Geoprocessing using Python

- Using the ArcPy site package:
  - Site package in Python is like a library of functions that add functionality to Python
  - ArcPy is organized in modules, functions, tools and classes.

```
import arcpy
import arcpy.mapping
```

  - Once you import ArcPy or one of its specialized modules, you can start using its modules, functions and classes
  - When using classes, the syntax is:

```
arcpy.<class>.<property>
arcpy.env.workspace="d:/arcgis"
```

- Using the ArcPy site package:
  - from-import statement to import only a portion of a module

    from arcpy import env

    env.workspace= “d:/arcgis”
  - from-import-as giving a module or part of the module a custom name

    from arcpy import env as myenv

    myenv= “d:/arcgis”

- Using tools:
  - When working with geoprocessing tools, the tools are referred by name, not the tool label; the name of the tool contains no spaces.
  - A reference to a particular tool also requires the toolbox alias; toolbox alias is not the same as either the name or the label of the toolbox-it is typically an abbreviated version.

- Using tools:
    - Two ways to access a tool:
        1. arcpy.<toolname_toolboxalias>(<parameters>)
        2. arcpy.<toolboxalias>.<toolname>(<parameters>)
    - Geoprocessing tool syntax for the parameters:
        - Required and optional parameters, separated by comma
        - Optional parameters are surrounded by curly brackets{ }
        - Required parameters come first, followed by optional parameters
        - Input dataset are usually the first parameter, followed by the output dataset, next are additional required parameters, and finally optional parameters

- Using tools:
    - Specify some optional parameters and skip others:
        - Setting the optional parameters using an empty string (“ ”) or the number sign (“#”)
        - Specifying the name of the parameters that needs to be set; bypassing all the others

- Using tools:

```
Buffer_analysis (in_features, out_feature_class, buffer_distance_or_field,
{line_side}, {line_end_type}, {dissolve_option}, {dissolve_field})
```

```
arcpy.Buffer_analysis(“roads”, “buffer”, “100 METERS”, “ ”, “ ”, “LIST”,
“Code”)
```

```
arcpy.Buffer_analysis(“roads”, “buffer”, “100 METERS”, “ ”, “ ”, “LIST”,
“Code”)
```

```
arcpy.Buffer_analysis(“roads”, “buffer”, “100 METERS”,
dissolve_optiona=“LIST”, dissolve_field=“Code”)
```

- Using variables for parameters

```
import arcpy
arcpy.env.workspace=“C:/data”
infc=“roads.shp”
outfc=“resutls.shp”
buffer_distance=“100 METERS”



arcpy.Buffer_analysis(infc, outfc, “100 METERS”, “”, “”, “LIST”,
“Code”)
```

- Result object:
  - Output of a tool could be a new or updated feature class, a string, a number, or a Boolean value; when the output of a tool is a feature class, the result object includes the path to the dataset.
  - Result object has properties and methods
  - The result object can be used as an input to another tool or function

- Working with toolboxes:
  - Once the ArcPy site package is imported into Python, all the system toolboxes are available.
  - Even if a custom toolbox has been added to Arctoolbox in ArcMap or ArcCatalog, Python is not aware of this toolbox until it has been imported.
  - ImportToolbox function with optional alias parameter

```
import arcpy
arcpy.ImportToolbox (“c:\data\sampletoolbox.tbx”)
arcpy.ImportToolbox(“c:\data\sampletoolbox.tbx”, mytools)
```

- Using functions:
  - All geoprocessing tools are provided as functions in ArcPy: tool function; But ArcPy provides a number of functions that are not geoprocessing tools: nontool function;
  - Function syntax: arcpy.<functioname>(<arguments>)

```
import arcpy
print arcpy.Exists(“c:/data/stream.shp”)
```

- Using classes:
  - ArcPy classes can be used to create objects; then the objects can be used as parameters for tools.
  - Syntax for setting the property of a class: <classname>.property=<value>

```
import arcpy
arcpy.env.workspace="c:/data"
```

- Using classes:
  - SpatialReference class:
    - Syntax for using a method to initialize a new instance of a class: arcpy.<classname>.(parameters)

```
import arcpy
prjfile=“c\data\myprojection.prj”
spatialref=arcpy.SpatialReference(prjfile)
myref=spatialref.name
print myref
```

- Using environment settings:
  - Environment settings are exposed as properties of the env class; these properties can be used to retrieve the current values or to set them; each property has a name and a label, Python works with name only

```
import arcpy
print arcpy.ListEnvironments ()
```

  - overwriteOutput property: default is False

```
import arcpy
from arcpy import env
env.overwriteOutput=True
```

- Working with tool messages:
  - When a tool is run from Python window of ArcGIS, only error messages that indicate a particular situation prevented the tool from running appear.
  - Running of a stand-alone Python script, messages are not added to Results window.
  - All messages have a severity property: 0 (information), 1 (warning), 2 (error).

- Working with tool messages:
    - Message from the last tool run are maintained by ArcPy and can be retrieved by the GetMessages function

      ```
      print arcpy.GetMessages()
      ```

    - GetMessage function can retrieve individual message with one parameter: the index position of the message.

      ```
      print arcpy.GeMessage(0)
      ```

    - Retrieve the last message:

      ```
      count=arcpy.GetMessageCount()
      print arcpy.GetMessage(count-1)
      ```

- Working with tool messages:
  - Query the maximum severity of the messages using the GetMaxSeverity function:

    ```
    print arcpy.GetMaxSeverity()
    ```

  - To get messages from any tool run, you can use result object.

    ```
    import arcpy
    arcpy.env.workspace=“c:/data”
    result=arcpy.GetCount_management (“stream.shp”)
    count=result.messageCount
    print result.getMessage(count-1)
    ```