Python Performance Tips

Jason Pardy
jpardy@newfoundgeo.com

Local is faster than global

- Slower:

```
import arcpy

with arcpy.da.SearchCursor("C:/data/roads.shp", "shape@length") as rows:
  for row in rows:
    # Do some processing
```

- Faster:

```
import arcpy

def process_roads(shapefile):
     with arcpy.da.SearchCursor(shapefile, "shape@length") as rows:
       for row in rows:
         # Do some processing

if __name__ == "__main__":
     process_roads("c:/data/roads.shp")
```

Eliminate accessing attributes

- Test: Call a square_root() function 100x where the input is a list of 1 million numbers

Started with…	`import math`
	
	`def square_root(nums):`
	`  result = []`
	`  for n in nums:`
	`  result.append(math.sqrt(n))`
	`  return result`
		~ 40 seconds

1. Eliminated attribute access
 	
2. Made the sqrt function local
	`from math import sqrt`
	
	`def square_root(nums):`
	`  result = []`
	`  results = result.append`
	`  sq_root = sqrt`
	`  for n in nums:`
	`  results(sq_root(n))`
	`  return result`
		~ 28 seconds

Ended with… (converted for loop to list comprehension)	`from math import sqrt`
	`def square_root(nums):`
	`  sq_root = sqrt`
	`  result = [sq_root(n) for n in nums]`
	`  return result`
		~ 25 seconds

More...

- Use comprehensions

```
result = [sq_root(n) for n in nums]
values = {key:value for key, value in prices.items() if value > 200}
```

- Examine your inner loops
- Check out and use the itertools module
- Use Sets and Dictionaries over lists where possible (looking up values is faster because of hashing)

Python for ArcGIS

- Work with layers and table views
- Use the “data access” module (arcpy.da)
- Avoid arcpy.Exists() and arcpy.Describe() when possible
- Avoid using Join Field tool - use Add Join, followed by CopyFeatures

Profiling

- time module -- (good for timing individual functions)
- cProfile module - (good for the entire program; create a detailed report)
- timeit module -- (good for small code snippets)

```
import time

def time_it(func):
    """A decorator to time a function."""
    def timed(*args, **kwargs):
      start = time.time()
      result = func(*args, **kwargs)
      end = time.time()
      print("func: {} took: {:.2f} sec".format(func.__name__, end - start))
      return result
  return timed

@time_it
def some_function():
     ...
```

Excellent resource & reference

Python Cookbook by David Beazley & Brian K. Jones

- 2nd Edition -- updated for Python 2.4
- 3rd Edition -- updated for Python 3.3