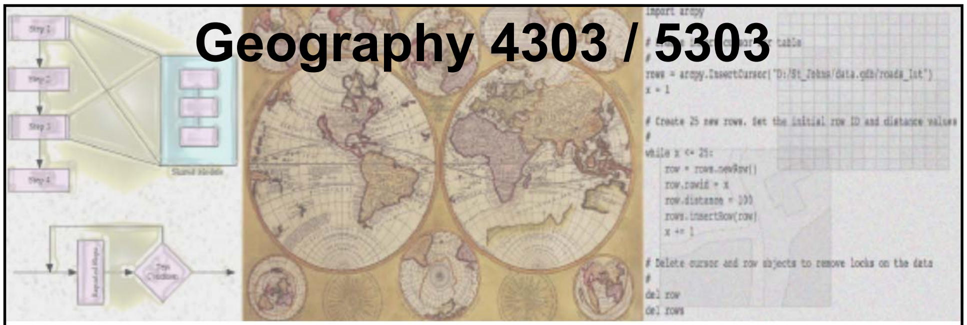


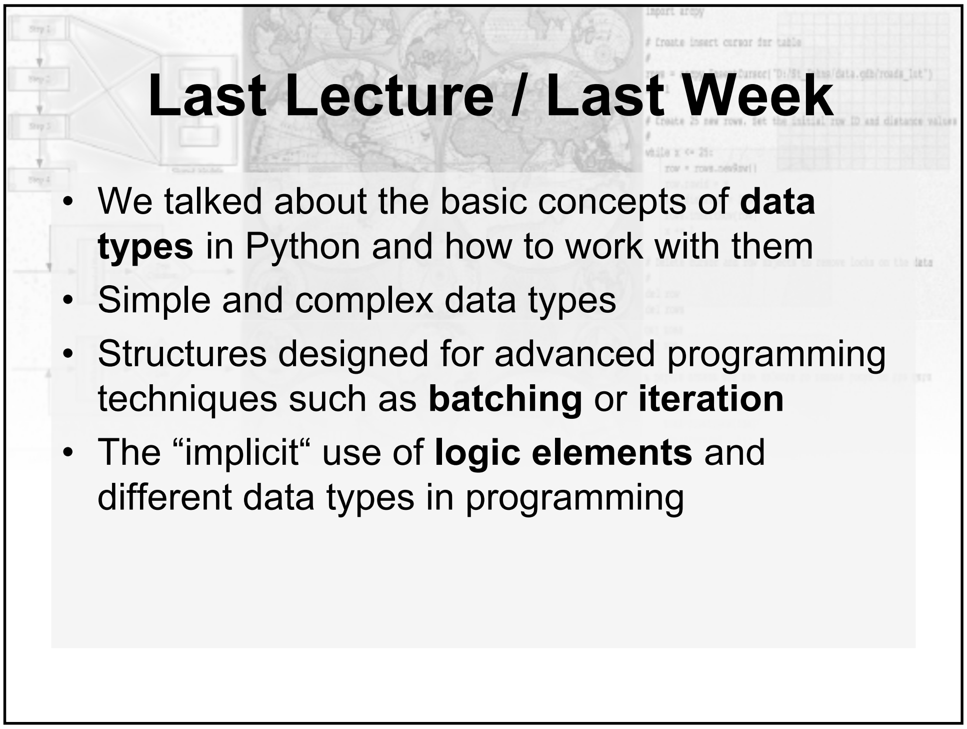
Geography 4303 / 5303

GIS PROGRAMMING FOR SPATIAL ANALYSIS

Class 03: Geoprocessing in Python

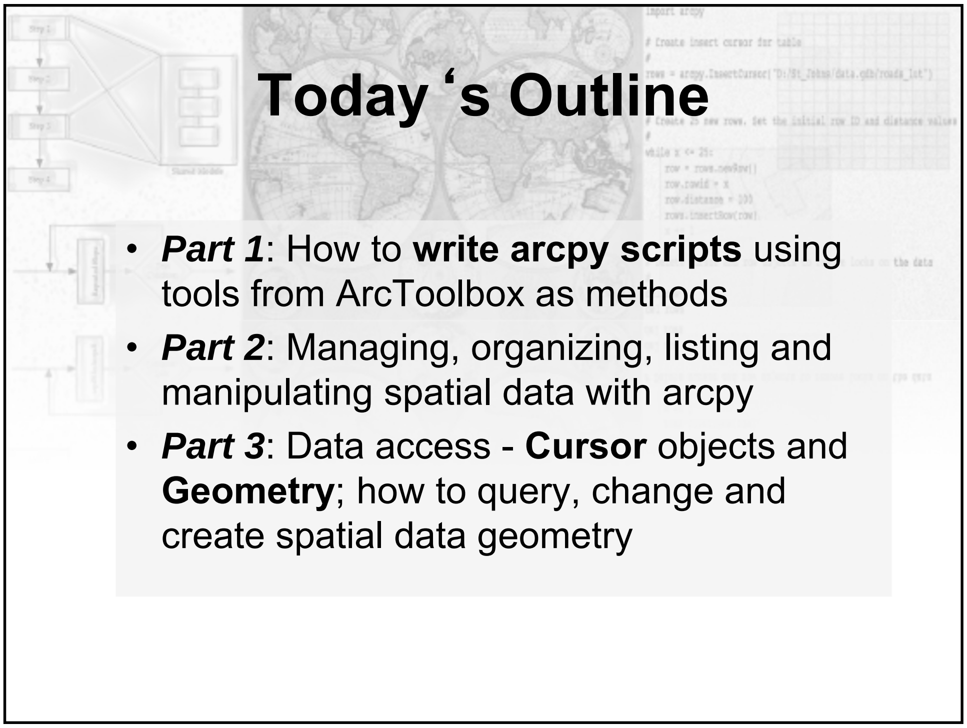
Some Updates

- Class website
- Reminder: Project proposals … and procedure
- Lab exercises and demos… any questions?



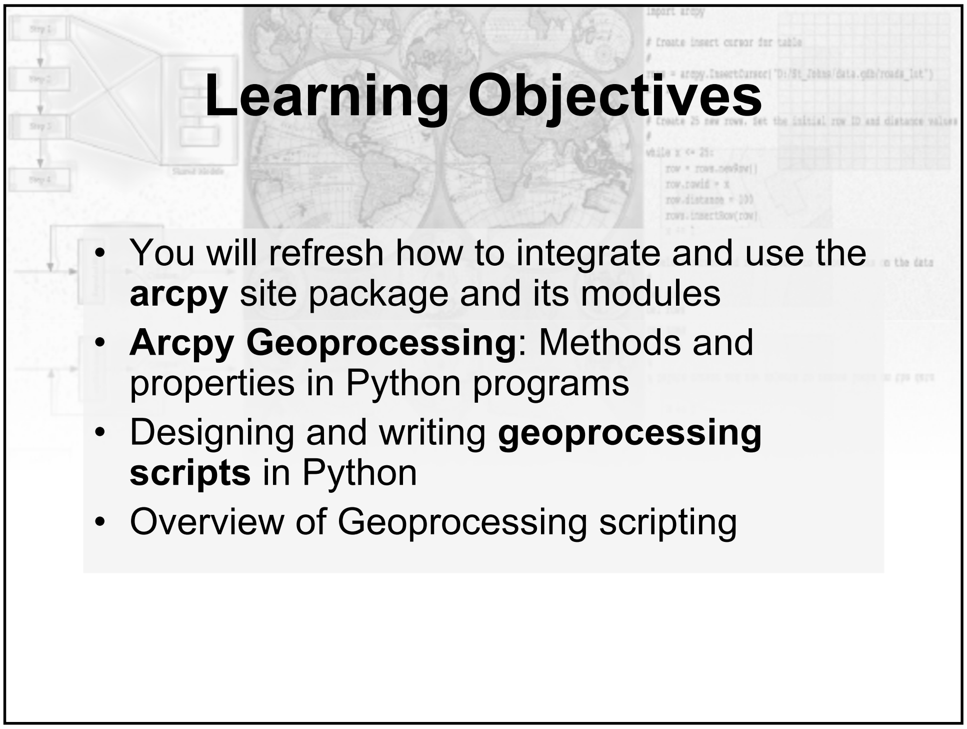
Last Lecture / Last Week

- We talked about the basic concepts of **data types** in Python and how to work with them
- Simple and complex data types
- Structures designed for advanced programming techniques such as **batching** or **iteration**
- The “implicit“ use of **logic elements** and different data types in programming



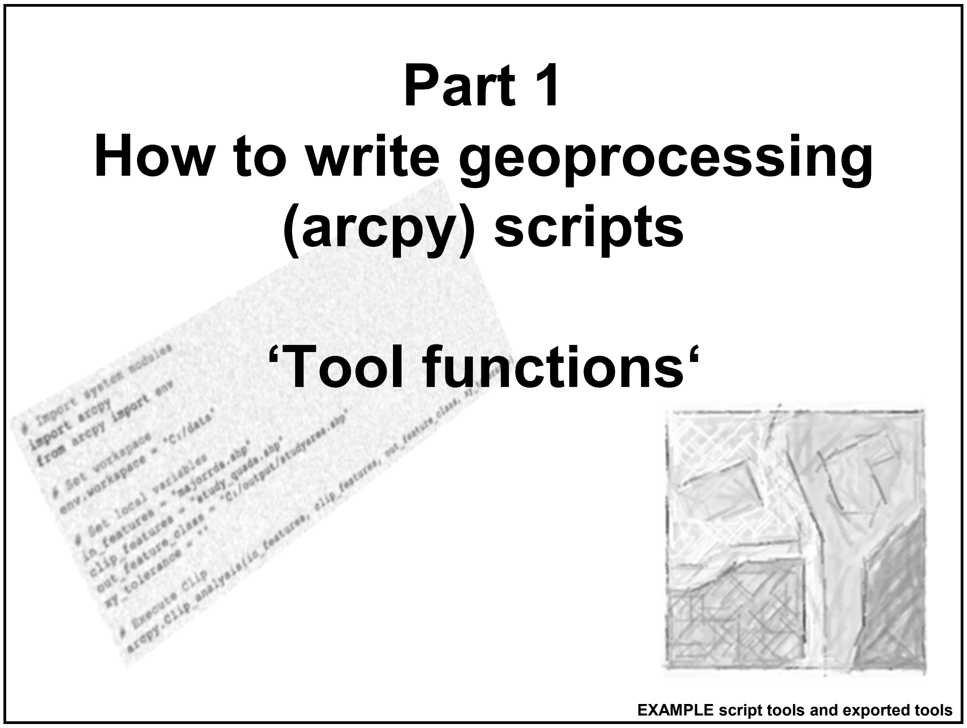
Today ‘s Outline

- ***Part 1***: How to **write arcpy scripts** using tools from ArcToolbox as methods
- ***Part 2***: Managing, organizing, listing and manipulating spatial data with arcpy
- ***Part 3***: Data access - **Cursor** objects and **Geometry**; how to query, change and create spatial data geometry



Learning Objectives

- You will refresh how to integrate and use the **arcpy** site package and its modules
- **Arcpy Geoprocessing**: Methods and properties in Python programs
- Designing and writing **geoprocessing scripts** in Python
- Overview of Geoprocessing scripting



Part 1 How to write geoprocessing (arcpy) scripts

‘Tool functions‘

EXAMPLE script tools and exported tools

Parts of your arcpy script

# 1. Import arcpy and other modules

# 2. Define workspace

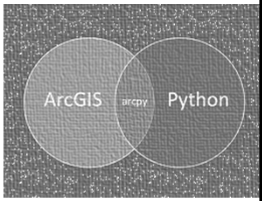
# 3. Variable definitions

# 4. Add try: and except: blocks

# 5. Add geoprocessing function (see Help)

- – # check to see if output already exists
- – # if it does, delete it
- – # Parameters using variables
- – (indent within the try: block)

```
# Import system modules
import arcpy
from arcpy import env

# Set workspace
env.workspace = "C:/data"

# Set local variables
in_features = "majorrds.shp"
clip_features = "study_quads.shp"
out_feature_class = "C:/output/studyarea.shp"
xy_tolerance = ""

# Execute Clip
arcpy.Clip_analysis(in_features, clip_features, out_feat
```



EXAMPLE (class03_01 prog struc)

Access to arcpy

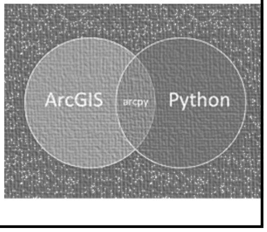
- Python needs to know we want to use **arcpy and its functionality**
- Import the modules you need

```
import arcpy, ...
from arcpy import env       #env class
from arcpy.mapping import * #mapping module
import arcpy as ap
```

- Structure your program

  variable declaration/assignments

  `try:` / `except:`



Other important modules

- `os`

access to operating system functionality e.g., file and directory path operations; often needed for file and data management that use Python standard syntax

- `sys`

Python system functions access e.g., for user input variables `(sys.argv[1])`

- `traceback`

error handling

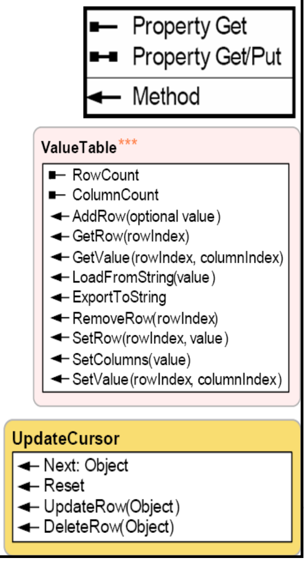
Interacting with the Geoprocessing Environment

- Through import arcpy access to properties and methods

  Property: Characteristic of an object – Get/Put option

  Method: Action an object can perform, may return a value or object

- We **interact** with **arcpy** by calling methods and properties, creating other objects, ...



Property Get
Property Get/Put
Method

ValueTable***
RowCount
ColumnCount
AddRow(optional value)
GetRow(rowIndex)
GetValue(rowIndex, columnIndex)
LoadFromString(value)
ExportToString
RemoveRow(rowIndex)
SetRow(rowIndex, value)
SetColumns(value)
SetValue(rowIndex, columnIndex)

UpdateCursor
Next: Object
Reset
UpdateRow(Object)
DeleteRow(Object)

Use Properties and Methods

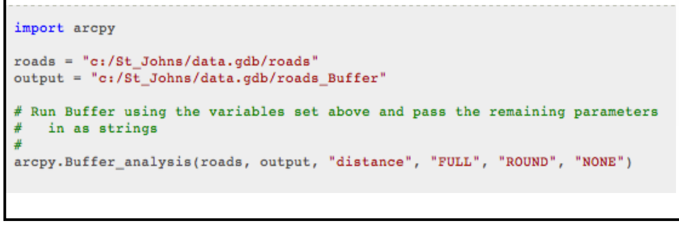
- Assignment of property values:
  ```
  <object.property = value>
  arcpy.env.workspace = "C:\\Temp"  #prop. of env class
  ```
- Get a property value
  ```
  <object.property>
  print arcpy.env.workspace
  ```
- Call and use a method with the arguments needed
  ```
  <object.Method(arg1,arg2,…)>
  arcpy.Buffer_analysis("Roads","Roadsbuffer”,"100")
  ```
  Arguments in parentheses, separated by commas (strings, objects or numbers)

Demo interactive

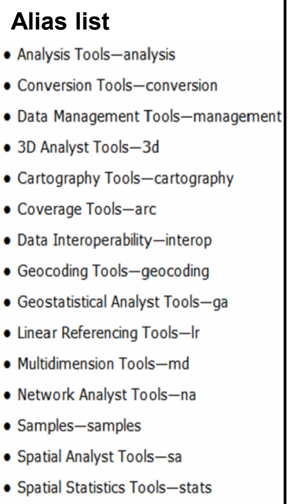
Tools as Methods

- Be specific about the tool you want to run by using the **alias** of the toolbox in referencing the tool

```
arcpy.Buffer_analysis(...)

OR:
arcpy.analysis.Buffer(...)
```



```
import arcpy

roads = "c:/St_Johns/data.gdb/roads"
output = "c:/St_Johns/data.gdb/roads_Buffer"

# Run Buffer using the variables set above and pass the remaining parameters
#   in as strings
#
arcpy.Buffer_analysis(roads, output, "distance", "FULL", "ROUND", "NONE")
```



Alias list

- Analysis Tools—analysis
- Conversion Tools—conversion
- Data Management Tools—management
- 3D Analyst Tools—3d
- Cartography Tools—cartography
- Coverage Tools—arc
- Data Interoperability—interop
- Geocoding Tools—geocoding
- Geostatistical Analyst Tools—ga
- Linear Referencing Tools—lr
- Multidimension Tools—md
- Network Analyst Tools—na
- Samples—samples
- Spatial Analyst Tools—sa
- Spatial Statistics Tools—stats

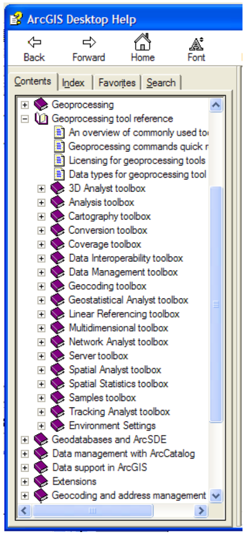
Intellisense & gdb

How to Get Help ...

- ArcGIS **Desktop and online Help** provide:

  (1) Usage, command syntax and examples of how to create scripts with **standard tools** from ArcToolbox

  (2) Usage and syntax of **additional properties and methods**, which are accessible through scripting only



- Geoprocessing
- Geoprocessing tool reference
  - An overview of commonly used to
  - Geoprocessing commands quick r
  - Licensing for geoprocessing tools
  - Data types for geoprocessing tool
  - 3D Analyst toolbox
  - Analysis toolbox
  - Cartography toolbox
  - Conversion toolbox
  - Coverage toolbox
  - Data Interoperability toolbox
  - Data Management toolbox
  - Geocoding toolbox
  - Geostatistical Analyst toolbox
  - Linear Referencing toolbox
  - Multidimensional toolbox
  - Network Analyst toolbox
  - Server toolbox
  - Spatial Analyst toolbox
  - Spatial Statistics toolbox
  - Samples toolbox
  - Tracking Analyst toolbox
  - Environment Settings
- Geodatabases and ArcSDE
- Data management with ArcCatalog
- Data support in ArcGIS
- Extensions
- Geocoding and address management

Show options

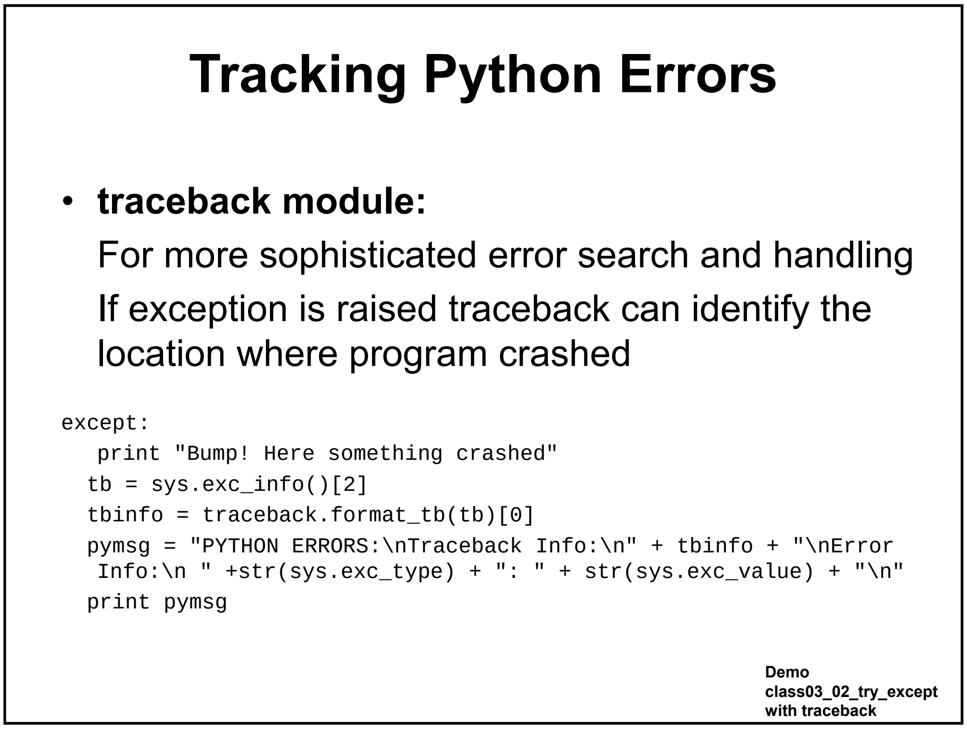
Handling of errors

- **Try/Except blocks & print statements:**

  If an error occurs in the **try** statement, **exception** is raised and code under the except: statement is executed

```
try:
  # Process: Buffer...
  arcpy.Buffer_analysis()
except:
  print "Oops, here is something wrong..."
```

Demo
class03_02_try_except
w/out traceback



Tracking Python Errors

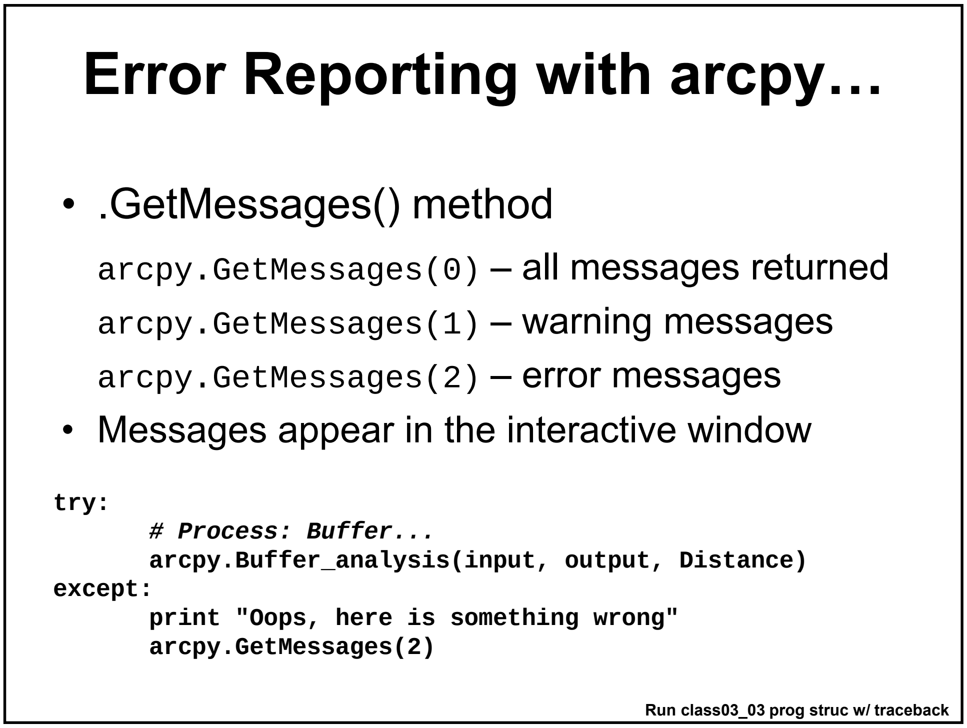
- **traceback module:**

  For more sophisticated error search and handling

  If exception is raised traceback can identify the location where program crashed

```
except:
   print "Bump! Here something crashed"
  tb = sys.exc_info()[2]
  tbinfo = traceback.format_tb(tb)[0]
  pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError
   Info:\n " +str(sys.exc_type) + ": " + str(sys.exc_value) + "\n"
  print pymsg
```

Demo
class03_02_try_except
with traceback



Error Reporting with arcpy…

- .GetMessages() method

  `arcpy.GetMessages(0)` – all messages returned

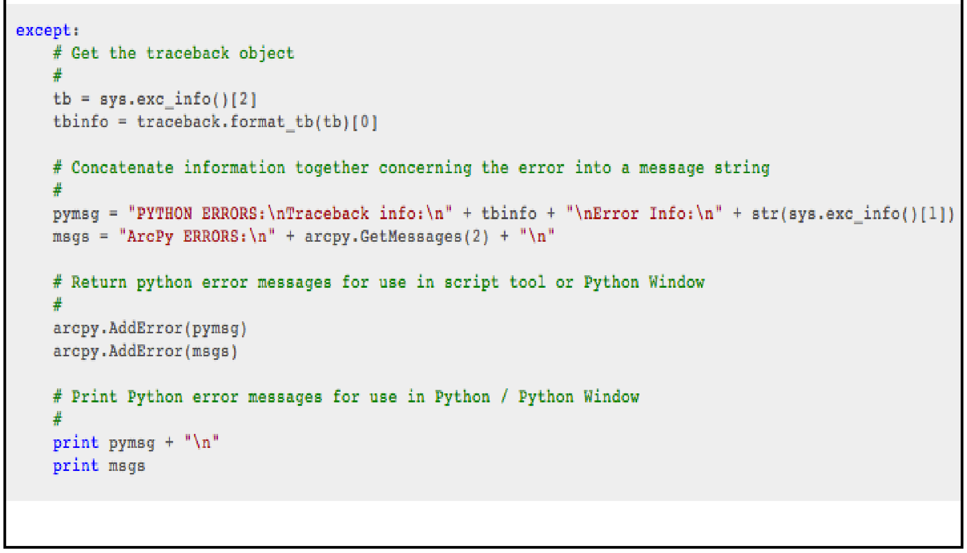
  `arcpy.GetMessages(1)` – warning messages

  `arcpy.GetMessages(2)` – error messages
- Messages appear in the interactive window

```
try:
       # Process: Buffer...
       arcpy.Buffer_analysis(input, output, Distance)
except:
       print "Oops, here is something wrong"
       arcpy.GetMessages(2)
```

Run class03_03 prog struc w/ traceback

Python & arcpy Errors Combined



```
except:
    # Get the traceback object
    #
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]

    # Concatenate information together concerning the error into a message string
    #
    pymsg = "PYTHON ERRORS:\nTraceback info:\n" + tbinfo + "\nError Info:\n" + str(sys.exc_info()[1])
    msgs = "ArcPy ERRORS:\n" + arcpy.GetMessages(2) + "\n"

    # Return python error messages for use in script tool or Python Window
    #
    arcpy.AddError(pymsg)
    arcpy.AddError(msgs)

    # Print Python error messages for use in Python / Python Window
    #
    print pymsg + "\n"
    print msgs
```

Summary Part I

- All about **scripting** for Geoprocessing
- The idea to **implement** the Geoprocessing object follows the principle of using **IDispatch** interfaces (using **COM** objects)
- While scripting we use available **tools** as **methods** and **properties** accessed via arcpy
- Geoprocessing Scripting in Python is **EASY**.
- Each tool in **ArcToolbox** relates to one logical line of source code, generally

AddField Script?

Part 2
Managing, organizing, listing and describing spatial data with arcpy

A Step Back

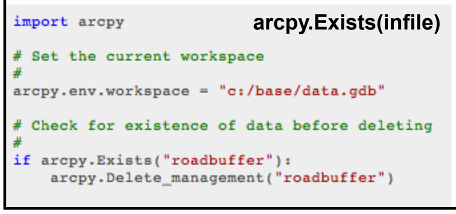
- => Tool functions (ArcToolbox)
- => Cataloguing, organizing and listing spatial data: Batching
- => Describing spatial data and their properties
- => Creating, editing and manipulating spatial data
- => More complex tasks combining above (e.g., Sampling in space w/ geometry)

Additional arcpy functionality

Additional properties and methods, which are accessible through scripting only

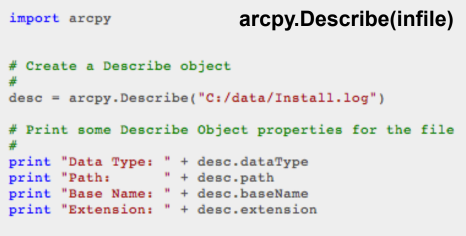
These are displayed in the **Geoprocessor Model Diagram**

In short, these are functionalities for **cataloguing**, **describing**, **listing** and creating/**editing** spatial data



arcpy.Exists(infile)

```
import arcpy

# Set the current workspace
#
arcpy.env.workspace = "c:/base/data.gdb"

# Check for existence of data before deleting
#
if arcpy.Exists("roadbuffer"):
    arcpy.Delete_management("roadbuffer")
```

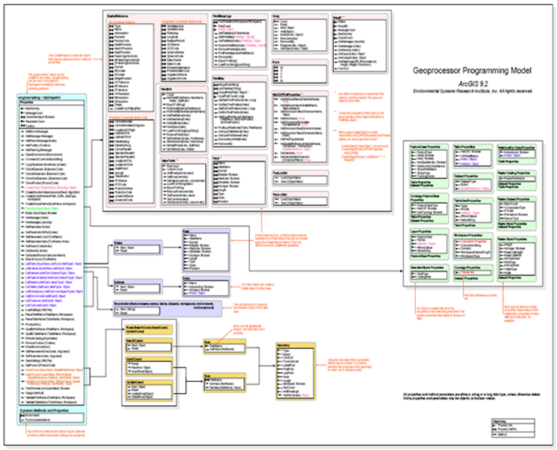


arcpy.Describe(infile)

```
import arcpy

# Create a Describe object
#
desc = arcpy.Describe("C:/data/Install.log")

# Print some Describe Object properties for the file
#
print "Data Type: " + desc.dataType
print "Path:      " + desc.path
print "Base Name: " + desc.baseName
print "Extension: " + desc.extension
```

The Old Geoprocessor Programming Model

- “Road Map” for GP


Geoprocessor Programming Model
ArcGIS 9.2
Environmental Systems Research Institute, Inc. All rights reserved

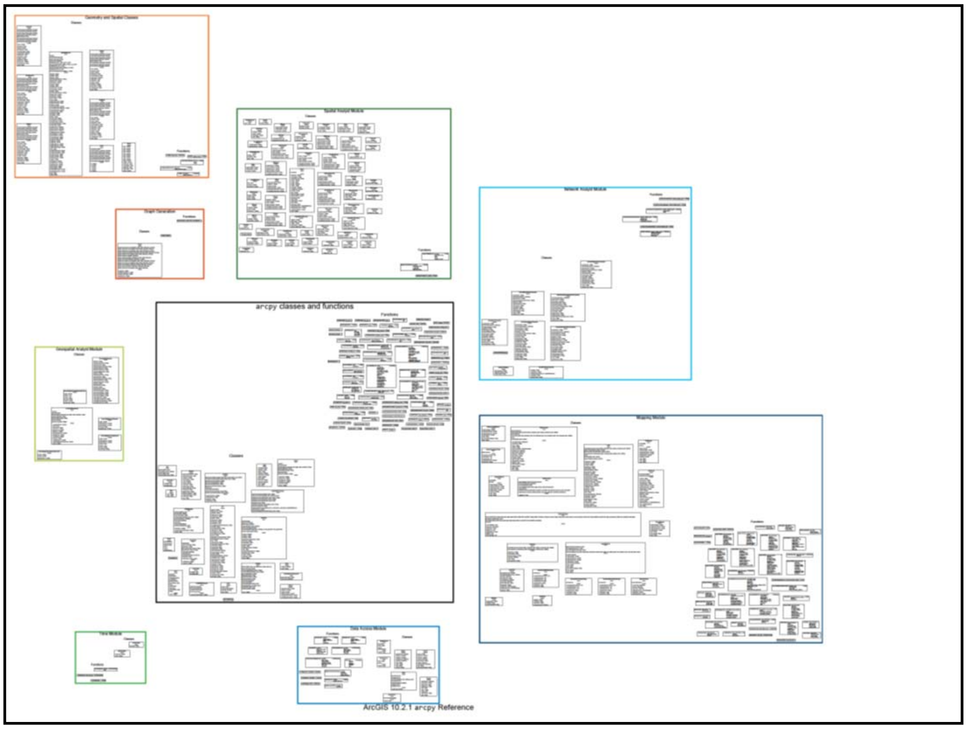
▪— Property Get
▪—▪ Property Get/Put
◄— Method

ValueTable***

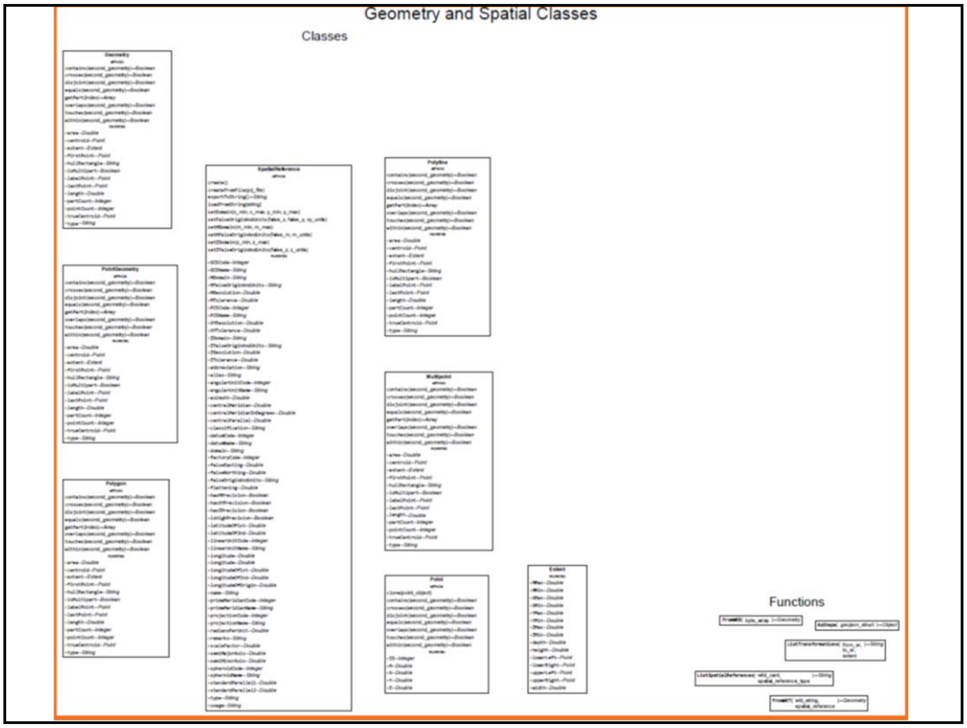
- ▪— RowCount
- ▪— ColumnCount
- ◄— AddRow(optional value)
- ◄— GetRow(rowIndex)
- ◄— GetValue(rowIndex, columnIndex)
- ◄— LoadFromString(value)
- ◄— ExportToString
- ◄— RemoveRow(rowIndex)
- ◄— SetRow(rowIndex, value)
- ◄— SetColumns(value)
- ◄— SetValue(rowIndex, columnIndex)

UpdateCursor

- ◄— Next: Object
- ◄— Reset
- ◄— UpdateRow(Object)
- ◄— DeleteRow(Object)



ArcGIS 10.2.1 arcpy Reference



Geometry and Spatial Classes

Classes

Geometry
methods
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
getPart(index)—Array
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-area—Double
-centroid—Point
-extent—Extent
-firstPoint—Point
-hullRectangle—String
-isMultipart—Boolean
-labelPoint—Point
-lastPoint—Point
-length—Double
-partCount—Integer
-pointCount—Integer
-trueCentroid—Point
-type—String

PointGeometry
methods
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
getPart(index)—Array
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-area—Double
-centroid—Point
-extent—Extent
-firstPoint—Point
-hullRectangle—String
-isMultipart—Boolean
-labelPoint—Point
-lastPoint—Point
-length—Double
-partCount—Integer
-pointCount—Integer
-trueCentroid—Point
-type—String

Polygon
methods
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
getPart(index)—Array
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-area—Double
-centroid—Point
-extent—Extent
-firstPoint—Point
-hullRectangle—String
-isMultipart—Boolean
-labelPoint—Point
-lastPoint—Point
-length—Double
-partCount—Integer
-pointCount—Integer
-trueCentroid—Point
-type—String

SpatialReference
methods
create()
createFromFile(prj_file)
exportToString()—String
loadFromString(string)
setDomain(x_min, x_max, y_min, y_max)
setFalseOriginAndUnits(false_x, false_y, xy_units)
setMDomain(m_min, m_max)
setMFalseOriginAndUnits(false_m, m_units)
setZDomain(z_min, z_max)
setZFalseOriginAndUnits(false_z, z_units)
properties
-GCSCode—Integer
-GCSName—String
-MDomain—String
-MFalseOriginAndUnits—String
-MResolution—Double
-MTolerance—Double
-PCSCode—Integer
-PCSName—String
-XYResolution—Double
-XYTolerance—Double
-ZDomain—String
-ZFalseOriginAndUnits—String
-ZResolution—Double
-ZTolerance—Double
-abbreviation—String
-alias—String
-angularUnitCode—Integer
-angularUnitName—String
-azimuth—Double
-centralMeridian—Double
-centralMeridianInDegrees—Double
-centralParallel—Double
-classification—String
-datumCode—Integer
-datumName—String
-domain—String
-factoryCode—Integer
-falseEasting—Double
-falseNorthing—Double
-falseOriginAndUnits—String
-flattening—Double
-hasMPrecision—Boolean
-hasXYPrecision—Boolean
-hasZPrecision—Boolean
-isHighPrecision—Boolean
-latitudeOf1st—Double
-latitudeOf2nd—Double
-linearUnitCode—Integer
-linearUnitName—String
-longitude—Double
-longitude—Double
-longitudeOf1st—Double
-longitudeOf2nd—Double
-longitudeOfOrigin—Double
-name—String
-primeMeridianCode—Integer
-primeMeridianName—String
-projectionCode—Integer
-projectionName—String
-radiansPerUnit—Double
-remarks—String
-scaleFactor—Double
-semiMajorAxis—Double
-semiMinorAxis—Double
-spheroidCode—Integer
-spheroidName—String
-standardParallel1—Double
-standardParallel2—Double
-type—String
-usage—String

Polyline
methods
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
getPart(index)—Array
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-area—Double
-centroid—Point
-extent—Extent
-firstPoint—Point
-hullRectangle—String
-isMultipart—Boolean
-labelPoint—Point
-lastPoint—Point
-length—Double
-partCount—Integer
-pointCount—Integer
-trueCentroid—Point
-type—String

Multipoint
methods
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
getPart(index)—Array
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-area—Double
-centroid—Point
-extent—Extent
-firstPoint—Point
-hullRectangle—String
-isMultipart—Boolean
-labelPoint—Point
-lastPoint—Point
-length—Double
-partCount—Integer
-pointCount—Integer
-trueCentroid—Point
-type—String

Point
methods
clone(point_object)
contains(second_geometry)—Boolean
crosses(second_geometry)—Boolean
disjoint(second_geometry)—Boolean
equals(second_geometry)—Boolean
overlaps(second_geometry)—Boolean
touches(second_geometry)—Boolean
within(second_geometry)—Boolean
properties
-ID—Integer
-M—Double
-X—Double
-Y—Double
-Z—Double

Extent
properties
-MMax—Double
-MMin—Double
-XMax—Double
-XMin—Double
-YMax—Double
-YMin—Double
-ZMax—Double
-ZMin—Double
-depth—Double
-height—Double
-lowerLeft—Point
-lowerRight—Point
-upperLeft—Point
-upperRight—Point
-width—Double

Functions

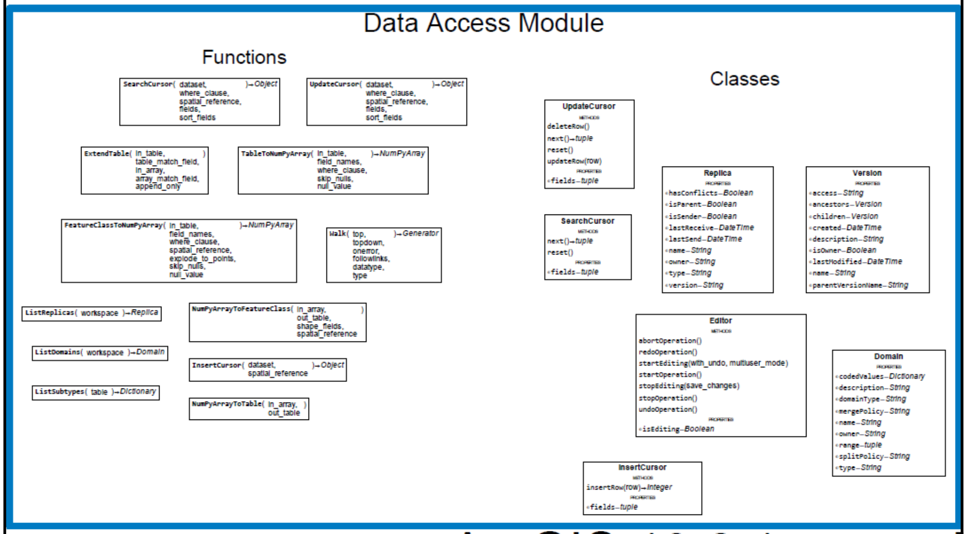
FromWKB(byte_array)—Geometry

AsShape(geojson_struct)—Object

ListTransformations(from_sr, to_sr, extent)—String

ListSpatialReferences(wild_card, spatial_reference_type)—String

FromWKT(wkt_string, spatial_reference)—Geometry



Data Access Module

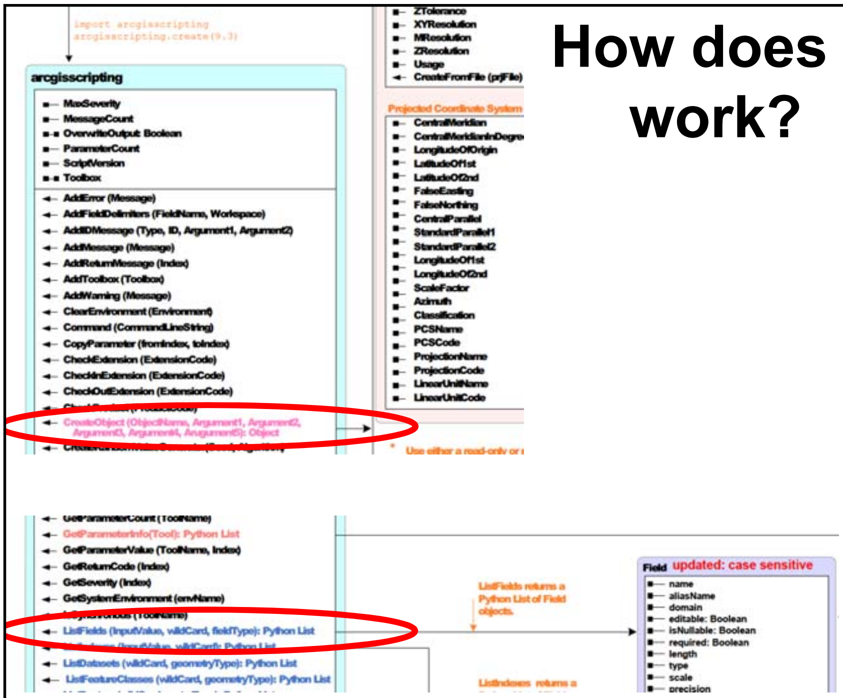
Functions

```
SearchCursor( dataset,                 )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields

UpdateCursor( dataset,                 )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields

ExtendTable( in_table,            )
             table_match_field,
             in_array,
             array_match_field,
             append_only

TableToNumPyArray( in_table,        )→NumPyArray
                   field_names,
                   where_clause,
                   skip_nulls,
                   null_value

FeatureClassToNumPyArray( in_table,           )→NumPyArray
                          field_names,
                          where_clause,
                          spatial_reference,
                          explode_to_points,
                          skip_nulls,
                          null_value

Walk( top,        )→Generator
      topdown,
      onerror,
      followlinks,
      datatype,
      type

ListReplicas( workspace )→Replica

NumPyArrayToFeatureClass( in_array,          )
                          out_table,
                          shape_fields,
                          spatial_reference

ListDomains( workspace )→Domain

InsertCursor( dataset,             )→Object
              spatial_reference

ListSubtypes( table )→Dictionary

NumPyArrayToTable( in_array,  )
                   out_table
```

Classes

```
UpdateCursor
  METHODS
  deleteRow()
  next()→tuple
  reset()
  updateRow(row)
  PROPERTIES
  ◦fields→tuple

SearchCursor
  METHODS
  next()→tuple
  reset()
  PROPERTIES
  ◦fields→tuple

Replica
  PROPERTIES
  ◦hasConflicts→Boolean
  ◦isParent→Boolean
  ◦isSender→Boolean
  ◦lastReceive→DateTime
  ◦lastSend→DateTime
  ◦name→String
  ◦owner→String
  ◦type→String
  ◦version→String

Version
  PROPERTIES
  ◦access→String
  ◦ancestors→Version
  ◦children→Version
  ◦created→DateTime
  ◦description→String
  ◦isOwner→Boolean
  ◦lastModified→DateTime
  ◦name→String
  ◦parentVersionName→String

Editor
  METHODS
  abortOperation()
  redoOperation()
  startEditing(with_undo, multiuser_mode)
  startOperation()
  stopEditing(save_changes)
  stopOperation()
  undoOperation()
  PROPERTIES
  ◦isEditing→Boolean

Domain
  PROPERTIES
  ◦codedValues→Dictionary
  ◦description→String
  ◦domainType→String
  ◦mergePolicy→String
  ◦name→String
  ◦owner→String
  ◦range→tuple
  ◦splitPolicy→String
  ◦type→String

InsertCursor
  METHODS
  insertRow(row)→Integer
  PROPERTIES
  ◦fields→tuple
```

How does it work?



```
import arcgisscripting
arcgisscripting.create(9.3)
```

arcgisscripting

- MaxSeverity
- MessageCount
- OverwriteOutput: Boolean
- ParameterCount
- ScriptVersion
- Toolbox
- AddError (Message)
- AddFieldDelimiters (FieldName, Workspace)
- AddIDMessage (Type, ID, Argument1, Argument2)
- AddMessage (Message)
- AddReturnMessage (Index)
- AddToolbox (Toolbox)
- AddWarning (Message)
- ClearEnvironment (Environment)
- Command (CommandLineString)
- CopyParameter (fromIndex, toIndex)
- CheckExtension (ExtensionCode)
- CheckInExtension (ExtensionCode)
- CheckOutExtension (ExtensionCode)
- CreateObject (ObjectName, Argument1, Argument2, Argument3, Argument4, Arugument5): Object
- GetParameterInfo(Tool): Python List
- GetParameterValue (ToolName, Index)
- GetReturnCode (Index)
- GetSeverity (Index)
- GetSystemEnvironment (envName)
- ListFields (InputValue, wildCard, fieldType): Python List
- ListDatasets (wildCard, geometryType): Python List
- ListFeatureClasses (wildCard, geometryType): Python List

- ZTolerance
- XYResolution
- MResolution
- ZResolution
- Usage
- CreateFromFile (prjFile)

Projected Coordinate System

- CentralMeridian
- CentralMeridianInDegre
- LongitudeOfOrigin
- LatitudeOf1st
- LatitudeOf2nd
- FalseEasting
- FalseNorthing
- CentralParallel
- StandardParallel1
- StandardParallel2
- LongitudeOf1st
- LongitudeOf2nd
- ScaleFactor
- Azimuth
- Classification
- PCSName
- PCSCode
- ProjectionName
- ProjectionCode
- LinearUnitName
- LinearUnitCode

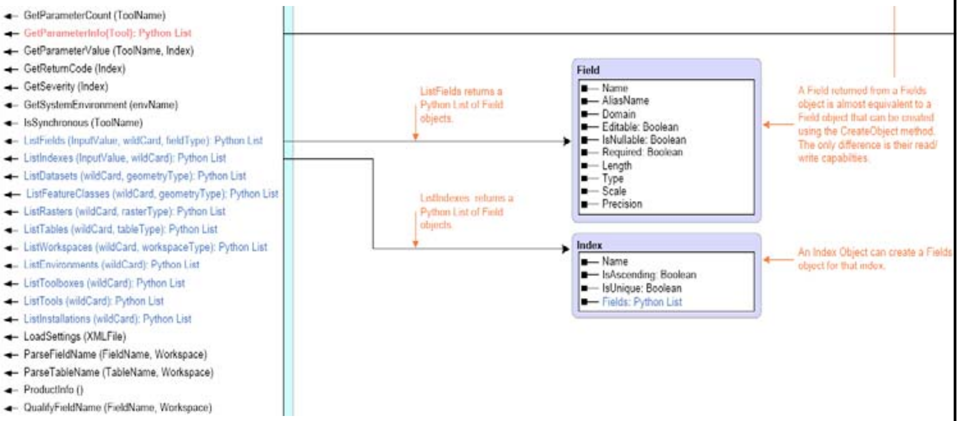
* Use either a read-only or

ListFields returns a Python List of Field objects.

ListIndexes returns a

Field updated: case sensitive

- name
- aliasName
- domain
- editable: Boolean
- isNullable: Boolean
- required: Boolean
- length
- type
- scale
- precision

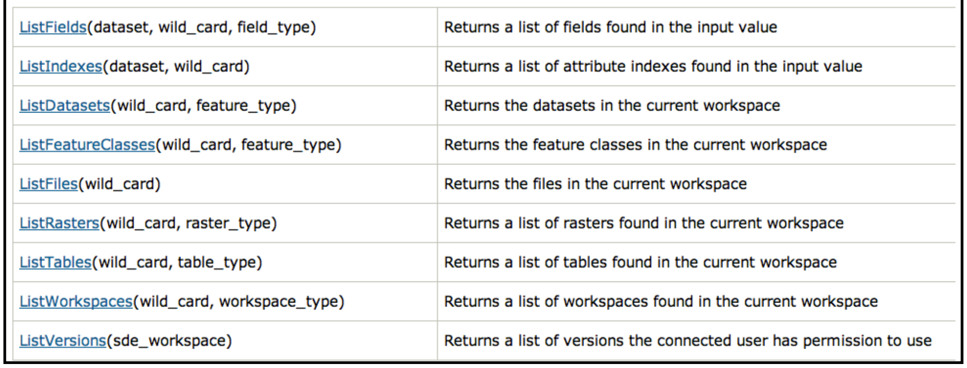

GetParameterCount (ToolName)
GetParameterInfo(Tool): Python List
GetParameterValue (ToolName, Index)
GetReturnCode (Index)
GetSeverity (Index)
GetSystemEnvironment (envName)
IsSynchronous (ToolName)
ListFields (InputValue, wildCard, fieldType): Python List
ListIndexes (InputValue, wildCard): Python List
ListDatasets (wildCard, geometryType): Python List
ListFeatureClasses (wildCard, geometryType): Python List
ListRasters (wildCard, rasterType): Python List
ListTables (wildCard, tableType): Python List
ListWorkspaces (wildCard, workspaceType): Python List
ListEnvironments (wildCard): Python List
ListToolboxes (wildCard): Python List
ListTools (wildCard): Python List
ListInstallations (wildCard): Python List
LoadSettings (XMLFile)
ParseFieldName (FieldName, Workspace)
ParseTableName (TableName, Workspace)
ProductInfo ()
QualifyFieldName (FieldName, Workspace)
ListFields returns a Python List of Field objects.
ListIndexes returns a Python List of Field objects.
Field
Name
AliasName
Domain
Editable: Boolean
IsNullable: Boolean
Required: Boolean
Length
Type
Scale
Precision
A Field returned from a Fields object is almost equivalent to a Field object that can be created using the CreateObject method. The only difference is their read/write capabilities.
Index
Name
IsAscending: Boolean
IsUnique: Boolean
Fields: Python List
An Index Object can create a Fields object for that index.

ListFeatureClasses and ListFields return a Py List
Let's see what is in these lists…

Run (quickly) class03_04 ListGP_Py

Cataloguing & Listing Spatial Data

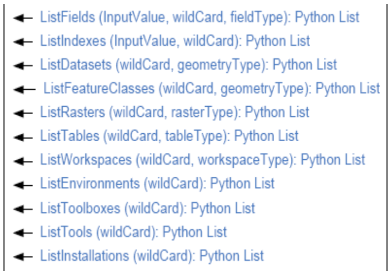
- Important task category to support **Batch Processing**
- Return List objects
- Organizing and cataloguing different data types by using List methods (filtering)



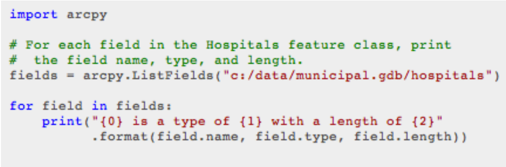
ListFields(dataset, wild_card, field_type)	Returns a list of fields found in the input value
ListIndexes(dataset, wild_card)	Returns a list of attribute indexes found in the input value
ListDatasets(wild_card, feature_type)	Returns the datasets in the current workspace
ListFeatureClasses(wild_card, feature_type)	Returns the feature classes in the current workspace
ListFiles(wild_card)	Returns the files in the current workspace
ListRasters(wild_card, raster_type)	Returns a list of rasters found in the current workspace
ListTables(wild_card, table_type)	Returns a list of tables found in the current workspace
ListWorkspaces(wild_card, workspace_type)	Returns a list of workspaces found in the current workspace
ListVersions(sde_workspace)	Returns a list of versions the connected user has permission to use

Batch (Geo-)Processing

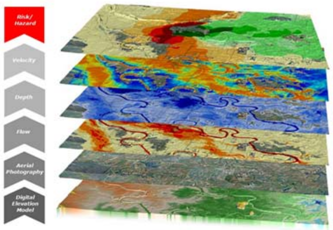
- Batch programming is one typical structure of programs
- Automating repetitive tasks
- Amounts of inputs and outputs to be organized for geoprocessing

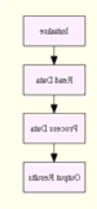


ListFields (InputValue, wildCard, fieldType): Python List
ListIndexes (InputValue, wildCard): Python List
ListDatasets (wildCard, geometryType): Python List
ListFeatureClasses (wildCard, geometryType): Python List
ListRasters (wildCard, rasterType): Python List
ListTables (wildCard, tableType): Python List
ListWorkspaces (wildCard, workspaceType): Python List
ListEnvironments (wildCard): Python List
ListToolboxes (wildCard): Python List
ListTools (wildCard): Python List
ListInstallations (wildCard): Python List



```
import arcpy

# For each field in the Hospitals feature class, print
#  the field name, type, and length.
fields = arcpy.ListFields("c:/data/municipal.gdb/hospitals")

for field in fields:
    print("{0} is a type of {1} with a length of {2}"
          .format(field.name, field.type, field.length))
```





Details of class03_4_GPLists.py

Parameters as Filters for List Methods

- **Wild card**: Restrict the objects and datasets to be inserted into the list by **name** using an **asterisk (*)**(name filter)

```
myForestList = arcpy.ListFeatureClasses("F*")
myForestList = arcpy.ListFeatureClasses("*forest*")
```

- **Type parameter:** Data property restrictions using type keywords

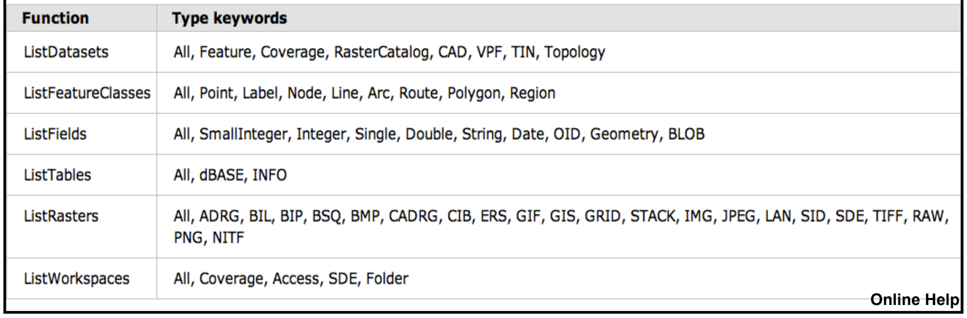
```
myForestList = arcpy.ListFeatureClasses("*F*", "polygon")
myTables = arcpy.ListFields(table,"G*","Integer")
```

“Filtering” the datasets in your directory

class03_4_GPLists.py

Type Filters for List methods

- Default behavior for List methods is to list all **supported** types
- Type keywords restrict the list to a specific type (**type filter**)

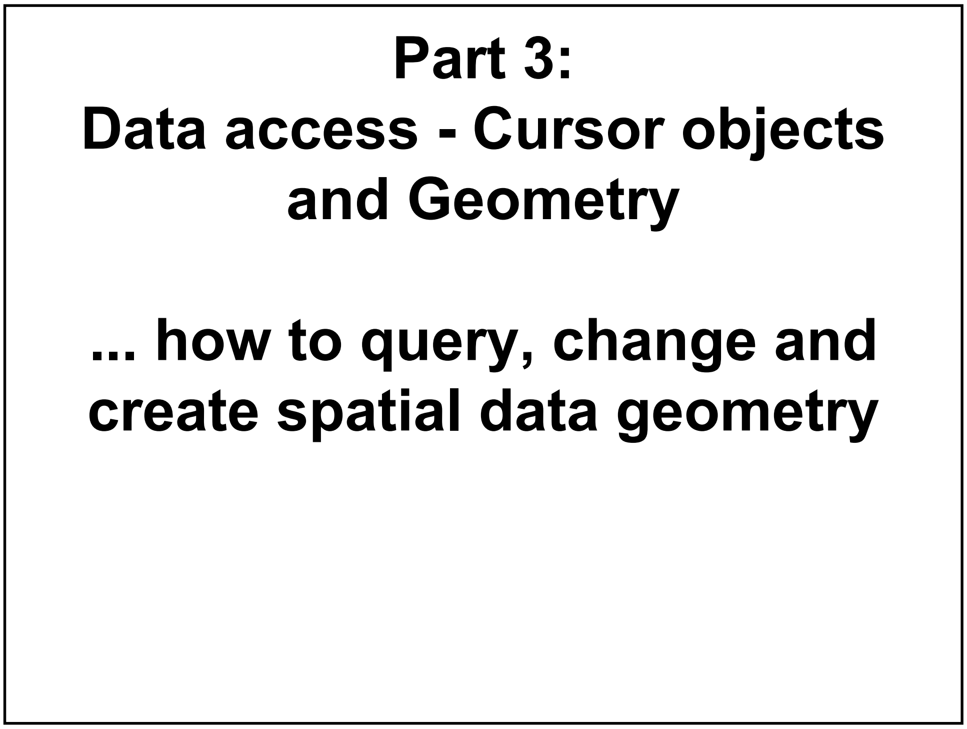


Function	Type keywords
ListDatasets	All, Feature, Coverage, RasterCatalog, CAD, VPF, TIN, Topology
ListFeatureClasses	All, Point, Label, Node, Line, Arc, Route, Polygon, Region
ListFields	All, SmallInteger, Integer, Single, Double, String, Date, OID, Geometry, BLOB
ListTables	All, dBASE, INFO
ListRasters	All, ADRG, BIL, BIP, BSQ, BMP, CADRG, CIB, ERS, GIF, GIS, GRID, STACK, IMG, JPEG, LAN, SID, SDE, TIFF, RAW, PNG, NITF
ListWorkspaces	All, Coverage, Access, SDE, Folder

Online Help

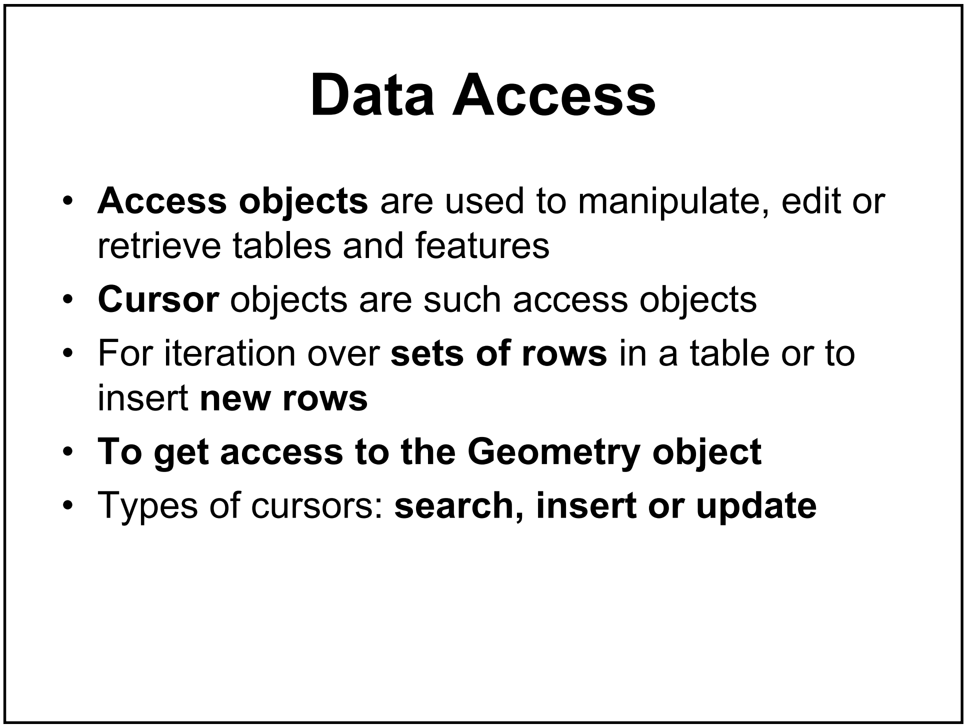
Summary Part II

- The Geoprocessing environment offers sophisticated functionalities for **cataloguing and organizing** spatial data using “**list**”- (and “describe”) methods that allow to filter datasets by names and types
- Batch processing can be done very efficiently for **automated** geoprocessing using list constructs (**enumerations**)
- Python offers suitable **built-in** functions to complement more complex tasks in batching (such as String manipulation)



Part 3:
Data access - Cursor objects and Geometry

... how to query, change and create spatial data geometry



Data Access

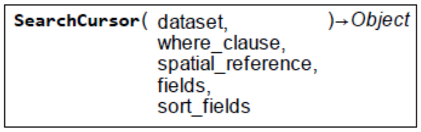
- **Access objects** are used to manipulate, edit or retrieve tables and features
- **Cursor** objects are such access objects
- For iteration over **sets of rows** in a table or to insert **new rows**
- **To get access to the Geometry object**
- Types of cursors: **search, insert or update**

Cursor Methods

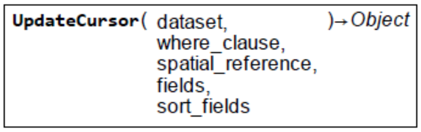
New Data Access Module

Arcpy.da.SearchCursor()

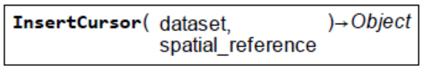
- **SearchCursor**()
  Retrieve rows of a table
- **UpdateCursor**()
  Update and delete rows
- **InsertCursor**()
  Insert rows into a table/feature class & create features



```
SearchCursor( dataset,             )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields
```



```
UpdateCursor( dataset,             )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields
```

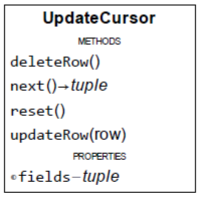


```
InsertCursor( dataset,             )→Object
              spatial_reference
```

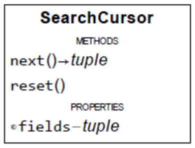
Show table

Cursor Objects

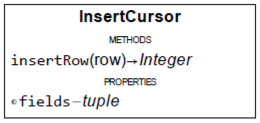
- Cursor methods create **Cursor Objects** ("containers with content")
- These **Cursor** objects contain data for as many rows as the table has
- Navigation in **forward** direction only
- All rows have the same "**ordered set**" of **fields** defined with the cursor methods (subsets)
- Methods that can be used **vary** depending on the **cursor type**



```
UpdateCursor
METHODS
deleteRow()
next()→tuple
reset()
updateRow(row)
PROPERTIES
◦fields–tuple
```

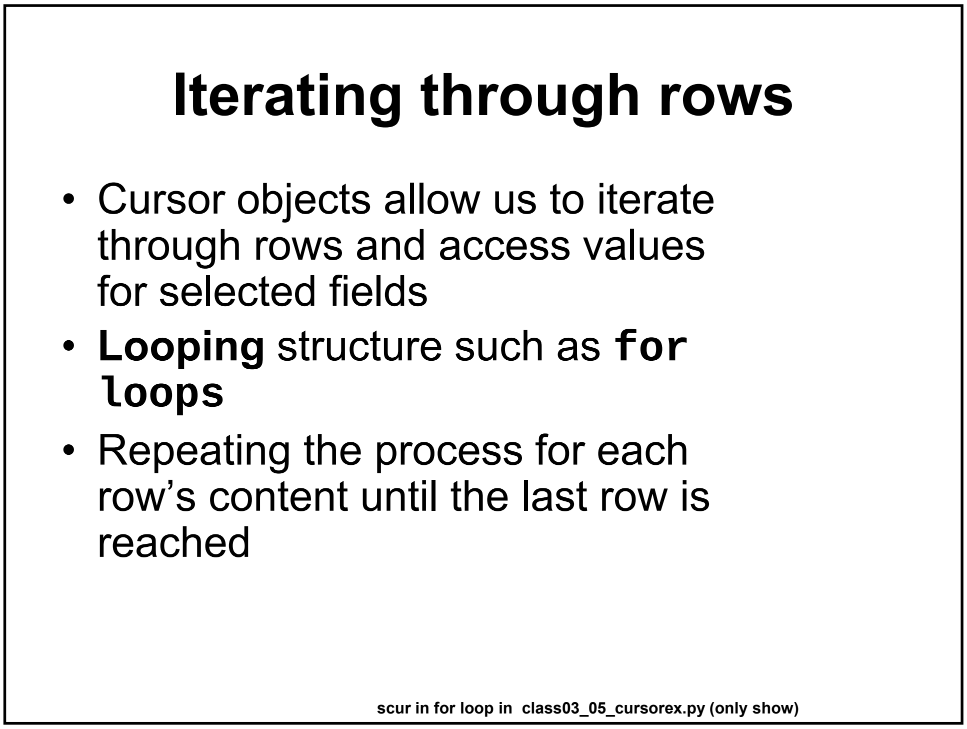


```
SearchCursor
METHODS
next()→tuple
reset()
PROPERTIES
◦fields–tuple
```



```
InsertCursor
METHODS
insertRow(row)→Integer
PROPERTIES
◦fields–tuple
```

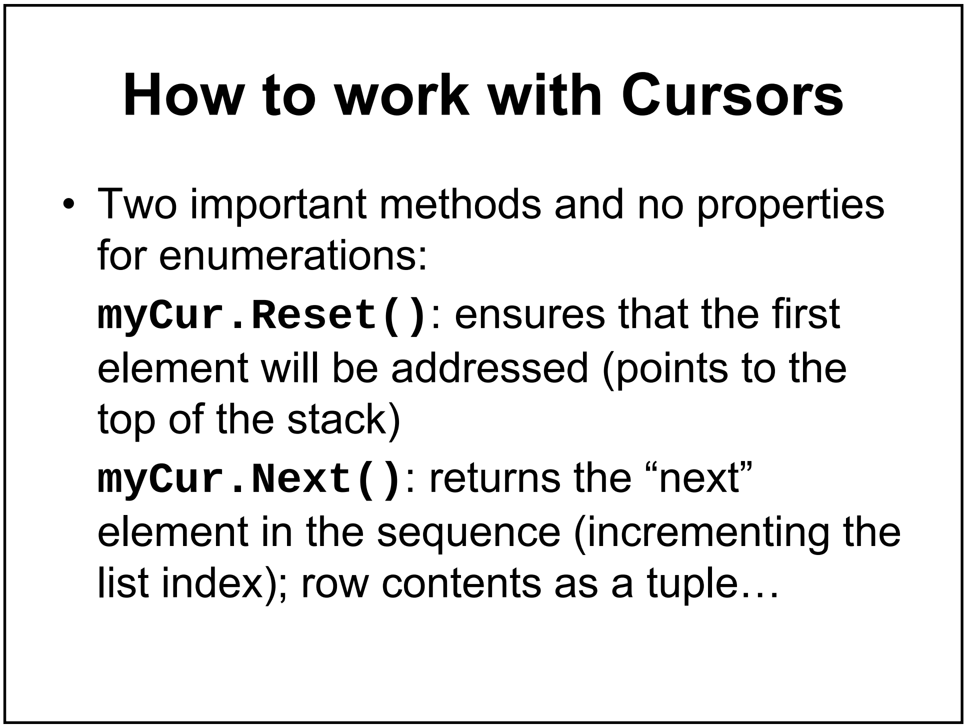
Start on class03_05_cursorex.py



Iterating through rows

- Cursor objects allow us to iterate through rows and access values for selected fields
- **`Looping`** structure such as **`for loops`**
- Repeating the process for each row’s content until the last row is reached

scur in for loop in class03_05_cursorex.py (only show)



How to work with Cursors

- Two important methods and no properties for enumerations:

  `myCur.Reset()`: ensures that the first element will be addressed (points to the top of the stack)

  `myCur.Next()`: returns the “next” element in the sequence (incrementing the list index); row contents as a tuple…

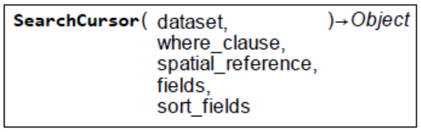
The Row Content

- The row contents are written into sequences (Python objects) through .Next() cursor methods
- Search Cursor: Tuples (cannot be changed)
- UpdateCursor: Lists (can be changed on the fly)
- **This is basically the Get/Put** mechanism for field values (**SearchCursor**: **only Get** property)

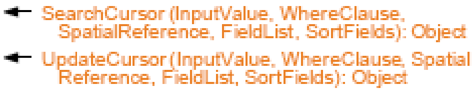
scur interactive (no geom) in class03_05_cursorex.py (run loop)

SQL Queries

- **Update** and **Search** cursors
- **Filtering** of data in a table using queries
- **SELECT WHERE**

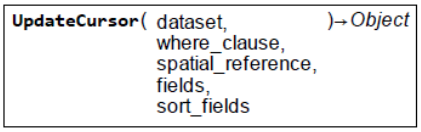


```
SearchCursor( dataset,              )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields
```



← SearchCursor (InputValue, WhereClause, SpatialReference, FieldList, SortFields): Object

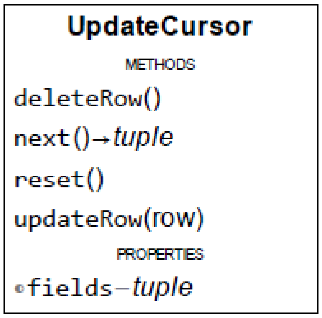
← UpdateCursor(InputValue, WhereClause, Spatial Reference, FieldList, SortFields): Object



```
UpdateCursor( dataset,              )→Object
              where_clause,
              spatial_reference,
              fields,
              sort_fields
```

UpdateCursor()

- **UpdateRow()** at **current position** of an update cursor
- **DeleteRow()** at the current position of an update cursor
- The row that was last called by .Next() is deleted



UpdateCursor

METHODS

deleteRow()

next()→*tuple*

reset()

updateRow(row)

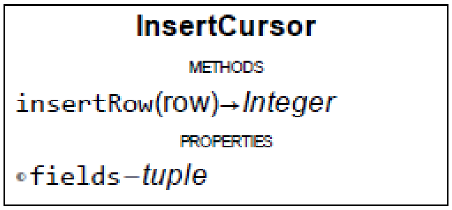
PROPERTIES

∘fields–*tuple*

ucur in for loop & interactive (no geom) in class03_05_cursorex.py

InsertCursor()

- **InsertRow()**takes a row as argument and inserts the row with defined field values into the table



InsertCursor

METHODS

insertRow(row)→*Integer*

PROPERTIES

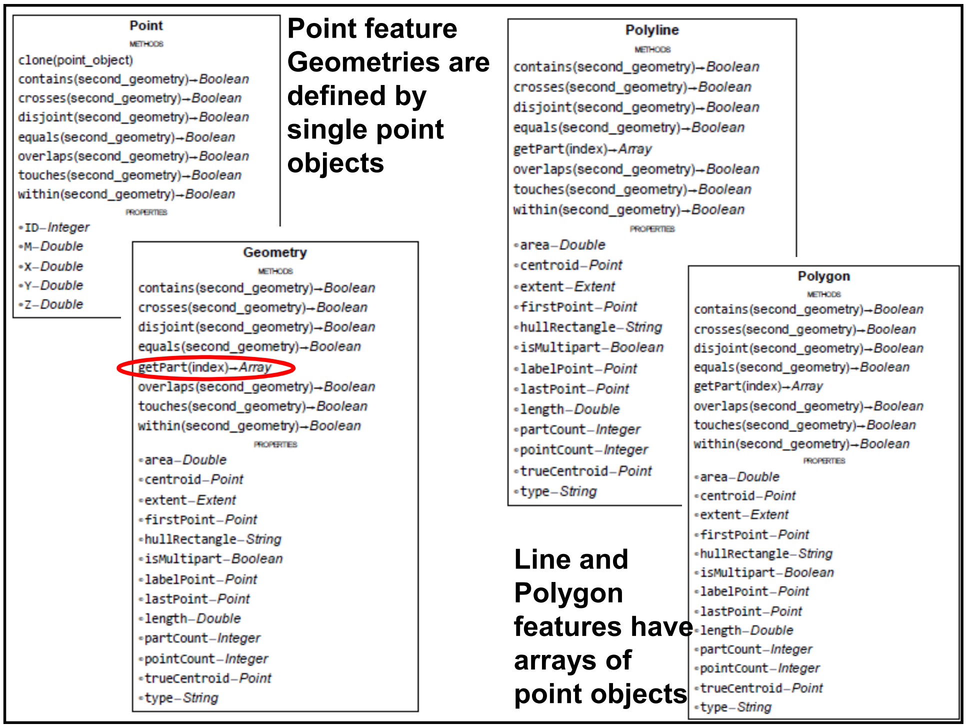
∘fields–*tuple*

Geometry Object

- Accessing **feature geometry**
- Geometry type field of feature classes:
  Actual geometry of the feature (“**Shape**”)
- Geometry Object created by specifying “SHAPE@” when creating the cursor object
- Geometrical properties of the feature can be exposed
- Geometry tokens: SHAPE@XY, SHAPE@AREA

http://pro.arcgis.com/en/pro-app/arcpy/get-started/reading-geometries.htm

scur interactive for geom properties in class03_05_cursorex.py



Point

METHODS

clone(point_object)
contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- ID–*Integer*
- M–*Double*
- X–*Double*
- Y–*Double*
- Z–*Double*

Point feature Geometries are defined by single point objects

Geometry

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Polyline

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Polygon

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

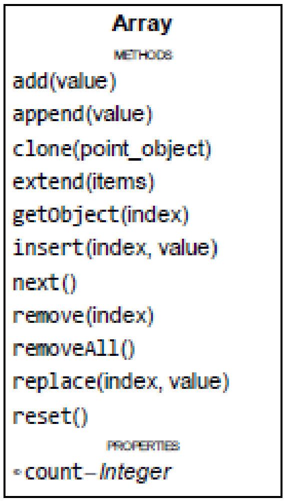
PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Line and Polygon features have arrays of point objects

Array Objects

- Arrays are the containers for point objects and their properties
- Primitives of the feature geometry



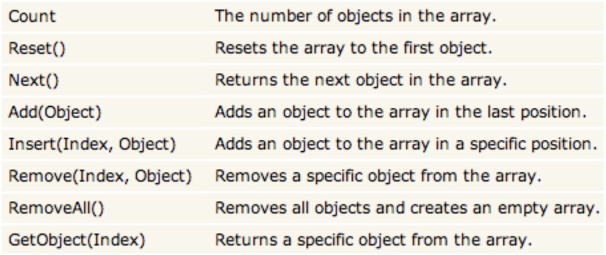
Array

METHODS

add(value)
append(value)
clone(point_object)
extend(items)
getObject(index)
insert(index, value)
next()
remove(index)
removeAll()
replace(index, value)
reset()

PROPERTIES

- count – *Integer*



Count	The number of objects in the array.
Reset()	Resets the array to the first object.
Next()	Returns the next object in the array.
Add(Object)	Adds an object to the array in the last position.
Insert(Index, Object)	Adds an object to the array in a specific position.
Remove(Index, Object)	Removes a specific object from the array.
RemoveAll()	Removes all objects and creates an empty array.
GetObject(Index)	Returns a specific object from the array.

Reading Geometries 1

- Each **feature** (each row) is defined by **points** (vertices or single coordinates)
- Accessing these points by using **geometry** objects
- “**Array**” of **point objects** through `.GetPart()`
- OR: just points / multi-points
- Multi-Parts consist of several objects
- Example Hawaii: The geometry of it has several polygons - several arrays describe one object
- `GetPart(0), … , GetPart(1)`

```
# Reading out the point geometry properties for checking
cursor = arcpy.da.SearchCursor(theme,["SHAPE@","MyField"])
row = cursor.next()
pnt=row[0]
p=pnt.getPart(0)
print p.X, p.Y
```

scur test for ex02_pointmaker.py

scur interactive for geom array in class03_05_cursorex.py