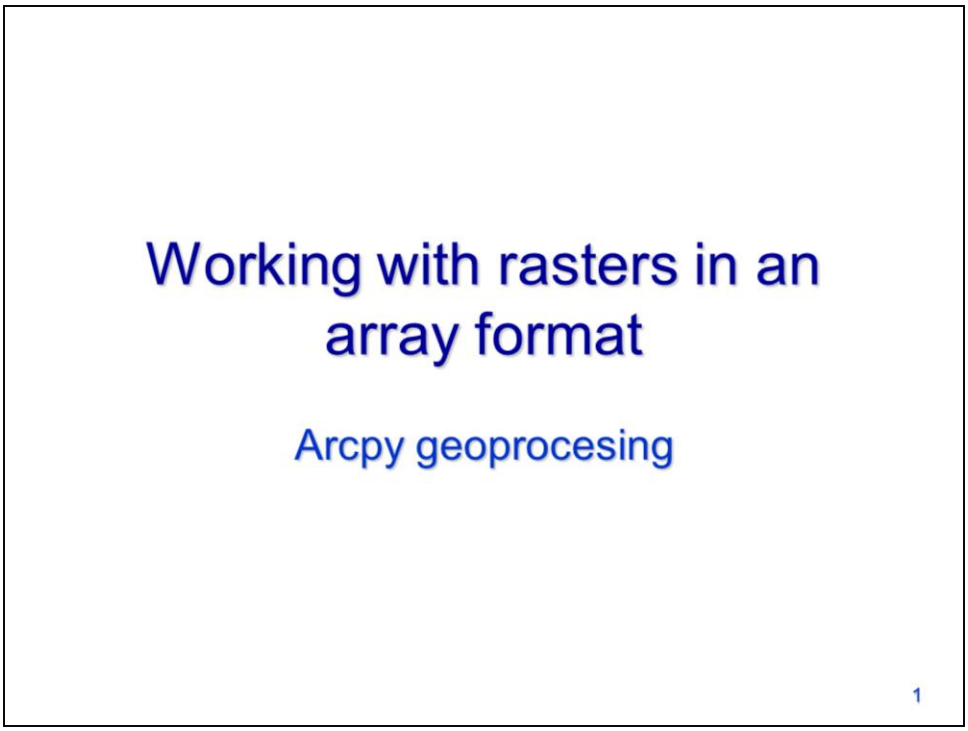
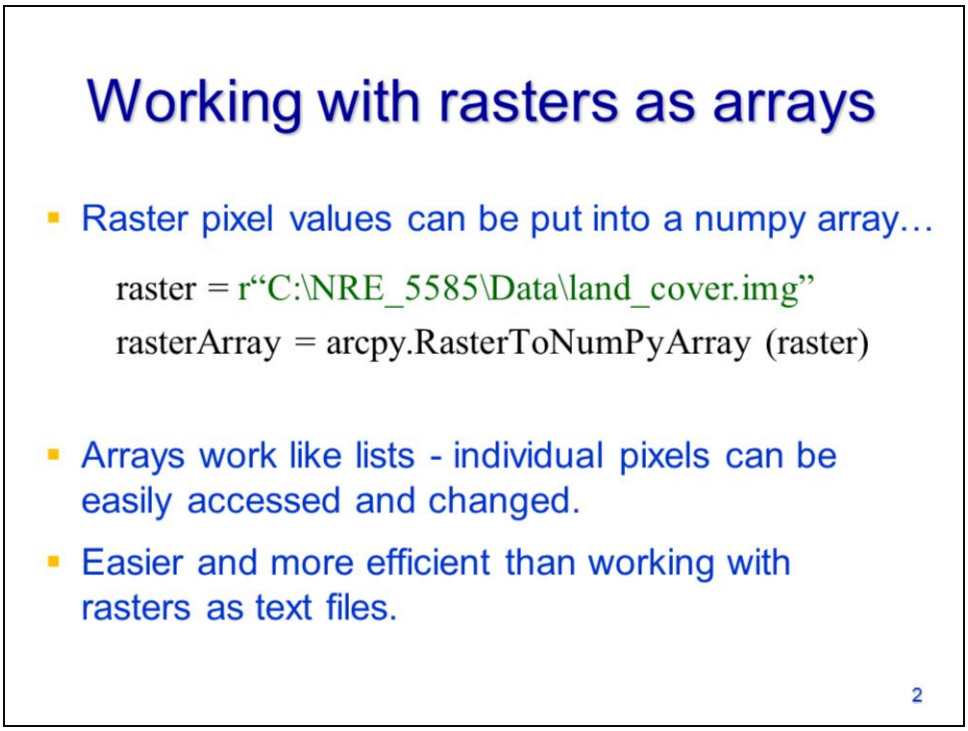


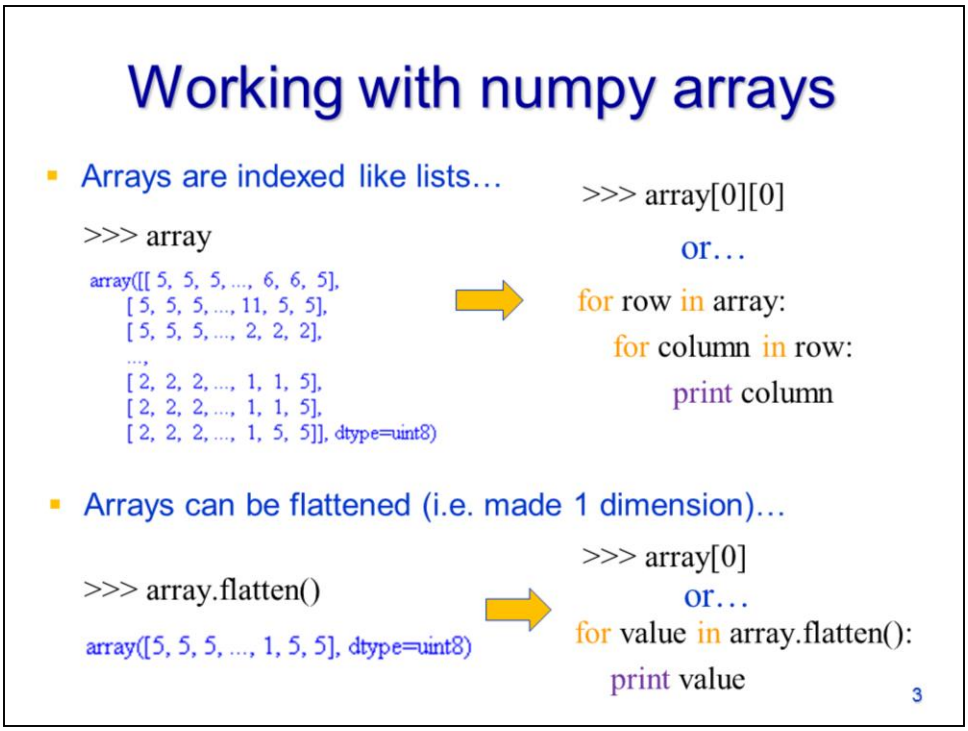
This video will discuss how to work with rasters in an array format.



Raster datasets can be converted to a numpy array using arcpy's **RasterToNumPyArray** tool.

Arrays work similar to lists in the way that pixels values are accessed and modified. Arrays have far more capabilities than lists but are slower for iterative processes.

Rasters can be converted to an array and processed efficiently without the need for spatial analyst tools. Working with raster arrays is quicker and easier than working with raster text files. However, the text file approach is capable of processing larger raster datasets than the array approach.

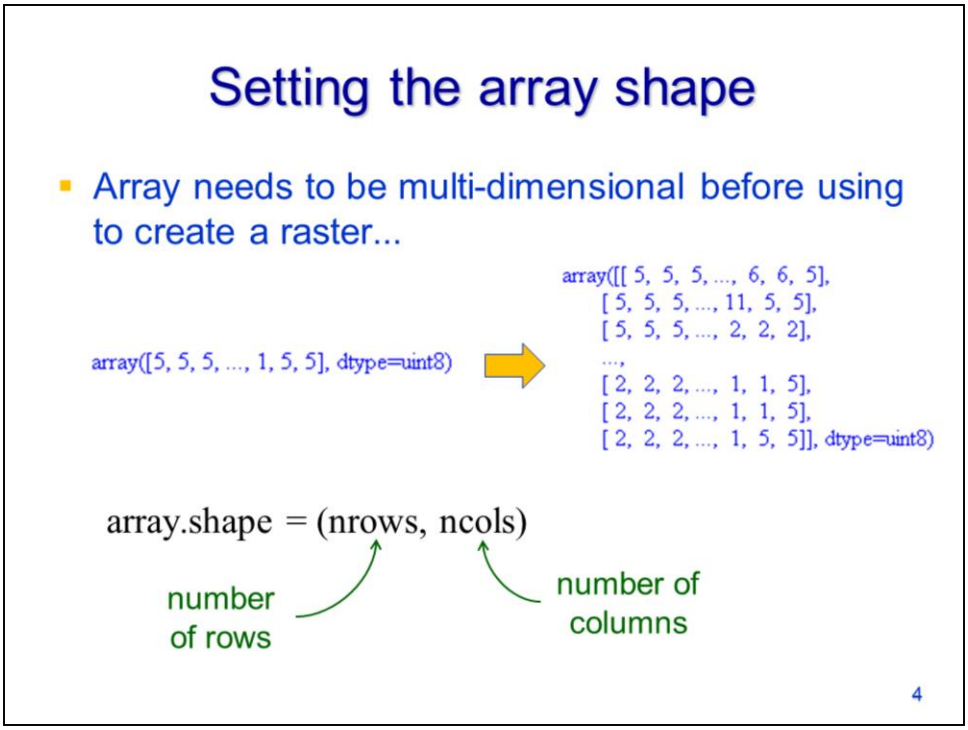


This is an example of an array printed in the python shell. Note that large arrays can be printed in the Python Shell with out a problem; printing other data collections, including lists and dictionaries, will cause the Python Shell to freeze up or respond slowly if they are large.

Items are retrieved from arrays the same as they are retrieved from lists.

Arrays also work with for loops and can have hierarchical structures.

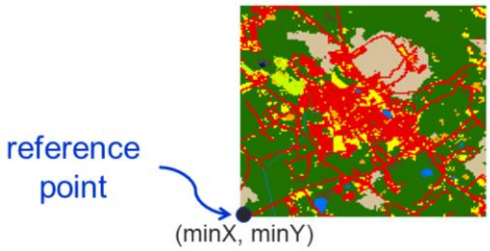
Multi-dimensional arrays can be made one-dimensional using the **flatten** method.



An array needs to be multi-dimensional in order to convert back to a raster.

The shape of an array can be specified using the array's shape property.

Creating the raster reference point




```
extentObj = arcpy.Describe(raster).Extent
minX = extentObj.XMin
minY = extentObj.YMin
refPnt = arcpy.Point(minX, minY)
```

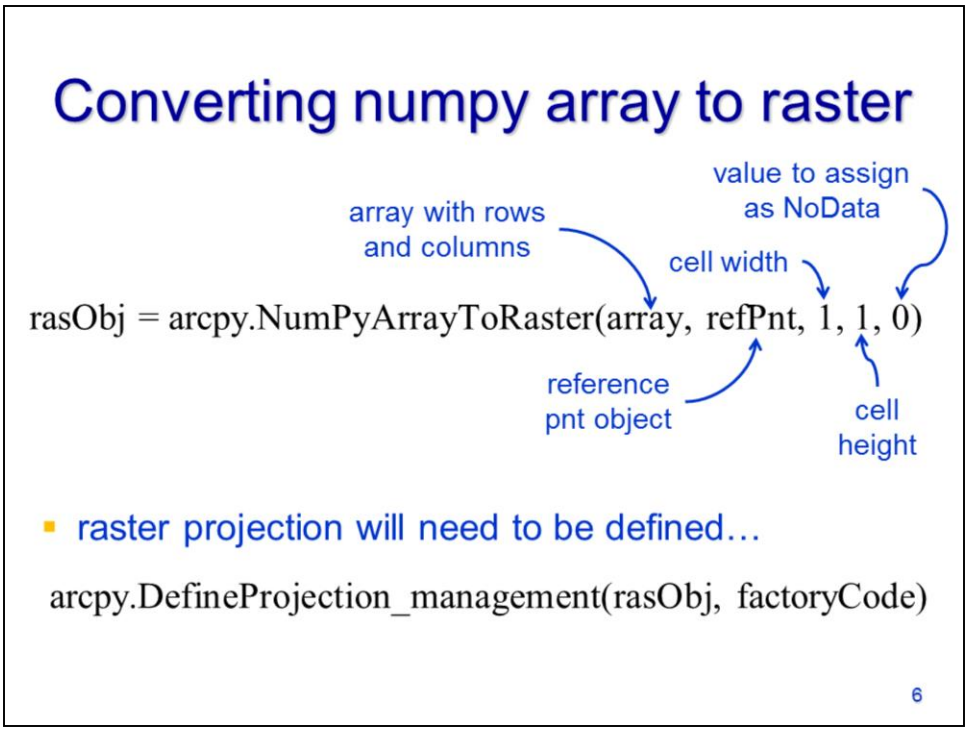
5

When converting a raster to an array, the spatial reference information is lost. This information should be obtained from the original raster and used to set the spatial reference for the new raster.

The extent can be obtained using the describe tool or from a raster object.

The lower left point of the extent is used to georeference the raster.

The **minX** and **minY** coordinates need to be converted to a point object – it will be used as the reference point when converting the array to a raster.



Arcpy's **NumPyArrayToRaser** converts an array to a raster.

The array needs to be multi-dimensional with the number of rows and columns corresponding to the raster.

The reference point can specified as a point object that corresponds to the lower left corner (minX, minY) of the raster extent. The default reference point is 0, 0.

The cell width and height should correspond to the raster pixel size; these parameters will default to 1 if omitted.

The final parameter indicates a pixel value that should be assigned NoData in the raster. In this example, all values of zero in the array will be assigned a NoData value in the raster.

The raster created from the NumPyArrayToRaster will have an undefined projection. The **DefineProjection** tool can be used to define the raster's projection information using a factoryCode.

Numpy array – example script

```
# convert raster to array…
array = arcpy.RasterToNumPyArray (raster)

# calculate statistics from array…
mean = array.mean()    # mean of values
stdev = array.std()    # standard deviation of values

# flatten array…
array = array.flatten()

# process items in array…
for position in range(len(array)):
    value = array[position]
    array[position] = (value – mean) / stdev
# make array multi-dimensional…
array.shape = (1000, 1000)
```

7

This script example will show how to work with a raster in an array form and then return it to a raster format. The first statement converts the raster to a numpy array format.

The array can be used to calculate statistics on the pixel values. In this case, the mean and standard deviation of the pixel values are calculated.

The array is flattened so that it can be processed without the need for a multi-level loop (i.e. loop within a loop). Flattening the array is optional but it affects how the array will be processed.

In this case, the values in the array are processed in a for loop. Each value is retrieved and a new value is calculated and used to replace the original value.

The shape of the array is set to have the number of rows and columns corresponding to the original raster.

Numpy array – example script cont.

```
# create reference point…
extent = arcpy.Describe(raster).Extent
minX = extent.XMin
minY = extent.YMin
refPnt = arcpy.Point(minX, minY)
# get cellsize from input raster…
cellsize = float(arcpy.GetRasterProperties_management(raster,
    "CELLSIZEX").getOutput(0))
# convert array to output raster…
rasObj = arcpy.NumPyArrayToRaster(array, refPnt, cellsize, cellsize)
# save output raster…
rasObj.save(r"C:\NRE_5585\Results\outRaster.img")
```

8

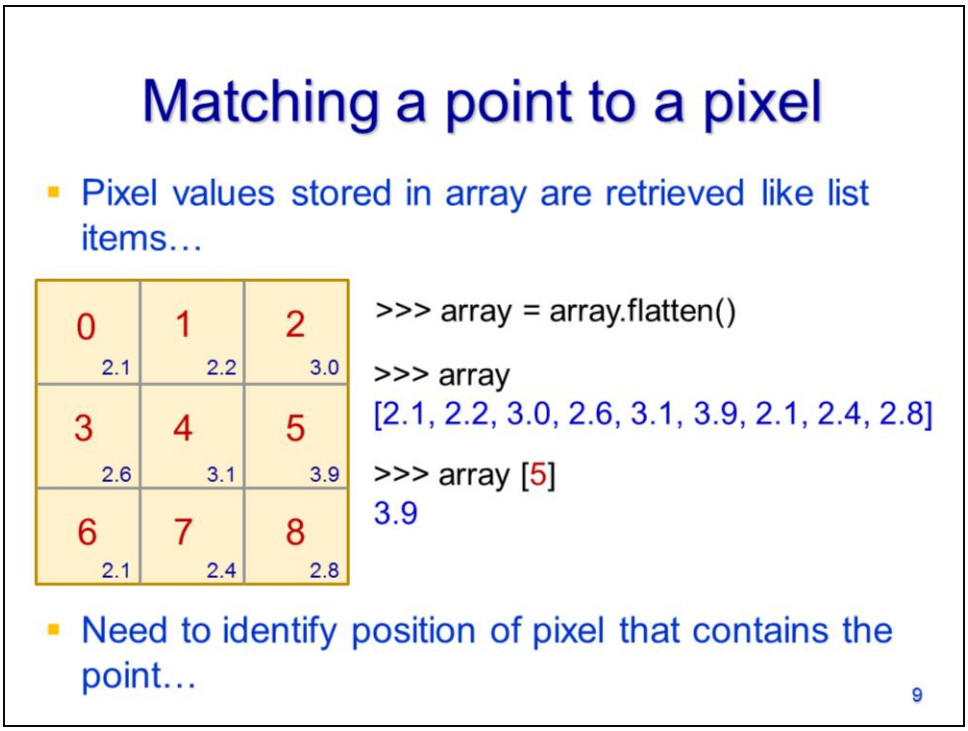
Continuing with the example script on the previous slide…

The minimum X and Y coordinates of the original raster are retrieved and used to create the reference point object.

The cellsize of the original raster is obtained and converted to a decimal number.

The array is then converted back to a raster

And the raster object is saved as a permanent dataset.



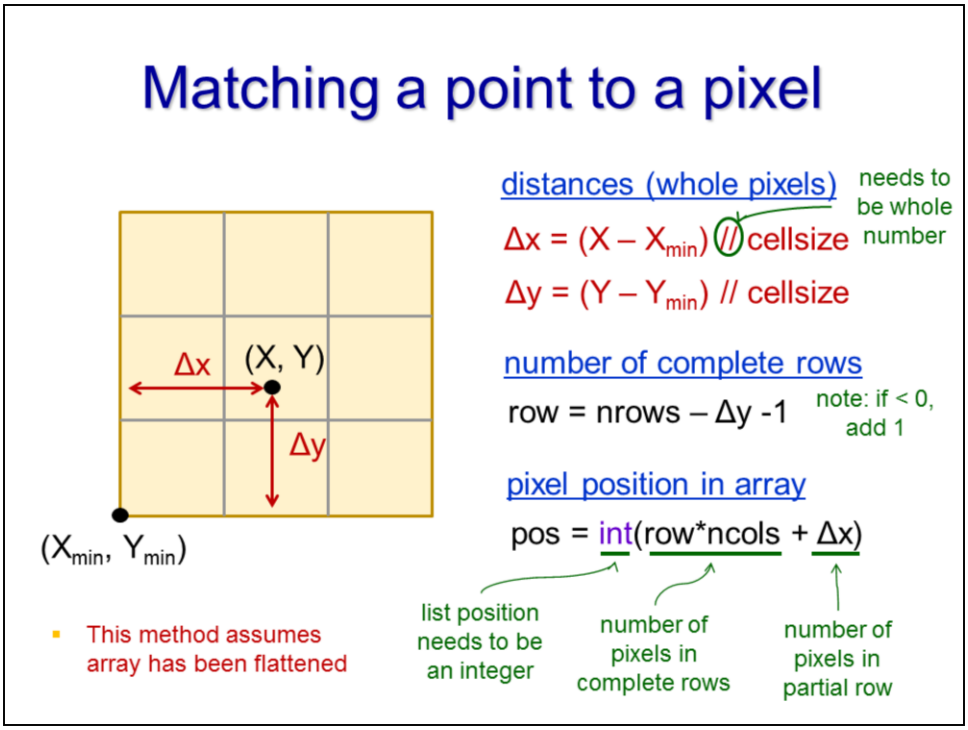
This slide shows how values in a raster array correspond to locations in a raster.

In this example, the array is flattened

And the values in the array are shown in their corresponding locations in the raster.

The value at a particular location in the raster can be retrieved by specifying the corresponding location in the array.

If we can match a point location to a pixel in the raster, then we can retrieve its corresponding value in the array. Identifying the pixel value that corresponds to a point location can be often be useful.



The pixel in which a point is located can be determined based on the X and Y offsets of that point relative to the reference point (minX, minY) of the raster.

The X and Y offset distances need to be calculated in terms of pixels. The distances from map units (e.g. meters) to pixel by dividing by the cellsize. The division is truncated.

The deltaY value tells us the distance (in pixels) from the bottom of the raster to the point-of-interest. However, what we really need to know is the distance (in pixels) from the top of the raster to the point-of-interest. This equation shows how to convert our bottom-up reference to a top-down reference. **nrows** is the number of rows in the raster. Subtracting the distance (in pixels) from the bottom of the raster gives the distance (in pixels) from the top of the raster. Subtracting 1 excludes the row in which the point-of-interest is located. Thus, **row** is the number of full rows that are above the point-of-interest.

This final equation calculates the position of the point-of-interest in the raster. **ncols** is the number of columns in the raster.

The **int** function converts the final position to an integer.

The **row*ncols** gives the number of pixels in the rows above the point-of-interest.

The deltaX gives the column number in which the point-of-interest is located.

Note that the approach discussed here assumes that the array has been flattened – as illustrated on the previous slide. Flattening the array allows this approach to be used with dictionaries as well as arrays and lists.

Calculating pixel key for numpy array - example script

```
# get raster information…
desc = arcpy.Describe (raster)
ncols = desc.Width
nrows = desc.Height
cellsize = desc.MeanCellWidth
extent = desc.Extent
minX = extent.XMin
minY = extent.YMin

# convert raster to array…
rasterArray = arcpy.RasterToNumPyArray(raster).flatten()
```

11

This example script shows how to determine the pixel value at the location of a point. The script begins by getting the necessary information from the original raster including the

1) number of columns,
2) number of rows,
3) cellsize,
4) minimum X coordinate and 5) minimum Y coordinate.

The raster is converted to a numpy array and flattened.

Calculating pixel key for numpy array - example script

```
# calculate X and Y distances in terms of whole pixels…
dX = (X – minX) // cellsize
dY = (Y-minY) // cellsize

# number of complete rows…
row = nrows – dY – 1
if row == -1: row = 0

# calculate pixel position in array…
pos = int(row*ncols + dX)

# get value from array…
pixelValue = rasterArray[pos]
```

12

Here the point-of-interest has the coordinates of (X,Y). The X and Y offsets, relative to the lower left corner of the raster, are calculated in terms of whole pixels.

The number of completed rows above the point-of-interest (X, Y) is calculated – this number must be at least zero (a value of -1 could occur for a point that occurs on the topmost edge of the raster).

The position of the point-of-interest in the array is calculated.

The pixel value corresponding to the point-of-interest is retrieved from the array.