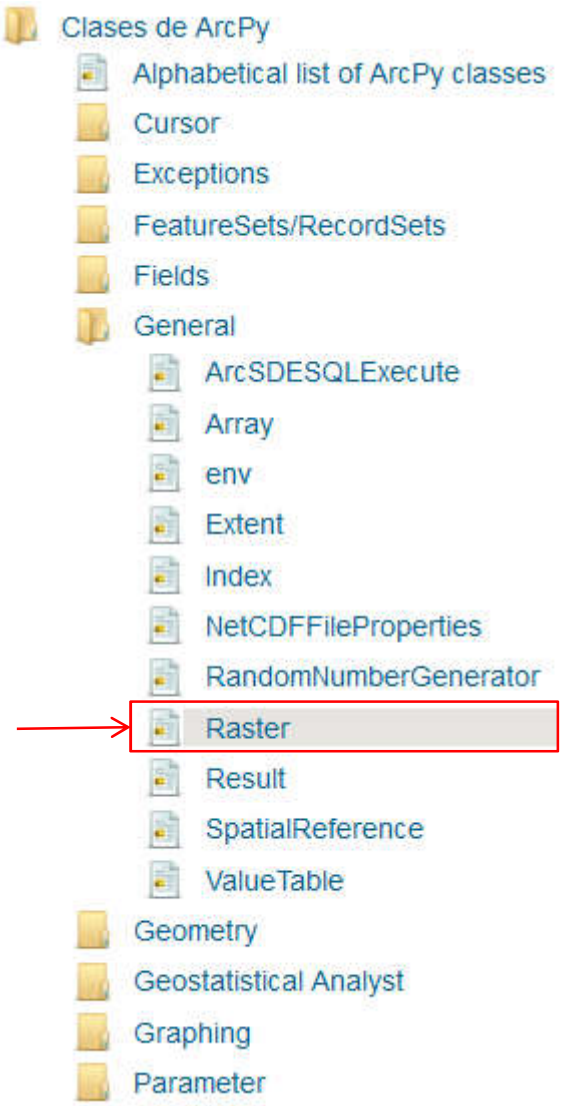
python
ArcGIS

Raster layers

Access to raster layers

Raster layers usually are used as both input or output parameters in map algebra. These outputs will be always temporary unless you save this output as permanent object to the hard drive. To access a raster layer, arcpy has a class called "Raster". This class has methods and properties that allow us to save this raster or to know some characteristics like bands number, cell size, etc.



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

Access to raster layers

#In arcpy, the starting point to manage raster layers is the class "Raster". One object of this class represents a raster layer loaded in memory. From that point on, we are able to access its properties:

```
import arcpy
raster = arcpy.Raster(r'C:\data\elevation')
name = raster.name
```

#The parameter of the "Raster" method can be either a path to the whole raster or a path to a specific band:

```
band1 = arcpy.Raster(r'C:\data\image.tif\Band_1')
```

#In the event we get "None" values from some of the statistical properties (maximum, mean, etc.), that means that it is necessary to run firstly the tool "CalculateStatistics_management". This tool will make an .xml file with all this information.

- bandCount
- catalogPath
- compressionType
- extent
- format
- hasRAT
- height
- isInteger
- isTemporary
- maximum
- mean
- meanCellHeight
- meanCellWidth
- minimum
- name
- noDataValue
- path
- pixelType
- spatialReference
- standardDeviation
- uncompressedSize
- width

Access to raster layers

```
#importing module
import arcpy

#raster to array conversion
#C:\asignaturas\sig1\2013-2014\cuatrimestreA\datos\sagunto\sagtm1.tif
#Also : C:\asignaturas\sig1\2013-2014\cuatrimestreA\datos\sagunto\sagtm1.tif\Band_1
#C:\asignaturas\sig1\2013-2014\cuatrimestreB\teoria\MT9\elevgrd (grid's have not extension)
raster = arcpy.Raster(r'E:\Asignaturas\DAS\2016-2017\datos\sagunto\sagtm1.tif')
#Getting raster properties
extent = raster.extent
name = raster.name
format_type = raster.format
number_of_bands = raster.bandCount
spatial_reference = raster.spatialReference
pixel_type = raster.pixelType
cell_size = raster.meanCellHeight
number_of_pixels_x = raster.width
number_of_pixels_y = raster.height
#In some cases, statistical values can be "None".
#So you have to run firstly the tool CalculateStatistics_management
minimum = raster.minimum
maximum = raster.maximum
mean = raster.mean
if minimum == None:
    arcpy.CalculateStatistics_management(raster)
    minimum = raster.minimum
    maximum = raster.maximum
    mean = raster.mean
#Equivalent: mean = arcpy.GetRasterProperties_management(raster,'MEAN')
#Output
print('Name: ' + name)
print ("XMin:" +str(extent.XMin)+" ; YMin: "+str(extent.YMin))
print('Format: ' + format_type)
print('Number of bands: '+ str(number_of_bands))
print('Spatial reference: '+ spatial_reference.name)
print('Type of data: ' + pixel_type)
print('Cell size: ' + str(cell_size))
print('Width: ' + str(number_of_pixels_x))
print('Heigh: ' + str(number_of_pixels_y))
print('Minimum value: ' + str(minimum))
print('Maximum value: ' + str(maximum))
print('Mean value: ' + str(mean))
```

1_access_to_raster.py

Pixel by pixel processing

#An efficient way to deal with raster data is convert a raster to a matrix. In this case, arcpy provides a method called "RasterToNumpyArray", that converts the raster data to a numpy matrix. Numpy is a python library that supports high-performance scientific computing. That way, we can apply filters or whatever other advanced operations using just a few lines of code.

RasterToNumPyArray (in_raster, {lower_left_corner}, {ncols}, {nrows}, {nodata_to_value})

```
#importing modules
import arcpy
import numpy as np

#Access to raster
raster = arcpy.Raster(r'E:\Asignaturas\DAS\2016-2017\datos\sagunto\sagtm1.tif')
#raster to numpy array conversion
raster_matrix = arcpy.RasterToNumPyArray(raster)
#array size
height,width = raster_matrix.shape
print height,width
#sum of all values in the matrix
matrix_sum = np.sum(raster_matrix)
#Computation of mean value in matrix
mean = matrix_sum/(float(height)*float(width))
#Output
print mean
```

2_raster_to_numpy.py

http://docs.scipy.org/doc/numpy/reference/arrays.html

Pixel by pixel processing

#Once we have a raster object as matrix, we can access to each value in order to read or write each pixel. If we need to save this matrix to a permanent image, firstly we have to convert again the matrix to a raster object. To do this, we can use the method "NumpyArrayToRaster" and then, the method "save".

NumPyArrayToRaster (in_array, {lower_left_corner}, {x_cell_size}, {y_cell_size}, {value_to_nodata})

Arcpy.Point type

```
#How to save a raster layer
#matrix to raster object conversion
lower_left_corner = arcpy.Point(0,0)
cell_size = 10
res = arcpy.NumPyArrayToRaster(matrix,lower_left_corner,cell_size,cell_size)
#Save the image
res.save(r'I:\asignaturas\sig-I\2012-2013\cuatrimestreB\teoria\MT9\salida\media')
```

Pixel by pixel processing

```
def equalize(matrix):
    ''' compute histogram equalization
        input and output: numpy array
    '''
    #compute the histogram
    image_histogram ,bins = np.histogram(raster_matrix.flatten(),256)
    #cumulative distribution function
    cdf = image_histogram.cumsum()
    cdf = 255*cdf/cdf[-1] #normalize
    #linear interpolation of cdf to find new pixel values
    equalization = np.interp(raster_matrix.flatten(),bins[:-1],cdf)
    new_matrix = equalization.reshape(raster_matrix.shape)
    return new_matrix
```

Use this function to perform a equalization operation over an image. Use RasterToNumpyArray and NumPyArrayToRaster and save the result as a new image.

Pixel by pixel processing

```python
#importing modules
import arcpy
#output overwrite
arcpy.env.overwriteOutput = True

#kernel components
kernel = [1,1,1,1,1,1,1,1,1]

#raster properties
raster = arcpy.Raster(r'E:\Asignaturas\DAS\2016-2017\datos\sagunto\sagtm1.tif')
size = raster.meanCellHeight
width = raster.width
height = raster.height
extent = raster.extent
esquina_ii = arcpy.Point(extent.XMin,extent.YMin)

#raster to array conversion
raster1 = arcpy.RasterToNumPyArray(raster)
raster2 = arcpy.RasterToNumPyArray(raster)
rows,columns = height,width
#filter application
for row in range(1,rows-1):
    for column in range(1,columns-1):
        value1 = raster1.item(row-1,column-1)* kernel[0]
        value2 = raster1.item(row-1,column)* kernel[1]
        value3 = raster1.item(row-1,column+1)* kernel[2]
        value4 = raster1.item(row,column-1)* kernel[3]
        value5 = raster1.item(row,column)* kernel[4]
        value6 = raster1.item(row,column+1)* kernel[5]
        value7 = raster1.item(row+1,column-1)* kernel[6]
        value8 = raster1.item(row+1,column)* kernel[7]
        value9 = raster1.item(row+1,column+1)* kernel[8]
        value = (value1+value2+value3+value4+value5+value6+value7+value8+value9)/9
        raster2[row,column] = value

#array to raster conversion
res = arcpy.NumPyArrayToRaster(raster2,esquina_ii,size,size)
#save the raster
res.save(r'E:\Asignaturas\DAS\2016-2017\datos\sagunto\sagtm1_m.tif')
```

3_raster_mean_filter.py

Spatial Analyst module

#Arcpy has the module "Spatial Analyst" (sa). This module provides all operators and functions to work with a raster as a block of data.

To work with a raster as a block of data, we can follow the same rules apply in map algebra.

http://resources.arcgis.com/en/help/main/10.1/index.html#//00p600000008000000

It is very important to remember that if we want to work with the module sa, firstly we must check if we have an enabled license of this module. This is due to the fact that the module sa depends on the Spatial Analyst Extension in ArcMap and and that these extensions are optionals.

http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//002z0000000z000000

Spatial Analyst module

```
#http://resources.arcgis.com/en/help/main/10.1/index.html#//00p600000003000000

#importing modules
import arcpy
from arcpy.sa import * #!!!WATCH OUT!!!

matrix = arcpy.Raster(r'E:\Asignaturas\DAS\2016-2017\datos\DEM\mde1.asc')
#We must check the extension license
#If the extension is available...
if arcpy.CheckExtension("Spatial") == "Available": #!!!OJO!!!
    #take a license
    arcpy.CheckOutExtension("Spatial")
    #Arithmetic calculus example (land relief exaggerated)
    res = matrix * 5
    #we apply two hillshade operations (map algebra expressions)
    hillshade1 = Hillshade(matrix,315,45)
    hillshade2 = Hillshade(res,315,45)
    #Save the outputs
    hillshade1.save(r'E:\Asignaturas\DAS\2016-2017\datos\output\hs1')
    hillshade2.save(r'E:\Asignaturas\DAS\2016-2017\datos\output\hs2')
    #release the license
    arcpy.CheckInExtension("Spatial")
else:
    print ('Spatial Analyst license not available')
```

4_raster_block_processing.py

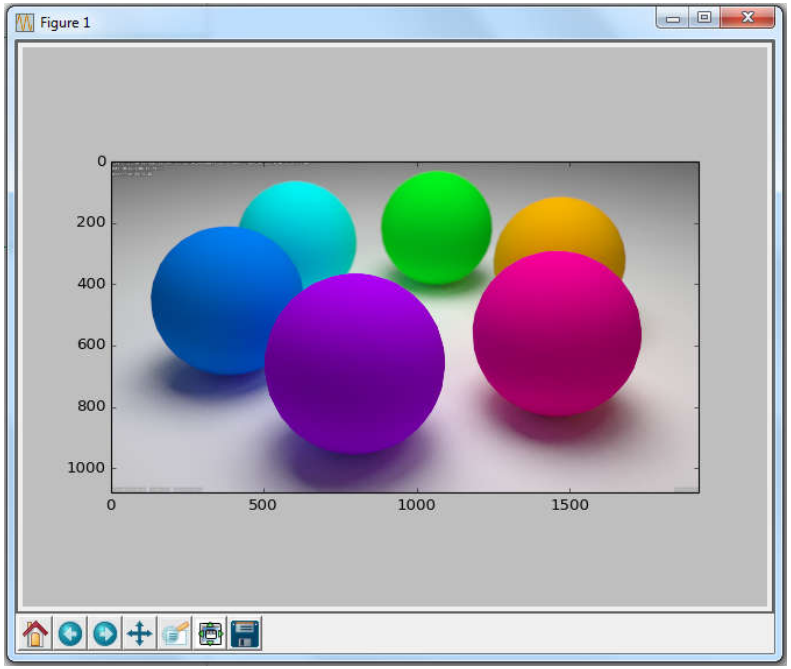
Additional libraries

#Some libraries can be very useful in specific fields (image processing, computer vision, etc.). In this example we are going to show haw to work with the libraries **OpenCV** and **matplotlib** in order to read and visualize an image.

https://opencv-python-tutroals.readthedocs.org/en/latest/
http://matplotlib.org/
http://www.lfd.uci.edu/~gohlke/pythonlibs/

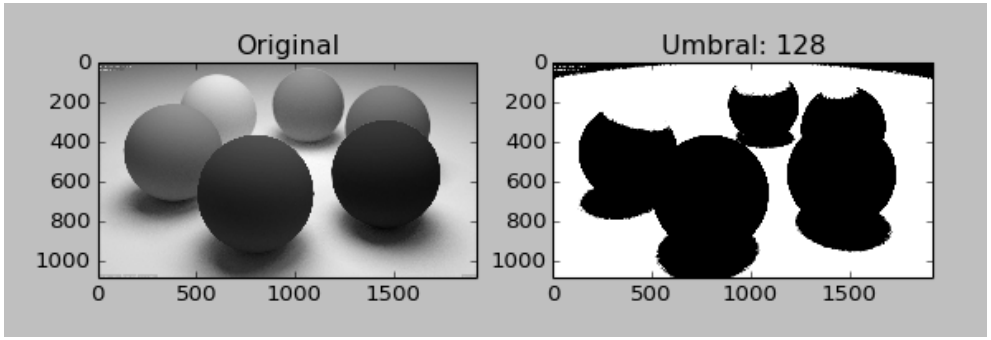
```
#importing modules
import cv2
from matplotlib import pyplot as plt

#open the image
img = cv2.imread('imagenes\\bolas.png',1)
#draw the image using matplotlib
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
#open a window to view the image
plt.show()
```

5_opencv_open_image.py



Additional libraries

#In this case, we show how to work with these libraries to perform a thresholding operation




```
import cv2
import numpy as np
from matplotlib import pyplot as plt

#open the image
path = 'imagenes\\bolas.png'
image = cv2.imread(path,0) #0: grayscale mode
#segmentation threshold
threshold = 128
#segmentation function with Opencv
vret,processed = cv2.threshold(image,threshold,255,cv2.THRESH_BINARY)
#matplotlib settings
plt.subplot(1,2,1) #two subplots (rows: 1; columns: 2)
plt.imshow(image,cmap = 'gray')
plt.title('Original')
plt.subplot(1,2,2)
plt.imshow(processed,cmap = 'gray')
plt.title('Threshold: {0}'.format(threshold))
#open a window to view the image
plt.show()
```

6_opencv_image_segmentation.py