

Exercise 4: Setting up the environment in Python

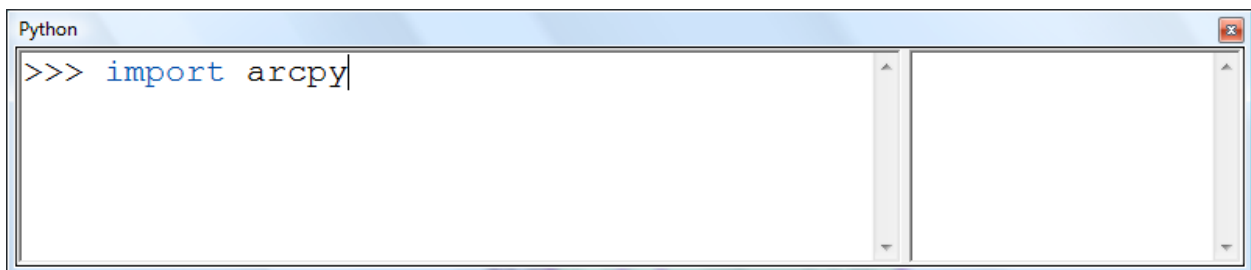
Geoprocessing environment settings are additional parameters that affect a tool's results. They differ from normal tool parameters in that they don't appear on a tool's dialog box (with certain exceptions). Rather, they are values you set once using a separate dialog box and are interrogated and used by tools when they are run.

Changing the environment settings is often a prerequisite to performing geoprocessing tasks. For example, you may already be familiar with the Current and Scratch workspace environment settings, which allow you to set workspaces for inputs and outputs. Another example is the Extent environment setting, which allows your analysis to be limited to a specific geographic area, or the Output Coordinate System environment setting, which defines the coordinate system (map projection) for new data. Only certain environments will be used by an individual tool, but tools use the environments in the same way. For example, all tools that use the output extent environment use it in the same way—they only process features in the output extent. To determine the environments that a tool will use, consult the tool's reference page.

In ArcPy, geoprocessing environments are organized as properties under the ArcPy class env. In the example below, several environment values are printed to the display, then set to new values. Following this, values will be restored once more to its default state since ArcGIS keeps tracks of default values.

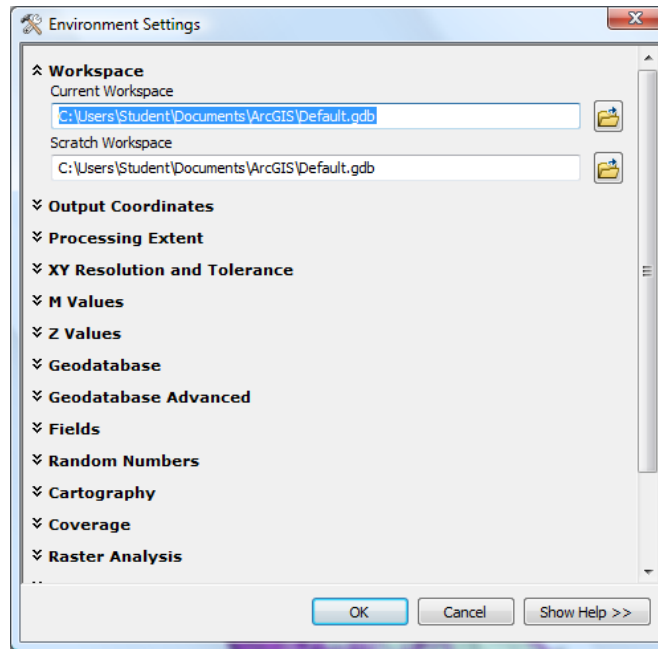
Step 1: Import Arcpy

- From the Python window type in the following: `import arcpy`

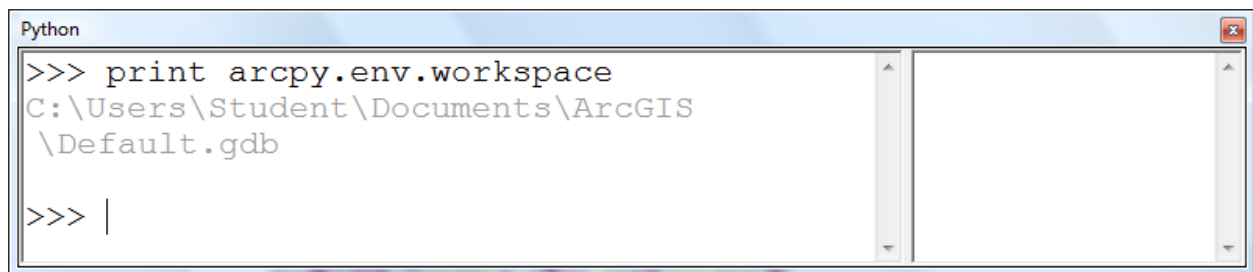


Step 2: Check the default settings

- Open ArcGIS and on the menu toolbar go to Geoprocessing → Environments... → Workspace. The default workspace should be displayed



- Open the Python window in ArcGIS and type the following command: `print arcpy.env.workspace` followed by the enter key. The information for the Current workspace should now be displayed



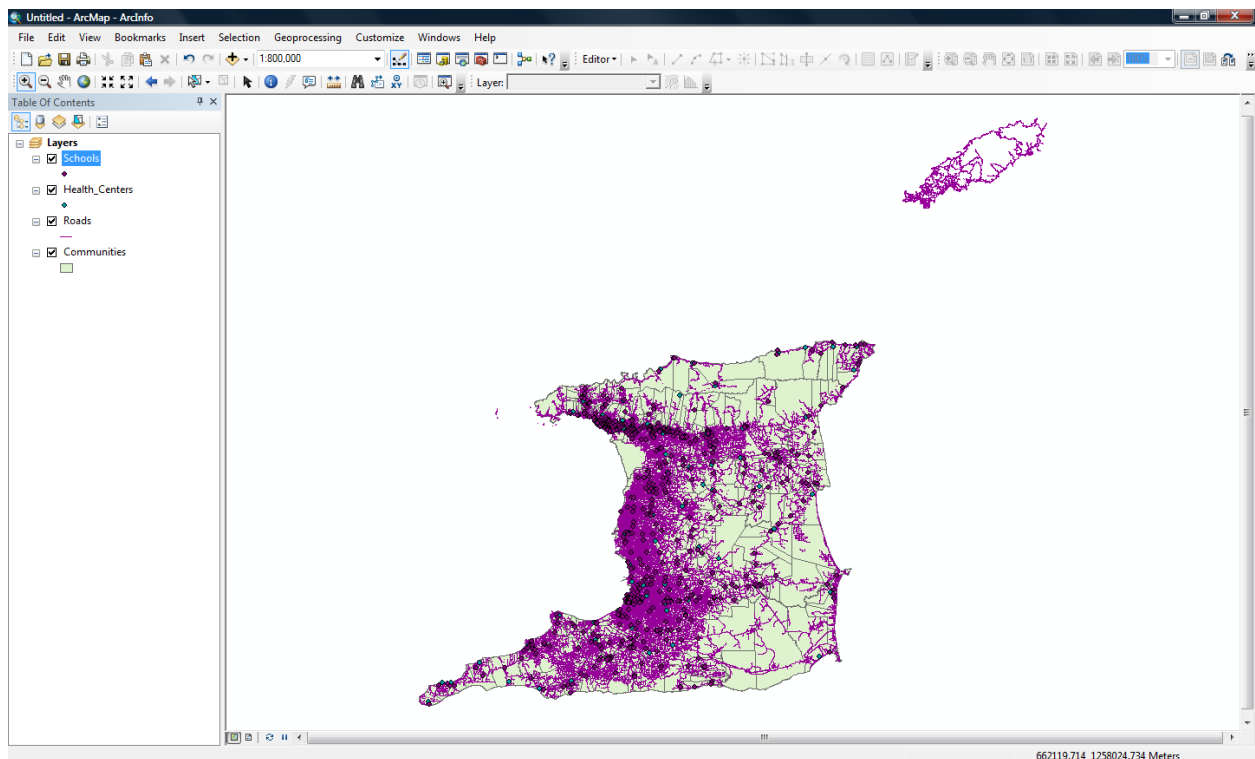
- You will have notices that after typing the period after 'arcpy.env.' several other options are shown from the autocomplete dialog box. They have meaning only to the type of data being processed. For example, because the data being used are shapefiles using the option `arcpy.env.cellSize` has little or no meaning at all. Try a few others and experiment.

`print arcpy.env.overwriteOutput` - prints true if this geoprocessing option is active, false otherwise

```
Python
>>> if arcpy.env.overwriteOutput == True:
...     print "The overwrite option is active"
...
The overwrite option is active
>>> |
```

Step 3: Setting up the environment

- Connect to and add all data from the Exercise4.mdb geodatabase



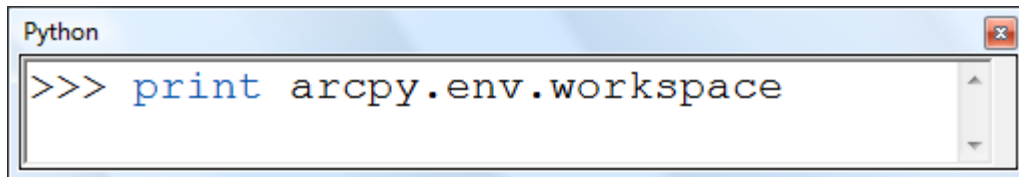
- In the Python window set the workspace path using the following

```
Python
>>> arcpy.env.workspace = "C:\Users\Student\Desktop\GIS
Programming\Training\Data\Exercise4.mdb"
>>> |
```

In the example above, the Exercise4.mdb is stored in "C:\Users\Student\Desktop\GIS Programming\Training\Data". Replace this with the path where your data is stored on the your computer.

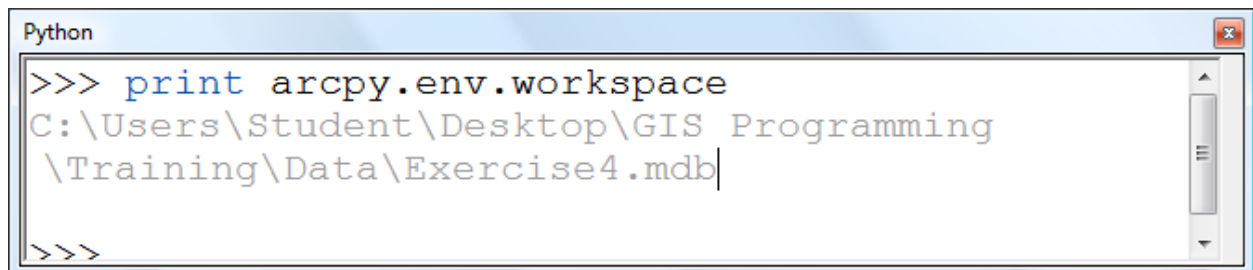
Step 4: Check the workspace

- In the Python window type the following and hit enter



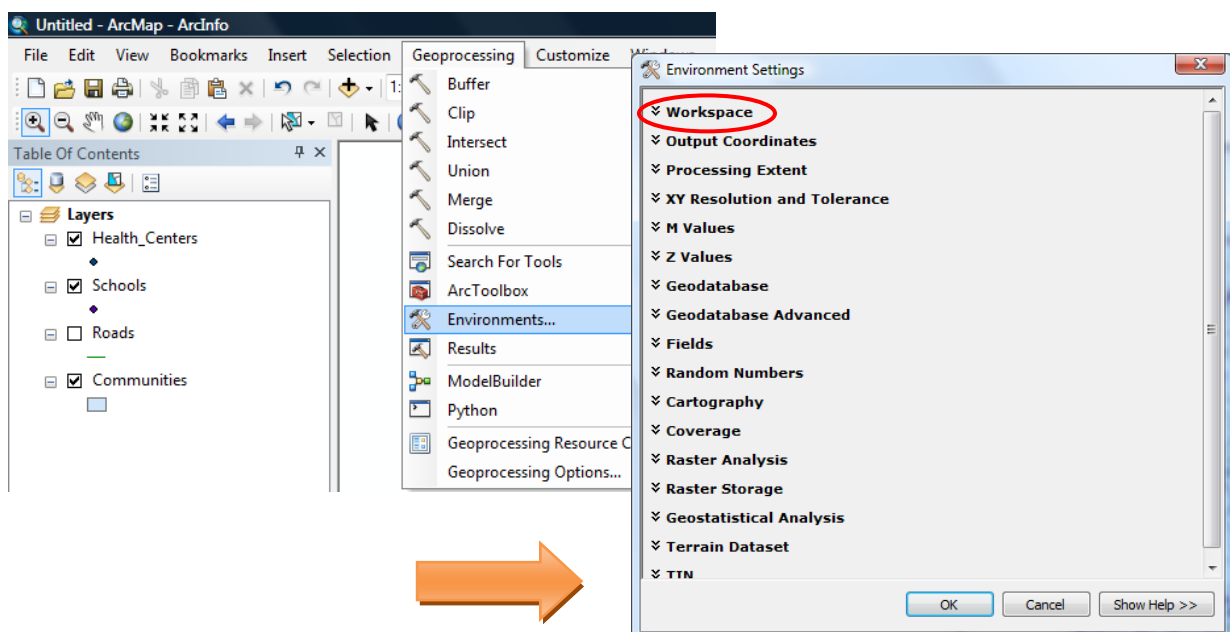
```
>>> print arcpy.env.workspace
```

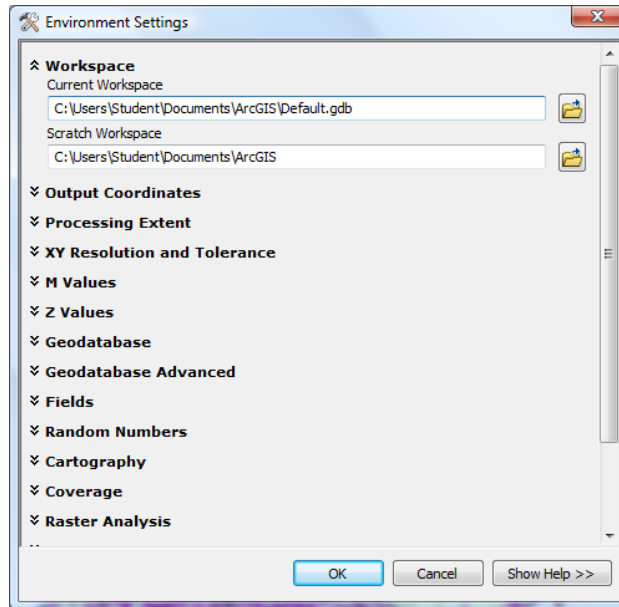
The workspace path the Exercise4.mdb should be displayed.



```
>>> print arcpy.env.workspace
C:\Users\Student\Desktop\GIS Programming
\Training\Data\Exercise4.mdb
>>>
```

- Check the Environment settings for the workspace. Notice that the settings at the application level remained the same. Remember, when using specific environmental settings in Python they are only used once for the execution of the necessary tools and therefore do not change the global settings at the application level.





Step 5: Buffer schools

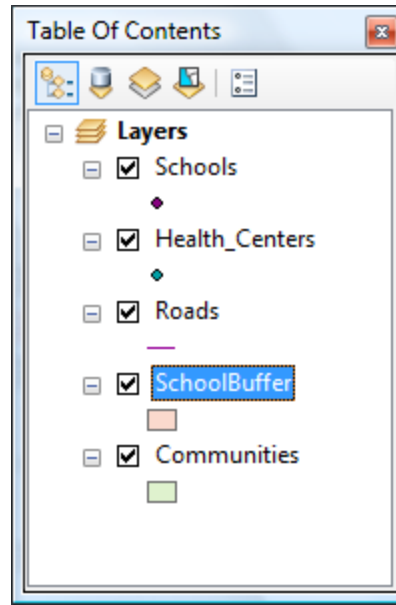
- Type the following into the Python window and hit the ENTER key.

```
>>> arcpy.Buffer_analysis("Schools","SchoolBuffer","100  
METERS","FULL","ROUND","","")
```



```
>>> arcpy.Buffer_analysis("Schools","SchoolBuffer","100  
METERS","FULL","ROUND","","")  
<Result 'C:\\Users\\Student\\Desktop\\GIS Programming\\Training\\Data  
\\Exercise4.mdb\\SchoolBuffer'>  
>>>
```

A new shapefile SchoolBuffer.shp has been added to the Table of Content in ArcMap



Step 6: Check the Exercise4.mdb geodatabase

- A new polygon shapefile with the name SchoolBuffer.shp has been added.

Step 7: List all Feature Classes

- Type the following and type the ENTER Key

```
Python
>>> arcpy.ListFeatureClasses()
```

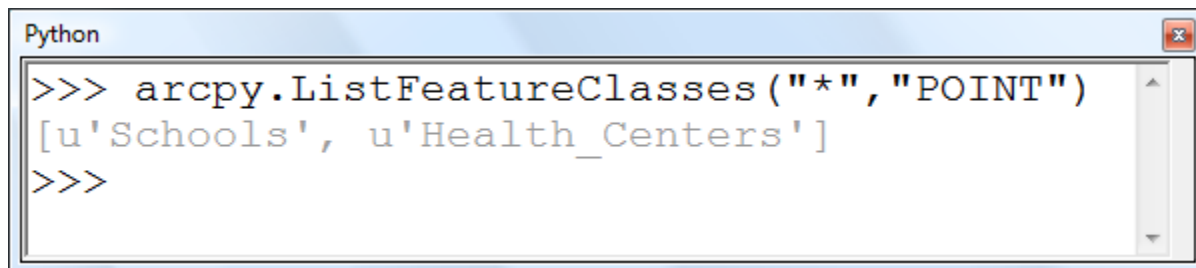
```
Python
>>> arcpy.ListFeatureClasses()
[u'Schools', u'Roads',
u'Health_Centers', u'Communities',
u'SchoolBuffer']
>>> |
```

The purpose of ListFeatureClasses() is to provide the user with a listing of feature classes in a workspace. The command is simple and accepts two parameters, a wildcard, and the type of feature class one is seeking. Below is a listing of the parameters:

wildcard	OPTIONAL. A combination of the star character (*) and any number of characters to help limit results. For example, listing all feature classes ending with the "fc" characters in their name would use a wildcard "*fc".
Feature Class Type	OPTIONAL. Limits the type of feature classes that are returned. Possible values are: <ul style="list-style-type: none"> • ANNOTATION: Only annotation feature classes are returned. • ARC: Only arc feature classes are returned. • DIMENSION: Only dimension feature classes are returned. • LABEL: Only label feature classes are returned. • LINE: Only line feature classes are returned. • NODE: Only node feature classes are returned. • POINT: Only point feature classes are returned. • POLYGON: Only polygon feature classes are returned. • REGION: Only region feature classes are returned. • ROUTE: Only route feature classes are returned. • TIC: Only tic feature classes are returned.

In order to execute the command, you also need to have your workspace defined.

- Retrieve all point feature classes from the Workspace



```

Python
>>> arcpy.ListFeatureClasses("*", "POINT")
[u'Schools', u'Health_Centers']
>>>

```

- Repeat the above step to repeat a list of all polygon and polyline datasets. NOTE: The Feature Class Type parameter is NOT case sensitive.
- List all workspaces

The purpose of ListWorkspaces() is to provide the user with a listing of the types of workspaces that ArcGIS can make use of. The command is simple and accepts two parameters, a wildcard, and the type of workspace one is using. Below is a listing of the parameters:

wildcard	OPTIONAL. A combination of the star character (*) and any number of characters to help limit results. For example, listing all workspaces with the word nature anywhere in the name would use a wildcard "*nature*".
Workspace	OPTIONAL. Limits the type of workspaces that are returned. Possible values are:

Type	<ul style="list-style-type: none"> • Access: Only personal geodatabases are returned. • Coverage: Only coverage workspaces are returned. • FileGDB: Only file geodatabases are returned. • Folder: Only shapefile workspaces are returned. • SDE: Only SDE databases are returned. • ALL: Returns all above types of workspaces.
-------------	--

In order to execute the command, you also need to have your workspace defined. This may sounds a bit contradictory, as you are trying to find workspaces, but what this does is identify an initial location on the disk to start from. So, if your previous workspace was a personal or file geodatabase for example, you would need to reset the workspace to one level above in order to have the geodatabase be listed using the ListWorkspaces command.

- To reset the workspace, use Step 3 above.
- Try the following commands in the Python Window
 - `arcpy.ListWorkspaces("*", "Access")` # Return all personal geodatabases type workspaces
 - `arcpy.ListWorkspaces("*", "Coverage")` # Return all coverages type workspaces
 - `arcpy.ListWorkspaces("*", "FileGDB")` # Return all file geodatabases type workspaces
 - `arcpy.ListWorkspaces("Exercise3*", "Folder")` # Return all folders starting with the word "Exercise3"
 - `arcpy.ListWorkspaces("*", "SDE")` # Return all SDE type workspaces
 - `arcpy.ListWorkspaces("*", "ALL")` # Return all workspaces of all types
- List Fields

The purpose of ListFields() is to provide the user with a listing of the fields within a table, shapefile or feature class that ArcGIS can make use of. The command is simple and accepts three parameters, an input table/shapefile/feature class, a wildcard, and the type of field one is trying to list. Below is a listing of the parameters:

Input Table	REQUIRED: The input file to be used to read fields. Possible types of files accepted are: <ul style="list-style-type: none"> • Table (dBase, INFO) • Shapefile (.shp) • Feature Class (any of the feature classes shown above in ListFeatureClasses)
wildcard	OPTIONAL. A combination of the star character (*) and any number of characters to help limit results. For example, listing all fields that begin with the word "Shape" would use a wildcard "Shape*".
Field	OPTIONAL. Limits the type of workspaces that are returned. Possible values are:

Type

- SmallInteger: Only fields of Small Integer type are returned.
- Integer: Only fields of Integer type are returned.
- Single: Only fields of Single precision floating point are returned.
- Double: Only fields of Double precision floating point are returned.
- String: Only fields of String type are returned.
- Date: Only fields of Date type are returned.
- OID: Only fields of OID type are returned (internal Object IDentifier number).
- Geometry: Only fields of Geometry type are returned.
- BLOB: Only fields of Binary Large Object are returned.
- ALL: Returns all above types of fields.

In order to execute the command, you also need to have your workspace defined. This may sound a bit contradictory, as you are trying to find workspaces, but what this does is identify an initial location on the disk to start from.

- Reset the workspace to the access database containing the shapefiles.
 - To reset the workspace, use Step 3 above.
- List all attributes of the "Schools" shape file. Type the following in the Python Window

```
fieldList = arcpy.ListFields("Schools") ## Set the Schools feature class
```

```
for field in fieldList:      ##iterates throughout all attributes in the schools feature class
```

```
    print field.name      ##Prints the name of each attribute in the schools feature class
```