Outline

- What can you do with raster in Python?
- ArcPy modules for raster analysis
- Raster object
- Visualization and exploration of raster data
- Raster data management and processing
- Raster analysis
- NumPy array and custom analysis
Working with Raster in Python

- Visualization and exploration
- Data preprocessing, batch processing, workflow automation
- Data management
- Raster analysis
- Custom raster analysis
Python API for Raster Processing and Analysis in ArcGIS

- **ArcGIS API**
  - Work with ArcGIS Enterprise/ArcGIS Online
  - Remote processing

- **ArcPy**
  - Requires Desktop App (ArcGIS Pro/ArcMap)
  - Local processing
ArcPy – Over 1300+ Comprehensive GIS Tools

**Data Management**
- Convert between formats, 2D <-> 3D, build topology, data reviewer, linear referencing,
- data comparison, manage spatial databases, distributed editing, servers & services
- Specialized datasets – parcels, utility networks, address locators, network datasets

**Raster Analysis**
- Extraction, overlay, proximity, distance, hydrology, multivariate, map algebra, neighborhood, fuzzy overlays

**Imagery analysis**
- **management** – mosaic datasets, raster algebra,
- **ortho mapping** – point cloud, stereo model, photogrammetry, surface creation, ortho rectification
- **information extraction** – classification, object detection, segmentation, spectral analysis, statistics
- Full motion imagery, multidimensional rasters, solar radiation
- Hundreds of math and stat tools

**Cartography**
- annotations, generalization, map series, masks
- 100+ projections and transformations
- **editing tools** – conflation, densify, cartographic refinement
- Layouts, printing, map series

**Spatial Machine Learning**
- Interpolations, spatial sampling, simulations,
- **Stats** – enrich, density, hot spot, clustering, autocorrelations, balanced zones, outliers, similarity search, local bivariate relationships
- **Prediction** – random forests, GLR, OLS, GWR, validation
- **Pattern mining** – emerging hot spot, time series clustering,
- **Big Data** – GeoAnalytics desktop tools
- **Deep Learning** – classify objects, classify pixels, detect objects, export training data, train model

**3D**
- LAS classification, extraction, terrain, TIN, DEM, DSM, volume analysis
- **Surface analysis** – aspect, contour, cut fill, hillshade, slope, line of sight, sun shadow, viewshed, hydrology

**Location analytics**
- **business intelligence** – market penetration, Huff model, suitability analysis, territory design,
- **Network** – routing, OD cost matrix, service area, location allocation
- **Geocoding** – batch, reverse, composite

**Imagery analysis**
- **management** – mosaic datasets, raster algebra,
- **ortho mapping** – point cloud, stereo model, photogrammetry, surface creation, ortho rectification
- **information extraction** – classification, object detection, segmentation, spectral analysis, statistics
- Full motion imagery, multidimensional rasters, solar radiation
- Hundreds of math and stat tools

**Imagery analysis**
- **management** – mosaic datasets, raster algebra,
- **ortho mapping** – point cloud, stereo model, photogrammetry, surface creation, ortho rectification
- **information extraction** – classification, object detection, segmentation, spectral analysis, statistics
- Full motion imagery, multidimensional rasters, solar radiation
- Hundreds of math and stat tools

**Imagery analysis**
- **management** – mosaic datasets, raster algebra,
- **ortho mapping** – point cloud, stereo model, photogrammetry, surface creation, ortho rectification
- **information extraction** – classification, object detection, segmentation, spectral analysis, statistics
- Full motion imagery, multidimensional rasters, solar radiation
- Hundreds of math and stat tools
ArcPy ia and sa Modules

- Modules of ArcPy for raster analysis and image processing
- Like all modules must be imported
- To access the operators and tools in an algebraic format the imports are important

```
import arcpy
from arcpy import env  # Analysis environment
from arcpy.sa import *  # BEST Practice for Map Algebra
from arcpy.ia import *  # BEST Practice for Map Algebra
```

```
import arcpy.sa  # NOT Recommended for Map Algebra
```
Raster Object

• Represents a raster in arcpy
  - Single-band raster, multi-band raster, multidimensional raster

• Output from a Map Algebra expression, pointing to an existing dataset, or an empty raster created using rasterInfo

• Output from Map Algebra or an empty raster is temporary and can be persisted using the save method

• Many properties describing the associated dataset

• Many methods to work with properties, bands, multidimensional variables, dimensions, slices

• Read/Write methods to convert raster to a NumPy array and to create a raster from NumPy array
Raster Object

**Properties**
- properties
- catalogPath
- path
- name
- isTemporary
- readOnly
- isInteger
- hasRAT
- isMultidimensional
- bandCount
- bandNames
- extent
- width
- height
- meanCellWidth
- meanCellHeight
- spatialReference
- maximum
- mean
- minimum
- standardDeviation
- noDataValue
- noDataValues
- mdinfo
- variableNames
- variables
- slices
- format
- pixelType
- compressionType
- uncompressedSize
- blockSize

**Methods**
- save
- exportImage
- getProperty
- setProperty
- getRasterInfo
- getRasterBands
- getVariableAttributes
- renameVariable
- removeVariables
- getDimensionNames
- getDimensionAttributes
- getDimensionValues
- addDimension
- read
- write

Create from a raster dataset
Output from a map algebra expression
Create an empty raster from rasterInfo

**Raster (inRaster, {is_Multidimensional})**
Multidimensional Raster

- Spatial, temporal, and vertical dimensions
- Contains one or multiple variables in one file
  - A variable is cube or cubes
  - A variable has a time and/or depth
  - Each slice is a 2D array
- Multidimensional Raster
  - A Mosaic Dataset
  - Or a Raster Dataset
Visualization and Exploration

- Render function displays a raster object by applying symbology
  
  \texttt{arcpy.ia.Render (in\_raster, \{rendering\_rule\}, \{colormap\})}

- There are other functions to control appearance

- Raster object has several properties and method to explore raster dataset
  
  - isInteger
  - hasRAT
  - isMultidimensional
  - bandCount
  - bandNames
  - extent
  - width
  - height
  - meanCellWidth
  - meanCellHeight
  - spatialReference
  - maximum
  - mean
  - minimum
  - standardDeviation
  - noDataValue
  - noDataValues
## Raster Data Management and Batch Processing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyEnvironment</td>
<td>Creates a new raster that is a copy of the input raster with the current environment settings being applied.</td>
</tr>
<tr>
<td>Clip</td>
<td>Creates a raster object that is clipped according to the extent of an input raster dataset or the shape of an input polygon feature class.</td>
</tr>
<tr>
<td>CompositeBand</td>
<td>Creates a raster object by combining multiple rasters to form a multiband image.</td>
</tr>
<tr>
<td>ExtractBand</td>
<td>Creates a raster object by extracting one or more bands from, or reordering the bands in, a multiband raster.</td>
</tr>
<tr>
<td>Mask</td>
<td>Creates a raster object by specifying one or more NoData values or a range of valid pixel values.</td>
</tr>
<tr>
<td>Merge</td>
<td>Creates a raster object by merging a list of rasters spatially or across dimensions.</td>
</tr>
<tr>
<td>Resample</td>
<td>Creates a raster object by changing the spatial resolution of the input raster and sets rules for aggregating or interpolating values across the new pixel sizes.</td>
</tr>
<tr>
<td>Subset</td>
<td>Creates a raster object that is a subset of the input multidimensional raster based on selected variables and dimension intervals.</td>
</tr>
<tr>
<td>TransposeBits</td>
<td>Creates a raster object by unpacking the bits of the input pixel and mapping them to specified bits in the output pixel. The purpose of this function is to manipulate bits from a couple of inputs, such as the Landsat 8 quality band products.</td>
</tr>
</tbody>
</table>

```python
arcpy.ListRasters ({wild_card}, {raster_type})
```
Raster Data Management in ArcGIS

- **Raster Dataset**
  - Represent single dataset

- **Mosaic Dataset**
  - Dynamic mosaic of imagery collection
  - Table of records
  - A Raster Dataset
  - Lives in Geodatabase

CreateMosaicDataset(in_workspace, in_mosaicdataset_name, coordinate_system, {num_bands}, {pixel_type}, {product_definition}, {product_band_definitions})

• Defines the group of input rasters and attributes to include in a collection, which will be used for processing.

`RasterCollection (rasters, {attribute_dict})`

**Properties**
- fields
- count

**Methods**
- filter
- filterByAttribute
- filterByCalendarRange
- filterByGeometry
- filterByRasterProperty
- filterByTime
- getFieldValues
- majority
- map
- max
- mean
- median
- min
- mosaic
- qualityMosaic
- selectBands
- sort
- sum
- toMultidimensionalRaster
Exploration Visualization Preprocessing

Hao Hu
Raster Analysis using Map Algebra

• Simple and **powerful algebra** to execute tools, operators, and functions to perform raster analysis
  - Comprised of input data, operators, tools, functions, parameters and output
  - output raster is specified to the **left** of an equal sign and
  - the tool and its parameters on the **right**

• The strength is in creating **complex expressions**

• Tools, operator and functions are available in **Spatial Analyst and Image Analyst modules**

• You can specify analysis environment using **arcpy.env**
Map Algebra Tools and Functions

- All Spatial Analyst and Image Analyst tools are available (e.g., Sin, Slope, Reclassify, etc.)
- Additional functions are available in arcpy.ia and arcpy.sa modules
- Spatial Analyst and Image Analyst tools/functions will require to checkout license
- Can use any Geoprocessing tools
- Each tool has its own unique set of parameters
- Some are **required**, others are **optional**
- Numbers, strings, and objects (classes)

```python
Slope(in_raster, {output_measurement}, {z_factor})
```

- `outRas = Slope(inRaster, "DEGREE", 0.3048)`
- `outRas = Slope(inRaster, "", 0.3048)`
- `outRas = Slope(inRaster)`

**Tip:** Keywords are in quotes and it is recommended they are capitalized
Map Algebra Operators

- Symbols for mathematical operations
- Many operators in both Spatial Analyst and Image Analyst

\[
\text{outRas1} = \text{inRaster1} + \text{inRaster2}
\]

- Cast the raster (Raster class constructor) to indicate operator should be applied to rasters

\[
\text{outRas1} = \text{Raster(“rasternam1”) + Raster(“rasternam2”)} \\
\text{outRas2} = \text{Raster(“rasternam1”) + 8} \\
\text{outRas3} = \text{outRas2 + Raster(“rasternam2”) * 8}
\]
Map Algebra Output

- Returns the results as a **Raster object**
- The output is **temporary**
- Make it permanent by calling the **save** method

```python
save({name})
outRas = Slope(inRaster, "DEGREE", 0.3048)
outRas.save("mySlopeOutput.tif")
```
Complex Expressions and Optimization

- Multiple operators and tools can be executed in a single expression
- Output from one expression can be the input to a subsequent expression
- A series of local tools (Abs, Sin, Cell Statistics, etc.) and operators can be optimized
  - Operators and local tools work on a per-cell basis
  - When entered into a single expression all local tools and operators are processed together on a per-cell basis

```
In [47]: suitability_map = Reclassify(Slope(elev * 0.3048),
   "Value",
   RemapRange([[0,3,1], [3,10,3], [10,25,6], [25,90,10]]) * 0.2 + \n   Reclassify(landuse,
   "Value",
   RemapValue([[4,2], [11,10], [1,1], [3,1], [2,1], [7,10],
   [6,10], [8,10], [5,8], [9,6], [12,1], [10,2]]) * 0.5 + \n   RescaleByFunction(DistanceAccumulation(arcpy.Clip_analysis("Streams",
   "StudyArea",
   "#")),
   TfHSSmall(), 1, 10) * 0.3
```
Classes

• Objects that are used as inputs to tools
  - Varying number of arguments depending on the selected parameter choice (neighborhood type)
  - The number of entries into the parameters can vary depending on the specific situation (a remap table)
• More flexible, you can query and modify the individual arguments

• Creating
  \[ nbr01 = NbrCircle(4, \text{“MAP”}) \]
• Querying
  \[ radius01 = nbr01.radius \]
• Changing arguments
  \[ nbr01.radius = radius01 + 2 \]
Vector integration

- Feature data is required for some raster analysis
  - IDW, Kriging etc.

- Geoprocessing tools that operate on feature data can be used in an expression
  - Buffer, Select, etc.

```python
rivbuff = arcpy.Buffer_analysis(arcpy.RasterToPolyline_conversion(riv,'#'),'rivbuff.shp',100)
```
Analysis

Suitability Modeling

Hao Hu
NumPy

• Can be used to extend raster analysis capabilities by creating
  - custom functions
  - custom tools

• An extension package to Python
  - Adds support for large, multi-dimensional arrays
  - Provides a large library of high-level mathematical functions

• Access the wealth of free functions built by the scientific community
  - Clustering
  - Filtering
  - Linear algebra
  - Optimization
  - Fourier transformation
  - Morphology
NumPy Arrays

• Two functions to work with raster
  
  RasterToNumPyArray(in_raster, {lower_left_corner}, {ncols}, {nrows}, {nodata_to_value})
  
  NumPyArrayToRaster (in_array, {lower_left_corner}, {x_cell_size}, {y_cell_size}, {value_to_nodata})

• Raster object Read and Write methods
  
  read ({upper_left_corner}, {origin_coordinate}, {ncols}, {nrows}, {nodata_to_value}, {cell_size})
  
  write (array, {upper_left_corner}, {origin_coordinate}, {value_to_nodata})
PixelBlock and PixelBlockCollection

- The PixelBlock object defines a block of pixels within a raster to use for processing. It is used in conjunction with the PixelBlockCollection object to iterate through one or more large rasters for processing.
- `getData()` method returns a NumPy array within the extent
  
  ```python
  PixelBlock(data, extent)
  ```

- The PixelBlockCollection object is an iterator of all PixelBlock objects in a raster or a list of rasters. It can be used to perform customized raster processing on a block-by-block basis, when otherwise the processed rasters would be too large to load into memory.
  
  ```python
  PixelBlockCollection(in_rasters, pixel_block_size, stride, overlay_type, nodata_to_values)
  ```
Raster Cell Iterator (RCI)

- Visit each cell in a Raster object by row/column indices and read/write cell values
- Support iterating multiple raster datasets and perform analysis at the same location
- Access to cell values at each location and neighboring locations is efficient
- Available through the `Spatial Analyst module`

```python
arcpy.sa.RasterCellIterator(*args),
*args is a dictionary that defines rasters and padding in the following format:
{"rasters": [in_rasterobj1, in_rasterobj2, ...], "padding": padding_factor}
```
with RasterCellIterator({'rasters':[dem, new_slope]}) as rcl:
    for r,c in rcl:
        slopes = []

        # Iterate through 8 neighbors
        for x,y in [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]:
            # Calculate the slope from center cell to each neighbor
            if dem[r,c] >= dem[r+x,c+y]:  # only look for downhill slope
                slope = abs(dem[r,c]-dem[r+x,c+y]) / math.sqrt((x*cell_x)**2+(y*cell_y)**2)
                slopes.append(slope)

        # Assign the steepest slope to the output cell value
        if len(slopes) != 0:
            new_slope[r,c] = max(slopes)

print('Calculation finished.
new_slope.save(r'C:\DevSummit2\PythonRaster\Data\SteepestDownhillSlope\Output\new_slope')
print('Output has been saved.')
Raster Analysis using Multiprocessing

- Types of raster operations:
  - Local
  - Focal
  - Zonal
  - Global

Local raster operation

Focal raster operation
Pleasingly Parallel Problems

Tool executed parallelly on a large input dataset.
Performance Improvement using Multiprocessing

Execute tool in parallel on 57344*57344 cells

- Method: Focal function in parallel; 16 chunks created
- Method: Local function in parallel; 16 chunks created
Use of Multiprocessing for Raster Analysis

```python
if __name__ == '__main__':
    # start the clock
    time_start = time.clock()
    # call create fishnet function
    logger.info("Creating fishnet features...")
    create_fishnet(in_raster_path, out_fishnet_path)
    # get extents of individual features, add it to a dictionary
    extDict = {}
    count = 1
    for row in arcpy.da.SearchCursor(out_fishnet_path, ['SHAPE@']):
        extent_curr = row[0].extent
        ls = []
        ls.append(extent_curr.XMin)
        ls.append(extent_curr.YMin)
        ls.append(extent_curr.XMax)
        ls.append(extent_curr.YMax)
        extDict[count] = ls
        count += 1
    # create a process pool and pass dictionary of extent to execute task
    pool = Pool(processes=cpu_count())
    pool.map(execute_task, extDict.items())
    pool.close()
    pool.join()
    # add results to mosaic dataset
    arcpy.env.workspace = os.getcwd()
    in_path = os.path.join(os.getcwd(), r"local_rast.setHorizontalSpace")
    arcpy.AddRastersToMosaicDataset_management("mosaic_out.gdb\localFn_Mosaic", "Raster Dataset", in_path)
    # end the clock
    time_end = time.clock()
    logger.info("Time taken in main in seconds(s) is : {}".format(str(time_end-time_start)))
```
Use of Multiprocessing for Raster Analysis

Multiprocessing with ArcGIS - Raster Analysis

With growing raster dataset sizes, the processing times for analysis are becoming longer and longer. By employing concepts from parallel and distributed computing, we can improve performance and scalability of workflows with Spatial Analyst tools. For some tools, Spatial Analyst offers enhanced performance with built-in parallel processing capability. These tools use the multi-core processors on modern computing hardware to complete processing tasks more quickly. Additionally, the ArcGIS Image Server provides capabilities for scalable distributed raster analysis and distributed storage of large image and raster collections.
Summary

• You can visualize, explore, manage, analyze and share raster data using ArcPy.

• Use ArcPy ia and sa modules primarily to work with raster.

• Raster object represents a raster and provides many useful properties and methods for single-band raster, multi-band raster, and multidimensional raster.

• Map Algebra is a simple but powerful way to perform raster analysis using tools, functions, and operators.

• You can combine raster and vector analysis tools together in an expression.

• Raster object output from map algebra expression is temporary by default. Save it if you need it as a final output.

• You can extend your analytical capability by writing custom code using NumPy, Raster Cell Iterator etc.