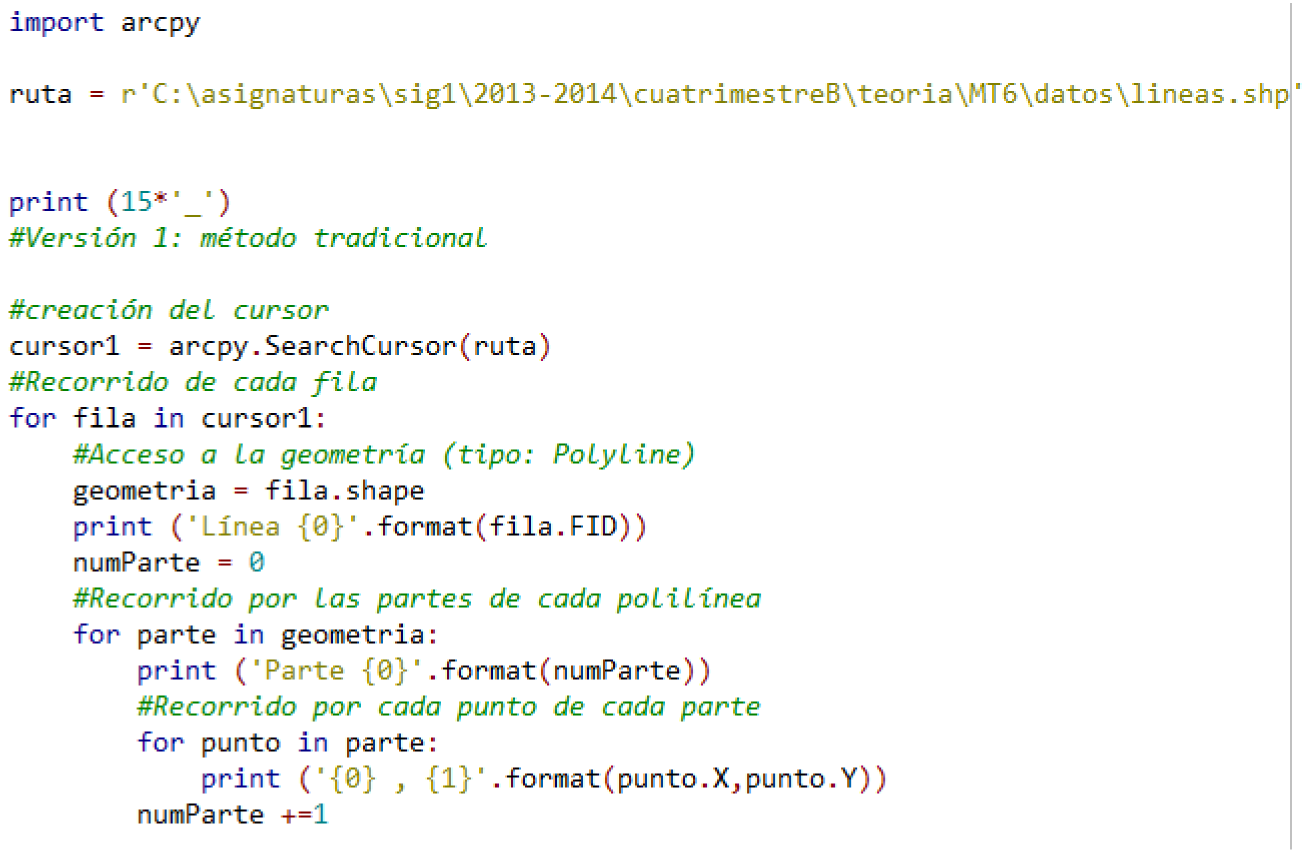
Reading points



```
import arcpy

ruta = r'C:\asignaturas\sig1\2013-2014\cuatrimestreB\teoria\MT6\datos\puntos.shp'


print (15*'_')
#Versión 1: método tradicional

#creación del cursor
cursor1 = arcpy.SearchCursor(ruta)
#Recorrido de cada fila
for fila in cursor1:
    #Acceso a la geometría (tipo: geometryPoint)
    geometria = fila.shape
    print ('Punto {0}'.format(fila.FID))
    #Acceso al primer punto de la geometría (tipo: Point)
    punto = geometria.firstPoint
    #Acceso a las coordenadas
    x = punto.X
    y = punto.Y
    print ('{0} , {1}'.format(x,y))

#Versión 2: utilizando TOKENS (ahorramos código)
print (15*'_')
#creación del cursor con acceso a la información del FID y de las coordenadas
cursor2 = arcpy.da.SearchCursor(ruta,['OID@','SHAPE@XY'])
#Recorrido de cada fila
for fila in cursor2:
    print ('Punto {0}'.format(fila[0]))
     #Acceso a las coordenadas
    x,y = fila[1]
    print ('{0} , {1}'.format(x,y))
```

15_read_points.py

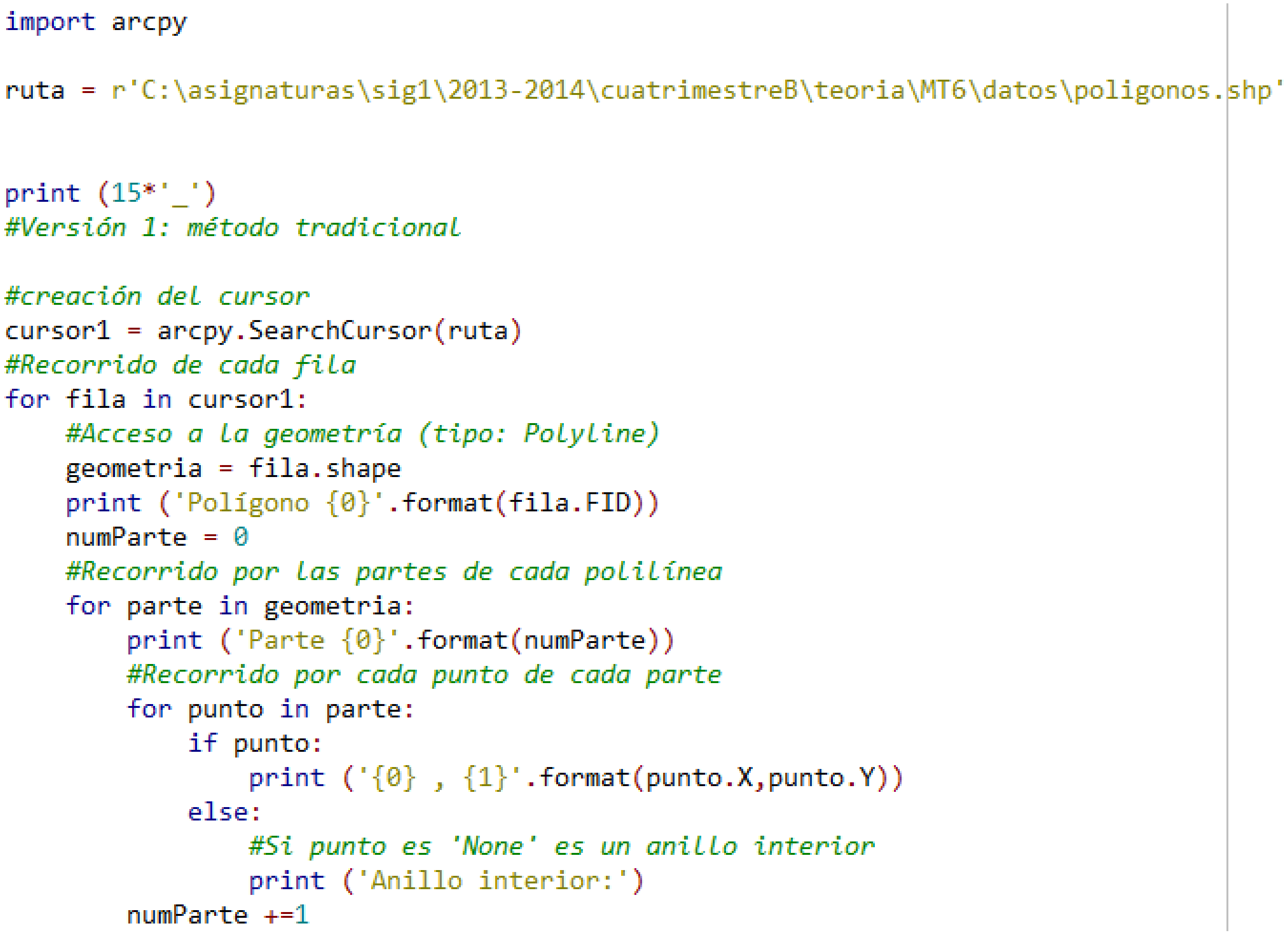
Reading polylines



```
import arcpy

ruta = r'C:\asignaturas\sig1\2013-2014\cuatrimestreB\teoria\MT6\datos\lineas.shp'


print (15*'_')
#Versión 1: método tradicional

#creación del cursor
cursor1 = arcpy.SearchCursor(ruta)
#Recorrido de cada fila
for fila in cursor1:
    #Acceso a la geometría (tipo: Polyline)
    geometria = fila.shape
    print ('Línea {0}'.format(fila.FID))
    numParte = 0
    #Recorrido por las partes de cada polilínea
    for parte in geometria:
        print ('Parte {0}'.format(numParte))
        #Recorrido por cada punto de cada parte
        for punto in parte:
            print ('{0} , {1}'.format(punto.X,punto.Y))
        numParte +=1
```

16_read_polylines.py

Reading polygons



```
import arcpy

ruta = r'C:\asignaturas\sig1\2013-2014\cuatrimestreB\teoria\MT6\datos\poligonos.shp'


print (15*'_')
#Versión 1: método tradicional

#creación del cursor
cursor1 = arcpy.SearchCursor(ruta)
#Recorrido de cada fila
for fila in cursor1:
    #Acceso a la geometría (tipo: Polyline)
    geometria = fila.shape
    print ('Polígono {0}'.format(fila.FID))
    numParte = 0
    #Recorrido por las partes de cada polilínea
    for parte in geometria:
        print ('Parte {0}'.format(numParte))
        #Recorrido por cada punto de cada parte
        for punto in parte:
            if punto:
                print ('{0} , {1}'.format(punto.X,punto.Y))
            else:
                #Si punto es 'None' es un anillo interior
                print ('Anillo interior:')
        numParte +=1
```

17_read_polygons.py