DevSummit DC

February 11, 2015 | Washington, DC

ArcGIS for Desktop: Advanced Python Topics

Andrew Chapkowski & James Tedrick

Topics

- Python Data structures
- Geometry objects
- Data Access
- Python Toolboxes
- Lambda Functions
- Multiprocessing
- External Modules
  - Numpy
  - ArcREST

Key Python data structures

- Lists
  - Flexible
  - Ordered
- Tuples
  - Immutable
  - Ordered
- Dictionary
  - Key/value pairs

```
l = ['10 feet', '20 feet', '50 feet']

t = ('Thurston', 'Pierce', 'King')

d = {'ProductName': 'desktop',
     'InstallDir': 'c:\\ArcGIS\\Desktop10.1'}
```

List comprehension

- **Compact way of mapping a list into another**

```
>>> distances = [10, 50, 200]
>>> new_distances = ['{} feet'.format(d) for d in distances]
>>> print(new_distances)
['10 feet', '50 feet', '200 feet']


>>> field_names = [f.name for f in arcpy.ListFields(table)]
>>> print(field_names)
[u'OBJECTID', u'NAME', u'ADDRESS']
```

Defining Functions

- **Organize and re-use functionality**

```
import arcpy

def increase_extent(extent, factor):
    """Increases the extent by the given factor"""

    XMin = extent.XMin - (factor * extent.XMin)
    YMin = extent.YMin - (factor * extent.YMin)
    XMax = extent.XMax + (factor * extent.XMax)
    YMax = extent.YMax + (factor * extent.YMax)

    return arcpy.Extent(XMin, YMin, XMax, YMax)

oldExtent = arcpy.Describe("boundary").extent
newExtent = increase_extent(oldExtent, .1)
```

Define your function

Return a result

Call the function

Python Classes

- Programming based on objects
- Objects can have state (data) and behavior (functions)
- Classes are templates for creating objects

Python Classes

- Keyword `self` – access to the object (every function in a class has this as a parameter)
- Function `__init__()` – startup function when an object is created
- Functions `__iter_()` and `next()` define how an object works in a for statement

```
class PortalUser():
    portal = None
    username = None
    password = None
    token = None
    groups = dict()
    folders = dict()
    def __init__(self, portal, username, password):
        self.portal = portal
        self.username = username
        self.password = password
        self.token = getToken(portal, username, password)
        #self.folders = getFolders(portal, username, self.token)
        #self.groups = getOwnGroups(self)

    def renew(self):
        self.token = getToken(self.portal, self.username, self.password)
```

Geometry and cursors

- Can create geometry in different ways
  - Geometry objects
  - List of coordinates
  - Using other formats
    - JSON, WKT, WKB

```
line = arcpy.Polyline(
    arcpy.Array([arcpy.Point(2,3),
                 arcpy.Point(3,5)]))

line = [(2,3), (3,5)]

line = {
    "type": "LineString",
    "coordinates": [[2, 3], [3,5]]}

cursor.insertRow([line])
```

Working with geometry

- **Relational:**
  - **Is a point within a polygon?**

```
point.within(polygon)
```

- **Topological**
  - **What is the intersection of two geometries?**

```
poly1.intersect(poly2)
```

- **Others (mid-point, geodesic area)**

```
boundary
buffer
clip
contains
convexHull
crosses
difference
disjoint
distanceTo
equals
getArea
getLength
getPart
intersect
overlaps
positionAlongLine
projectAs
symmetricDifference
touches
union
within
```

Cursors

- **Use cursors to access records and features**

SearchCursor	Read-only
UpdateCursor	Update or delete rows
InsertCursor	Insert rows

- **Two varieties**
  - **‘Classic’ cursors**
  - **‘Data access’ cursors (10.1+)**

Cursor mechanics

- **Data access cursors use lists and tuples**
  - **Values are accessed by index**

```
cursor = arcpy.da.InsertCursor(table, ["field1", "field2"])
cursor.insertRow([1, 10])
```

- **Classic cursors use row objects**
  - **Values are accessed by setValue/getValue**

```
cursor = arcpy.InsertCursor(table)
row = cursor.newRow()
row.setValue("field1", 1)
row.setValue("field2", 10)
cursor.insertRow(row)
```

With statements

- arcpy.da Cursors support with statements
- Automatic open/close access to database (no hanging locks)
- Preferred method for data access

```
with arcpy.da.SearchCursor(table, field) as cursor:
    for row in cursor:
        print row[0]
```

Cursor performance

- **Don’t leave cursors open – load into lists/dictionaries**
- **Use only those fields you need**
- **Use tokens for shape information**
  - **Get only what you need**
  - ***Full geometry is expensive***

- 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@ —A geometry object for the feature.
- SHAPE@AREA —A double of the feature's area.
- SHAPE@LENGTH —A double of the feature's length.
- OID@ —The value of the ObjectID field.

Demo: Geometry & Data Access

Python Toolboxes

- Everything is done in Python
  - Easier to create
  - Easier to maintain
- An ASCII file (*.pyt*) that defines a toolbox and tool(s)
- Tools look and behave like any other type of tool

Benefits

- All Python
- Frees you from Desktop
  - Frees you from the Script tool wizard
- Can easily make changes and refresh

Toolbox Structure

A tool does 3 types of work:

- Parameters
- Validation
- Source

```
class Tool(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Tool"
        self.description = ""
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""
        params = None
        return params

    def isLicensed(self):
        """Set whether tool is licensed to execute."""
        return True

    def updateParameters(self, parameters):
        """Modify the values and properties of parameters before internal
        validation is performed.  This method is called whenever a parameter
        has been changed."""
        return

    def updateMessages(self, parameters):
        """Modify the messages created by internal validation for each tool
        parameter.  This method is called after internal validation."""
        return

    def execute(self, parameters, messages):
        """The source code of the tool."""
        return
```

Demo: Python Toolbox

Lambda Functions

- There are two ways to create functions:
  - def and lambda

- Basic use of lambdas
  - anonymous functions
  - callback functions

```
def square_root(x):
    return math.sqrt(x)
```

```
square_root = lambda x: math.sqrt(x)
```

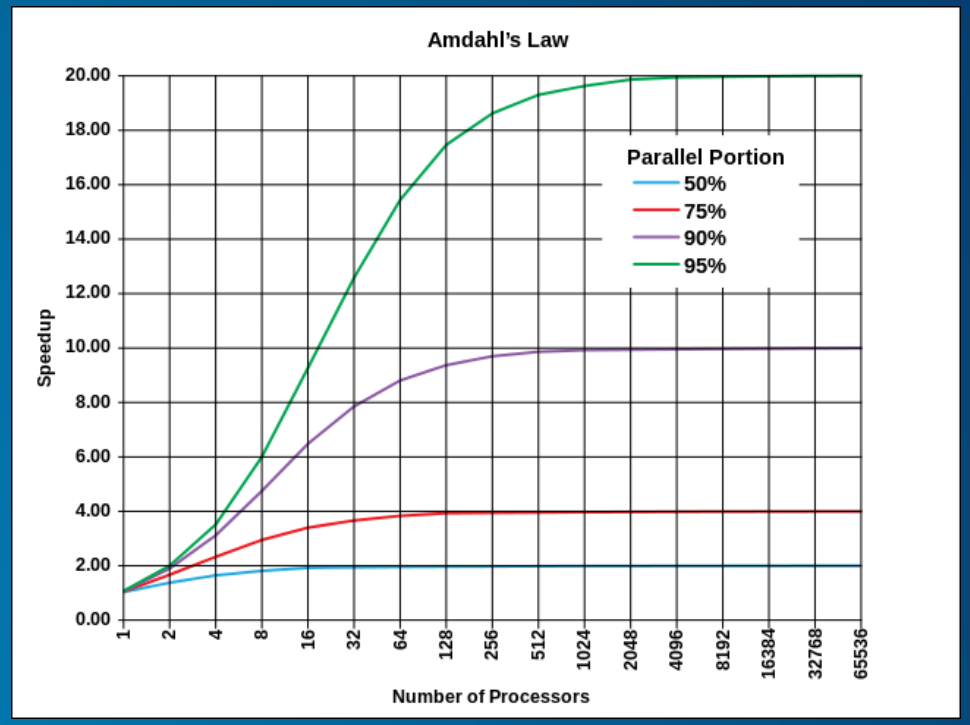
Lambda Functions

- Lambdas are often thought to be confusing
  - Takes a single expression
  - Returns a value even though there is no *return*
  - Used as callback functions in GUI development
    - tKinter, wxPython

```
def by_three(x):
    return x % 3 == 0

func = lambda x: x % 3 == 0

my_list = range(16)
print filter(func, my_list)
print filter(by_three, my_list)

#>>> [0, 3, 6, 9, 12, 15]
#>>> [0, 3, 6, 9, 12, 15]
```

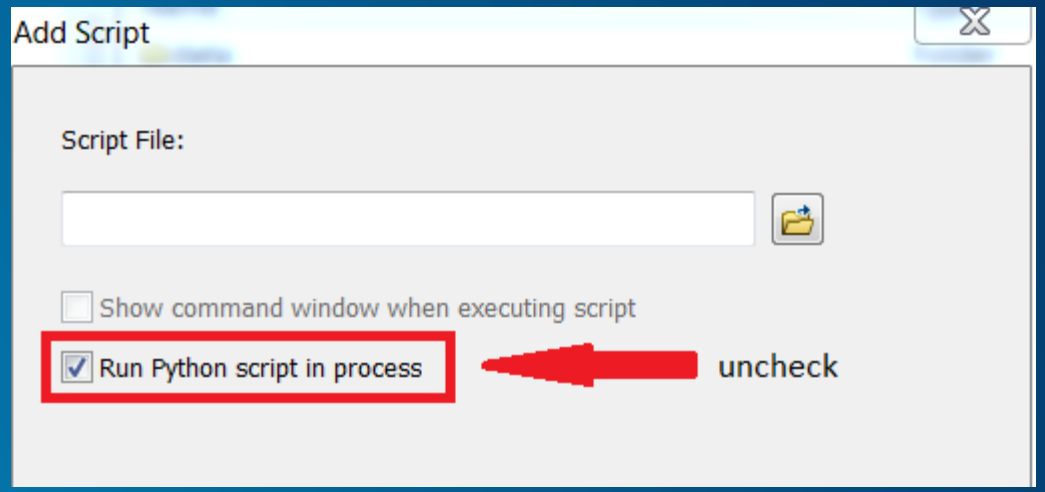
Multiprocessing and Toolboxes

- Multiprocessing is a package that supports the spawning of processes
- Workloads can be divided among multiple python.exe to get work done faster
- Remember Amdahl's Law comes into play.



Multiprocessing and Toolboxes

- **Toolbox Setup**
  - **Must always run in foreground**
  - **Must Work Out of Process**



Demo – Multiprocessing and Data Conversion

```
pool = multiprocessing.Pool(processCount)
for data in dataToConvert:
    jobs.append(pool.apply_async(convertToJSON,
                                 [";".join(data),
                                  outFolder]
                                 ))
    del data
del dataToConvert
pool.close()
pool.join()
results = []
for result in jobs:
    r = str(result.get()).split(';')
    results = results + r
    del r
```

numpy

- Numerical Python (NumPy) is a fundamental package for scientific computing in Python
- Provides a way to perform advanced mathematical operations
- Been in ArcGIS software since 9.2

The numpy Array

- Basic array are the same data type
- Still very powerful, but not very useful for spatial analysis

numpy – Structured Arrays

Array of records of differing data types!

NAME	AGE	WEIGHT
Tommy	40	176
Jane	30	130
Sam	20	209
Bob	9	155
Bill	34	125
Tommy	55	210

```
from numpy import dtype, empty
import numpy as np
frmt = dtype(
    [
        ('NAME', 'S15'),
        ('AGE', int),
        ('WEIGHT', float)
    ]
)
array = empty(6, dtype=frmt)

array['NAME'] = ['Tommy', 'Jane', 'Sam',
                 'Bob', 'Bill', 'Tommy']
array['AGE'] = [40, 30, 20, 9, 34, 55]
array['WEIGHT'] = [176, 130, 209,
                   155, 125, 210]
print array

#>>> [('Tommy', 40, 176.0) ('Jane', 30, 130.0) ('Sam', 20, 209.0)
#     ('Bob', 9, 155.0) ('Bill', 34, 125.0) ('Tommy', 55, 210.0)]
```

Working with Feature Classes as numpy Arrays

- Converts spatial data into structured arrays using the arcpy.da functions:
  - FeatureClassToNumPyArray
  - TableToNumPyArray

- Common tasks:
  - Adding a column or field to a feature class
  - Sorting Values
  - Getting Unique list of field values

Common numpy Tasks

Sorting Array By Column

```
inds = np.argsort(array['NAME'])
np.take(array, inds, out=array)
print array
```

Finding Unique Elements

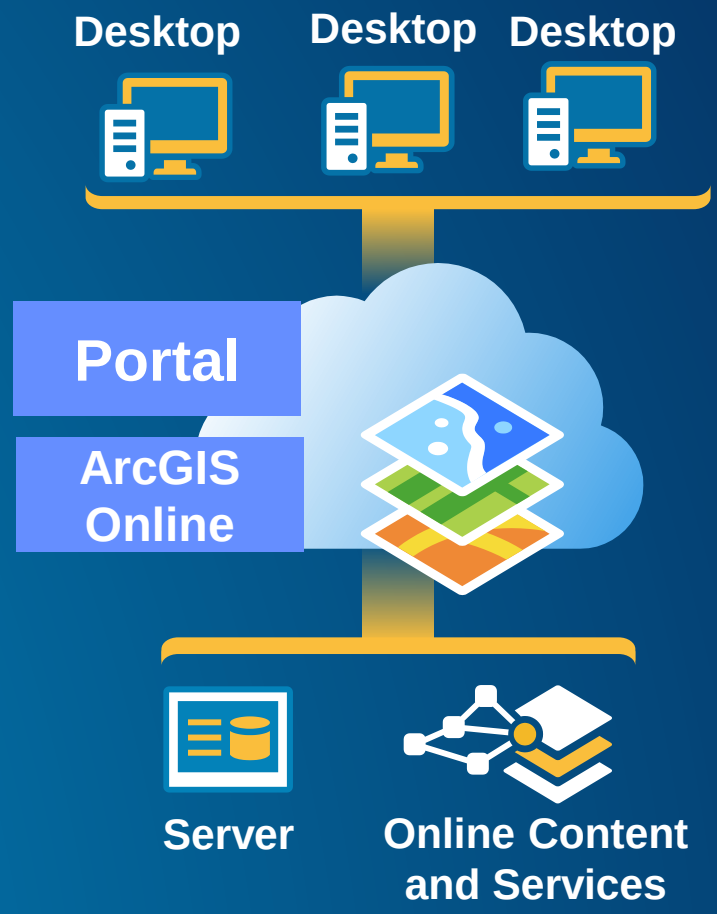
```
unique_elements, indices = np.unique(array[:]['NAME'],
                                     return_index = True)
print unique_elements
# >>> ['Bill' 'Bob' 'Jane' 'Sam' 'Tommy']
```

Demo: Adding Fields to Feature Classes

```
if __name__ == "__main__":
    fc = r"c:\temp\scratch.gdb\SmallAOI"
    flds = arcpy.Describe(fc).fields
    for fld in flds:
        print fld.name
    narray = numpy.array([],
                         numpy.dtype([('_ID', numpy.int),
                                      ('WELL_ID', numpy.int),
                                      ('DESC', '|S100'),
                                      ('DEPTH', numpy.float),
                                      ])
                         )
    arcpy.da.ExtendTable(fc, "OID@", narray, "_ID")
    flds = arcpy.Describe(fc).fields
    print '_____'
    for fld in flds:
        print fld.name
```

ArcGIS REST API

- **Using the ArcGIS REST API, you can:**
  - **Consume ready-to-use ArcGIS Online services hosted by Esri**
  - **Consume services published by you or by other organizations**
  - **Publish your own web services**
  - **Create and share items on ArcGIS Online or your own portal**
  - **Configure and automate parts of the ArcGIS system, such as ArcGIS for Server and Portal for ArcGIS**



Python & ArcGIS REST API

- Working with the ArcGIS REST API can allow for the management of many products:
  - ArcGIS Server
  - ArcGIS Online
  - Portal for ArcGIS
- Functions are called via a POST or GET method
  - urllib, urllib2, requests
- Token based security is the standard way to interact with the REST API

Using 3rd Party Library - ArcREST

- ArcREST is a package written in Python 2.7
  - Works with ArcGIS Server, ArcGIS Online and Portal items
  - Open source
  - Download at:
    https://www.github.com/Esri/ArcREST
- Other 3rd party Python REST libraries:
  - https://github.com/Esri/ago-admin-wiki/wiki/Tools

ArcREST Demo: Inserting Records to a Hosted Feature Service

```
url = get_config_value(config_file, "site", "url")
username = get_config_value(config_file, "site", "username")
password = get_config_value(config_file, "site", "password")
flURL = get_config_value(config_file, "site", "urlfl")
addData = get_config_value(config_file, "site", "updatefc")
#   Local Variables
#
prjFC = os.path.join(arcpy.env.scratchGDB, "prjFC")
sr = arcpy.SpatialReference(102100)
#   Logic
#
prjFC = arcpy.Project_management(addData, out_dataset=prjFC, out_coor_system=sr)[0]

sh = arcrest.AGOLTokenSecurityHandler(username, password)
agolFL = arcrest.agol.FeatureLayer(url=flURL, securityHandler=sh)
arcpy.AddMessage("There are %s features" % agolFL.query(returnCountOnly=True)['count'])
arcpy.AddMessage("Adding the features to the feature service")
arcpy.AddMessage("Added %s features" % len(agolFL.addFeatures(fc=prjFC)['addResults']))
```

ArcREST Demo: Adding a Field to a Hosted Feature Service

```
fieldToAdd = {

    "fields" : [
        {
            "name" : "FUNWITHREST",
            "type" : "esriFieldTypeString",
            "alias" : "FUNFUN",
            "sqlType" : "sqlTypeOther", "length" : 50,
            "nullable" : True,
            "editable" : True,
            "domain" : None,
            "defaultValue" : None
        } ]
}
sh = arcrest.AGOLTokenSecurityHandler(username, password)
agolServices = arcrest.hostedservice.Services(url, securityHandler=sh)

for service in agolServices.services:
    try:
        if service.adminServiceInfo['name'].lower() == featureServiceName.lower():
            for lyr in service.layers:
                print '_____'
                print lyr.name
                print 'Add to Field to Feature Layer'
                print lyr.addToDefinition(fieldToAdd)
                print '_____'
                print 'Delete Field from Feature Layer'
                print lyr.deleteFromDefinition(fieldToAdd)
                print '_____'

    except: pass
```

Desktop ▸ Pro: Python 2 ▸ 3

- ArcGIS Pro will use Python 3.4
- Mostly small differences / library reorganization
  - Ex: `print text` vs. `print(text)`
  - Ex: `urllib2` vs. `urlib.request` module
- Definitely possible to write code that will work in both Python 2 and 3
- arcpy has some changes
  - arcpy.mapping has evolved – arcpy.mp
  - A subset of geoprocessing tools will disappear

https://pro.arcgis.com/en/pro-app/arcpy/get-started/python-migration-for-arcgis-pro.htm

Resources

- **Python Toolbox example: https://github.com/tedrick/dcdev2015/tree/master/topN**
- **Using Curors: https://github.com/tedrick/dcdev2015/tree/master/joinShapesToTable**

Questions?

Andrew Chapkowski & James Tedrick

Understanding our world.