Manipulating spatial data

- Using cursors to access data:
  - A cursor is a database technology term for accessing a set of records in ta table; A cursor can be used to iterate over the set of records in a table or to insert new records into a table; in ArcGIS, cursors can be used to read and write geometries to and from records, row by row.
  - Three types of cursors in ArcGIS:
    - Search cursor: used to retrieve rows
    - Insert cursor: used to insert rows
    - Update cursor: used to update and delete rows

- Using cursors to access data:
  - Each type of cursor is created by the corresponding arcpy.da function SearchCursor, InsertCursor and UpdateCursor
  - All three cursors can work on a table, a table view, a feature class, or a feature layer; All three functions create a cursor object that can be used to access row object.
  -

Using cursors to access data:

Cursor	Explanation
`arcpy.da.InsertCursor(in_table, field_names)`	Inserts rows
`arcpy.da.SearchCursor(in_table, field_names, {where_clause}, {spatial_reference}, {explode_to_points}, {sql_clause})`	Read-only access
`arcpy.da.UpdateCursor(in_table, field_names, {where_clause}, {spatial_reference}, {explode_to_points}, {sql_clause})`	Updates or deletes rows

Using cursors to access data:

Methods supported by cursor type

Cursor	Method	Description
Search	next	Retrieves the next row
	reset	Resets the cursor to its starting position
Insert	insertRow	Inserts a row into the table
	next	Retrieves the next row object
Update	deleteRow	Removed the row from the table
	next	Retrieves the next row object
	reset	Resets the cursor to its starting postion
	updateRow	Update the current row

- Using cursors to access data:

```
import arcpy
fc="c:/data/study.gdb/roads"
Cursor=arcpy.da.SearchCursor(fc,["STREETNAME"])
for row in cursor:
        print "Streetname={0}".format(row[0])

Screen output
Streetname=National
……
```

- Using cursors to access data:

```
import arcpy
fc="c:/data/study.gdb/roads"
cursor=arcpy.da.InsertCursor(fc,["STREETNAME"])
cursor.insertRow(["NEW STREET"])
……
```

- Using cursors to access data:

```
import arcpy
fc="c:/data/study.gdb/zone"
cursor=arcpy.da.UpdateCursor(fc,["ACRES", "SHAPE_AREA"])
for row in cursor:
    row[0]=row[1]/43650
    cursor.updateRow(row)
……
```

- Using cursors to access data:

```
import arcpy
fc="c:/data/study.gdb/roads"
cursor=arcpy.da.UpdateCursor(fc,["STREETNAME"])
for row in cursor:
    if row[0]=="MAIN ST"
     cursor.deleteRow()
del row
del cursor
……
```

- Using cursors to access data:
  - Insert and update cursors apply an exclusive lock to a table or feature class, preventing other processes from making changes; insert and update cursors cannot be applied if an exclusive lock already exists.
  - Once an exclusive lock is created by a script, the lock persists until the script release the lock; using del statement to delete the cursor object creating the lock
  - To avoid using del statements, you can use with statement

- Using cursors to access data:

```
import arcpy
fc="c:/data/study.gdb/roads"
With arcpy.da.UpdateCursor(fc,["STREETNAME"]) as cursor:
  for row in cursor:
     if row[0]=="MAIN ST"
     cursor.deleteRow()
……
```

- Using SQL in Python:
  - SQL(structured query language) queries can be carried in Python using the SearchCursor:

    SearchCursor(in_table, firld_names, {where_clasue}, {spatial_reference}, {explode_to_points})

    import arcpy

    fc="c:/data/study.gdb/roads"

    cursor=arcpy.da.SearchCursor(fc, ["NAME", "CLASSCODE"], ' "CLASSCODE"=1')

- Using SQL in Python:
  - field delimiters for shapefile and file geodatabase consists of double quotation marks “ ” , personal geodatabase feature classes use square brackets [ ] , and ArcSDE geodatabase feature classes use no delimiters. To avoid confusion and to ensure the field delimiters are the correct ones, you can use the AddFieldDelimiters function:

```
import arcpy
fc=“c:/data/study.gdb/roads”
delimfield=arcpy.AddFieldDelimiters(fc, “CLASSCODE”)
cursor=arcpy.da.SearchCursor(fc, [“NAME”, “CLASSCODE”], delimfield+ “=1”)
```

- Working with table and field names:
  - ValidateTableName(name, {workspace}): determine whether a specific table name is valid for a specific workspace; invalid characters are replaces by underscore _
  - ValidateFieldName(name, {workspace}): takes a field name and a workspace and returns a valid field name for the workspace; invalid characters are replaces by underscore _
  - Use ListFields function to create a list of the fields a table or feature class, and the new field can be compared against this list
  -

- Working with table and field names:
  - CreateUniqueName function creates a unique name in the specified workspace by appending a number to the input name. This number is increased until the name is unique.

```
import arcpy
from arcpy import env
env.workspace="c:/data"
unique_name=arcpy.CreateUniqueName("buffer.shp")
arcpy.Buffer_analysis ("roads.shp", unique_name, "100 FEET")
```

  -

- Parsing table and field names:
  - It is essential when working with goedatabase that a geoprocessing tool needs to know the table name,database name, and owner name, etc.
  - ParseTableName function returns a single string with database name, owner name, and table name, each separated by a comma.
  - import arcpy
  - from arcpy import env
  - env.workspace="c:/data/study.gdb"
  - fc="roads"
  - fullname=arcpy.ParseTableName(fc)
  - Namelist=fullname.split(",")
  - databasename=namelist[0]
  - ownername=namelist[1]
  - fcname=namelist[2]
  -

- Working with text files
  - open(name, {mode}, {buffering}) opens a file and return a file object
  - Mode:

    r---read mode (default mode if no mode specifies);

    w---write mode;+:----read/write mode (added to another mode);

    b-----binary mode(added to another mode-for examples, rb,wb, and others)

    a------append mode

```
f=open (“c:/data/mytext.txt, “w”)
f.write(“GIS”)
f.read()
f.close()
```