

## 4.1 Object oriented programming and ArcObjects?

### OO Programming

- Work with and manipulate objects
- Objects are features with properties and methods
- A property is a characteristic/attribute
- A method represents a behavior of an object
- General syntax for accessing objects and properties of an object
  - Object.Property
  - Object.Method
- An method may contain additional information called parameters or arguments
- Object oriented language tries to model real world entities which are objects themselves.
- What makes object oriented programming interesting and rich is that objects have relationships to one another and effects on one another.
- In the real world, a tree has branches and branches have apples on them. These relationships are mirrored by computer objects.
- For example, you may be able to write an instruction to create a tree object (Tree.Create) but not to create a branch object. Branch.Create doesn't work because branches don't exist independently of trees and this is defined by the properties and methods of your objects. Instead, to make a branch, you first create a tree and then run its GrowBranch method. If you want to make an apple, it might be Tree.Create, followed by Tree.GrowBranch, followed by Branch.GrowApple. Suppose you want to elaborate and make a boy eat an apple object. You would create the Apple object and then a boy object and then write an instruction like Boy.Eat(AnApple). When the boy eats the apple, consequences ensue. For example, the apple will no longer exist and the boy's IsHungry property might change from Yes to No.
- Important to learn the relationships among the objects you work with, which ones you can make, which ones make others, which ones have properties and methods, and which ones are affected by the properties and methods of others.

### ArcObjects

- A set of computer objects
- Specifically designed for programming with ArcGIS Desktop applications.
- Includes data frames, layers, features, tables, cartographic symbols and the pieces that make up things: points, lines, polygons, records, fields, colors, and so on.
- As an ArcGIS user you have been using ArcObjects all along.
  - Click the save button in ArcMap and a procedure runs to save your document,...

## 4.2 Geoprocessing tools

- Geoprocessing tools which can be accessed by Arcpy are dependent on the ArcGIS Desktop license that you have purchased.
- Every geoprocessing tool has a fixed set of parameters that provide the tool with the information it needs for execution.
- Tools usually have input parameters that define the dataset or datasets that are typically used to generate new output data. Parameters have several important properties:

Name	Each tool parameter has a unique name.
Type	The type of data expected, such as feature class, integer, string, and raster.
Direction	The parameter defines either input or output values.
Required	Either a value must be provided for a parameter or it is optional.

- When a tool is used in a script, its parameter values must be correctly set so it can execute when the script is run.
- The documentation of each tool clearly defines its parameters and properties.
- Once a valid set of parameter values is provided, the tool is ready to be executed.
- Parameters are specified as either strings or objects. Strings are simply text that uniquely identifies a parameter value, such as a path to a dataset or a keyword.
- Most tool parameters can be specified as a simple string. For some parameters, such as a spatial reference, you can use an object instead of a string. In the following code example, input and output parameters are defined for the Buffer tool. Note that the tool name is always appended with its toolbox alias. In the example, two string variables are used to define the input and output parameters to make the call to the tool easier to read.

```
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")
```

- In the following code example, the CreateFeatureClass tool is executed using a spatial reference object for its optional Coordinate System parameter.

- The spatial reference object is created using the SpatialReference class, and its information is loaded from a projection file.

```
import arcpy

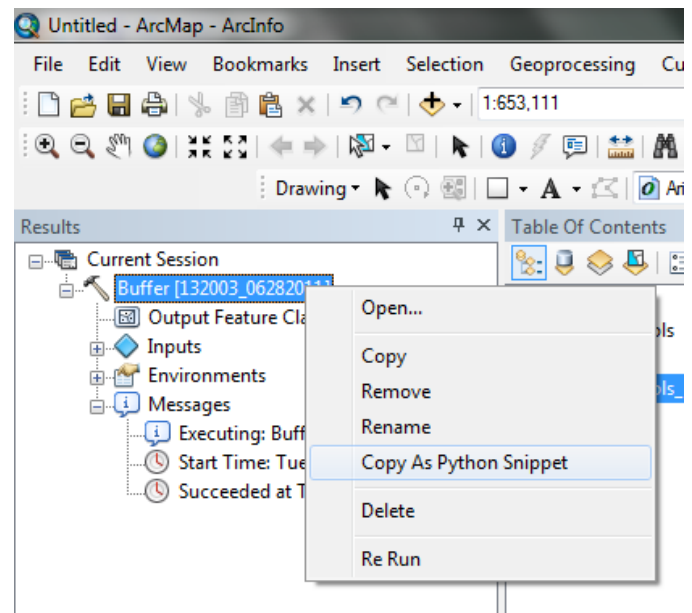
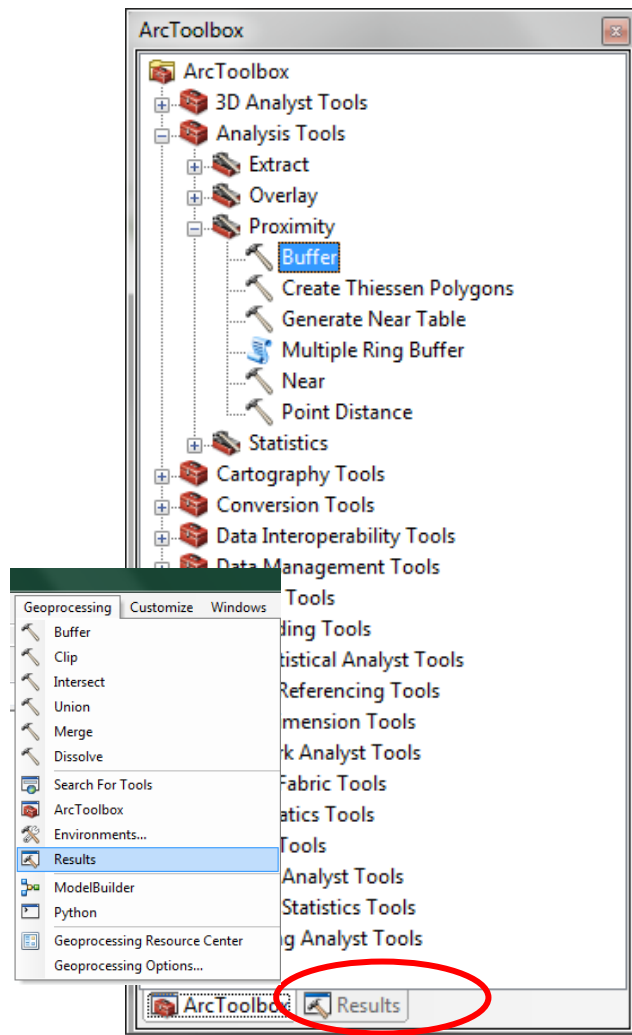
inputWorkspace = "c:/temp"
outputName = "rivers.shp"

# Create a spatial reference object
#
spatialRef = arcpy.SpatialReference()

# Use a projection file to define the spatial reference's properties
#
spatialRef.createFromFile("c:/program files/arcgis/Desktop10.0/Coordinate Systems/" + \
                        "Projected Coordinate Systems/Continental/North America/North America Equidistant Conic.prj")

# Run CreateFeatureclass using the spatial reference object
#
arcpy.CreateFeatureclass_management(inputWorkspace, outputName,
                                   "POLYLINE", "", "", "", spatialRef)
```

- Tools return a result object.
- The advantage of a Result object is that you can maintain information about the execution of tools, including messages, parameters, and output. These results can be maintained even after several other tools have been run.
- An easy way to create Python code that executes a particular tool is to use the Results window, as follows:
  - Use the tool dialog box to execute the tool.
  - After the tool executes, open the Results window.
  - Right-click the result and click Copy As Python Snippet.
  - In your Python code, paste the copied snippet.



- Tools produce messages, accessed through a variety of functions such as `GetMessages()`.
- All tools have tool reference pages

Section	Contents
Summary	A short summary of what the tool does. This summary may contain a link to another topic containing an in-depth discussion of how the tool works.
Illustration	Optional illustration that helps explain what the tool does.
Usage	Notes and clarifications about using the tool. When the notes refer to a particular parameter, the parameter label (what appears on the tool dialog box) is used rather than the parameter name (what appears in the syntax). The correspondence between the parameter label and name is usually obvious, but in some cases, the parameter name will be noted to avoid confusion.
Syntax	The exact syntax for executing the tool in a script. It is followed by a table listing all the parameters with their names, explanations, and the data (value) types. See <a href="#">Understanding tool syntax</a> for complete details.
Code sample	Contains one or more Python code samples using the tool.
Environments	This is a list of the <a href="#">environment settings</a> that the tool honors. Each environment is linked to the environment settings reference page. If the environment is of particular importance to the tool, there may be further information about the environment.
Related topics	A list of related topics, which always includes the toolset overview page.
Licensing information	Licenses required in order to use the tool.

- Results syntax:  
<http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v000000n7000000.htm>

### 4.3 Arcpy functions

- Accomplish a specific task as part of a larger program
- Functions provide better modularity for your application and a high degree of code reusing.
- Python gives you many built-in functions like print() etc. but you can also create your own functions. These functions are called *user-defined functions*.
- All geoprocessing tools are functions e.g. buffer, clips, ...
- Not all functions are geoprocessing tools
  - Listing datasets
  - Creating cursors
  - Describing datasets
- [http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/Alphabetical list of ArcPy functions/000v000000115000000/](http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#/Alphabetical%20list%20of%20ArcPy%20functions/000v000000115000000/)

### Tools vs. Functions

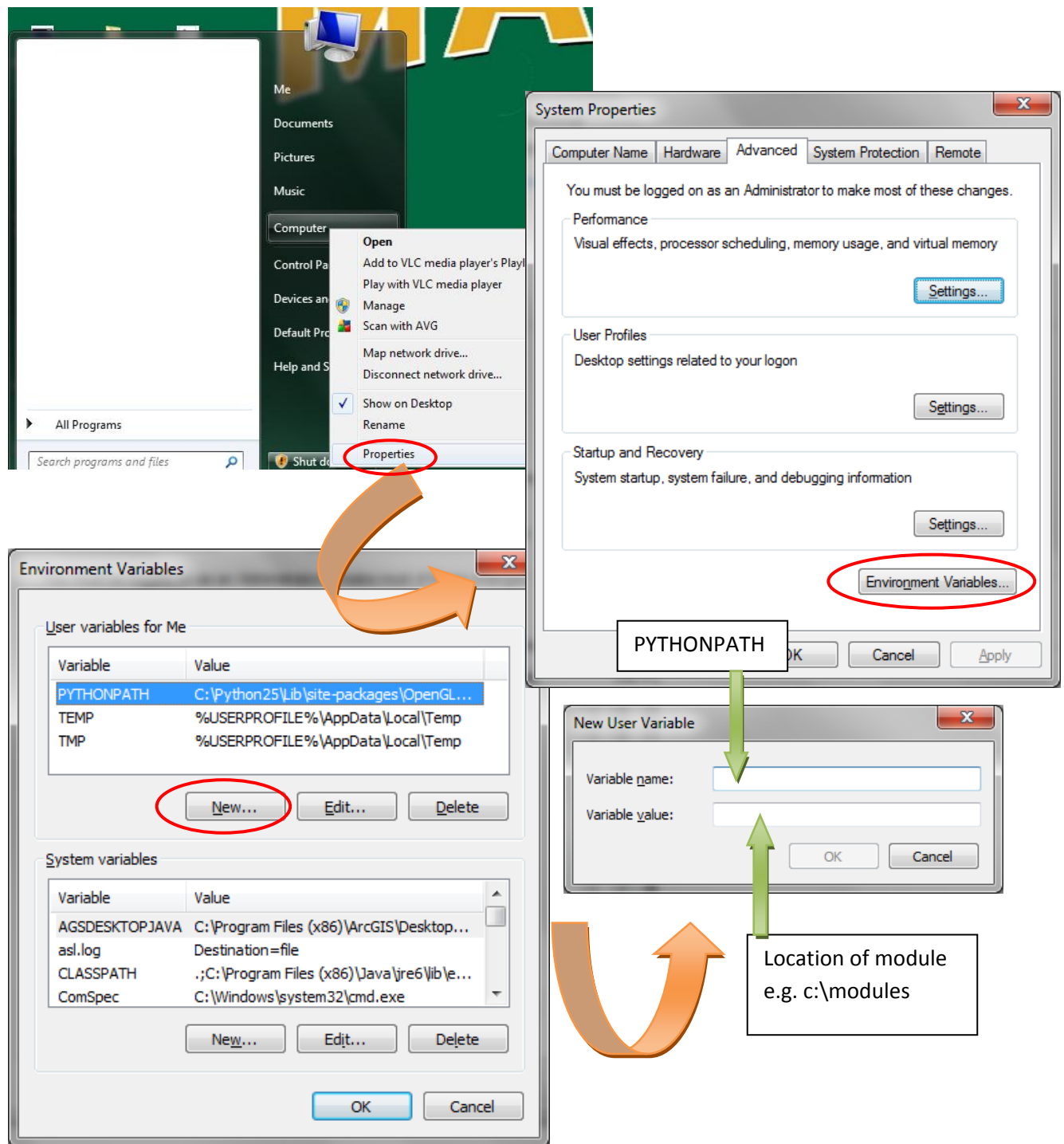
- Technically speaking, geoprocessing tools are functions available from Arcpy, that is, they are accessed like any other Python function. However, to avoid confusion, a distinction is always made between tool and non tool functions (such as utility functions like ListFeatureClasses()).
- Tools are documented differently than functions. Every tool has its own [tool reference page](#) in the ArcGIS Desktop help system. Functions are documented in the ArcPy documentation.
- Tools return a result object; functions do not.
- Tools produce messages, accessed through a variety of functions such as GetMessages(). Functions do not produce messages.
- Tools are licensed by product level (ArcView, ArcEditor, or ArcInfo) and by extension (Network Analyst, Spatial Analyst, and so on).
- You can find what license levels are required on the tool's reference page. Functions are not licensed—they are installed with ArcPy.

### 4.4 Arcpy classes

- A **class** is simply a *user-defined* block of code that lets you keep a bunch of closely related things "together"
- Creates the framework for creating objects
- Provide generic functionality e.g. spatial reference, point, polygon,...

## 4.5 Arcpy modules

- If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a *script*. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.
- To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module*; definitions from a module can be *imported* into other modules or into the *main* module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).
- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix `.py` appended.
- Within a module, the module's name (as a string) is available as the value of the global variable `__name__`.
- Specific purpose libraries containing function and classes
- Before a module can be used it must first be imported. This requires that a path to the module be set up.
  - Right click on "my computer" and select "properties".
  - Click Advanced tab on the top.
  - Click Environment Variables on the bottom.
  - Click new in either window and
  - Add the "PYTHONPATH" variable.
  - Use the variable value area to browse to the folder location where the module is located.
- When a module is imported, the interpreter searches for a file named `<module name>.py` in the current directory, and then in the list of directories specified by the environment variable [PYTHONPATH](#).



- Use `dir()` to get a list of all system modules

```
>>> import fibo, sys
>>> dir(fibo)
['_name_', 'fib', 'fib2']
>>> dir(sys)
['_displayhook_', '__doc__', '__excepthook__', '__name__', '__stderr__',
 '__stdin__', '__stdout__', '_getframe', 'api_version', 'argv',
 'builtin_module_names', 'byteorder', 'callstats', 'copyright',
 'displayhook', 'exc_clear', 'exc_info', 'exc_type', 'excepthook',
 'exec_prefix', 'executable', 'exit', 'getdefaultencoding', 'getdlopenflags',
 'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint', 'maxunicode',
 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache',
 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags',
 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',
 'version', 'version_info', 'warnoptions']
```

Without arguments, `dir()` lists the names you have defined currently:

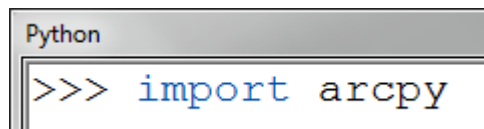
```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo
>>> fib = fibo.fib
>>> dir()
['_builtins_', '__doc__', '__file__', '__name__', 'a', 'fib', 'fibo', 'sys']
```

Task:

- Try creating a simple script and import it into the Python interpreter window
- In the created script, assign the value of 5 to the variable a
- Try accessing this variable using the interpreter window

## 4.6 Importing Arcpy modules

- ArcPy is supported by a series of modules, including a mapping module (arcpy.mapping), a Spatial Analyst module (arcpy.sa), and a Geostatistical Analyst module (arcpy.ga).
- To use Arcpy modules (mapping, sa and ga) Arcpy must first be imported

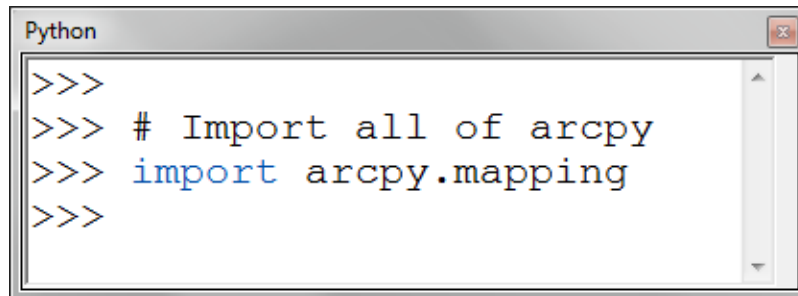


```
Python
>>> import arcpy
```

- Once you have imported ArcPy, you can run all geoprocessing tools found in the standard toolboxes installed with ArcGIS:
  - Analysis toolbox
  - Cartography toolbox
  - Conversion toolbox
  - Data Management toolbox
  - Editing toolbox

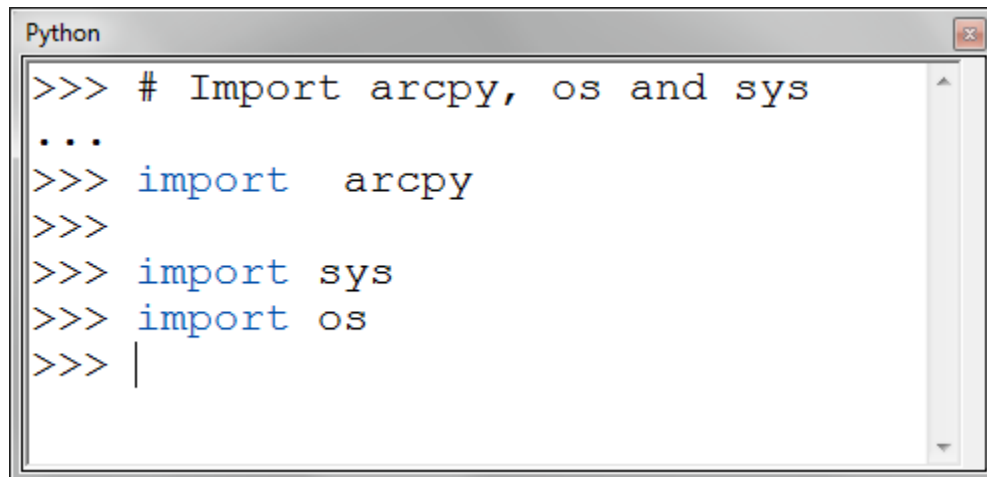


- Geocoding toolbox
  - Linear Referencing toolbox
  - Multidimension toolbox
  - Spatial Statistics toolbox
- To import an entire module, use the import module:



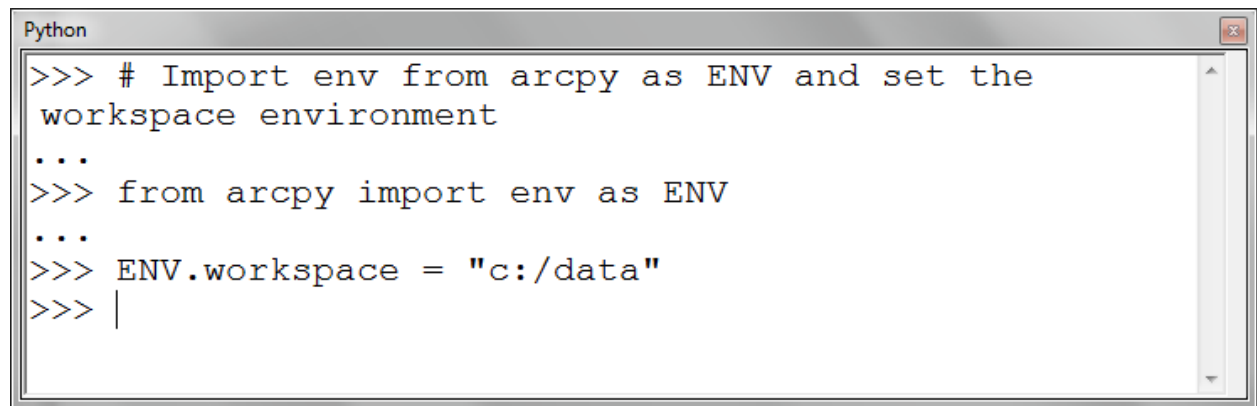
```
Python
>>>
>>> # Import all of arcpy
>>> import arcpy.mapping
>>>
```

- Of course, Python has many other core and third-party modules. If you wanted to also work with Python's core os and sys modules, you might use a similar import:



```
Python
>>> # Import arcpy, os and sys
...
>>> import arcpy
>>>
>>> import sys
>>> import os
>>> |
```

- In many cases, you might not plan or need to use the entire module. One way to import only a portion of a module is to use a from-import statement. The below example imports the env class (the env class contains all the geoprocessing environments). Now instead of having to access environments as arcpy.env, you can simplify it as env.

A screenshot of a Python command prompt window. The window has a title bar that says "Python" and a close button in the top right corner. The command prompt shows the following code being entered:

```
>>> # Import env from arcpy as ENV and set the  
workspace environment  
...  
>>> from arcpy import env as ENV  
...  
>>> ENV.workspace = "c:/data"  
>>> |
```

- Arcpy is built on Python version 2.6 so you need this version to access the full range of module functionality.

## 4.7 Quick vocabulary review

Term	Definition
<b>ArcPy</b>	ArcGIS 10 introduces ArcPy (often referred to as the ArcPy site-package), which provides Python access for all geoprocessing tools, including extensions, as well as a wide variety of useful functions and classes for working with and interrogating GIS data. A site-package is Python's term for a library that adds additional functions to Python. Using Python and ArcPy, you can develop an infinite number of useful programs that operate on geographic data.
<b>ArcPy modules</b>	A module is a python file that generally includes functions and classes. ArcPy is supported by a series of modules, including a mapping module ( <code>arcpy.mapping</code> ), a Spatial Analyst module ( <code>arcpy.sa</code> ), and a Geostatistical Analyst module ( <code>arcpy.ga</code> ).
<b>ArcPy classes</b>	A class is analogous to an architectural blueprint. The blueprint provides the framework for how to create something. Classes can be used to create objects, often referred to as an instance. ArcPy classes, such as the <code>SpatialReference</code> and <code>Extent</code> classes, are often used as shortcuts to complete geoprocessing tool parameters that would otherwise have a more complicated string equivalent.
<b>ArcPy functions</b>	<p>A function is a defined bit of functionality that does a specific task and can be incorporated into a larger program.</p> <p>In ArcPy all geoprocessing tools are provided as functions, but not all functions are geoprocessing tools. In addition to tools, ArcPy provides a number of functions to better support geoprocessing Python workflows. Functions or methods can be used to list certain datasets, retrieve a dataset's properties, validate a table name before adding it to a geodatabase, or perform many other useful scripting tasks.</p>

## 4.8 Exercise 3

- Data for this project is located in `\Data\Exercise3.zip` & `\Data\Exercise3.gdb`