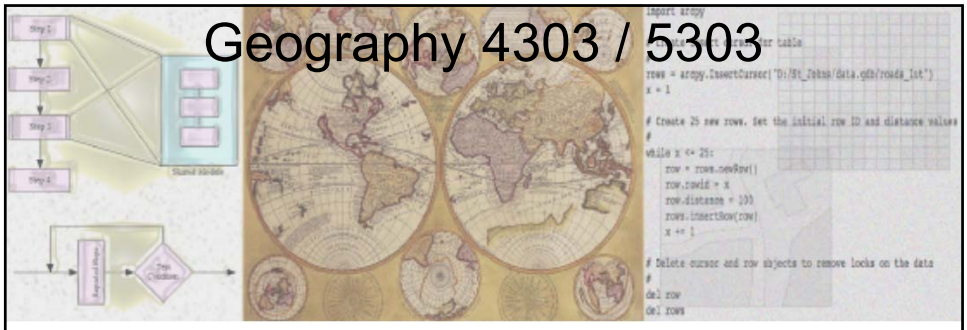


Geography 4303 / 5303

GIS Programming for Spatial Analysis

Class 06: Debugging, Geometry and Spatial Sampling for Infrormation Extraction

Some Updates

- Reminder: **Final Projects**
- **Next week: Proposal Presentations**
- Send us your three top picks
- Groups of 2-4 students
- Skills (don't forget!): Programming, Project management, Presentations, Communication and Writing

Last Lecture / Last Week

- You have seen already most of methodological **GP** capabilities you can use for Py Programming
- Also we had a look into more advanced programming techniques including **modules** and **functions**
- Altogether this enables you to think much more **conceptual** and **structured** (say: **systematic**) for solving specific problems in a **GIScience** context
- Solution exercise class 05!

Today ‘s Outline

- We will talk about **sampling strategies** and implementing sampling designs in space
- We will discuss **random** and **systematic** designs
- How to use the Geometry object for flexible spatial adjustments and modifications for information extraction
- You will discuss some fundamental techniques helpful for different aspects of **statistical analysis**
- We will look at the **debugging** environment

Learning Objectives

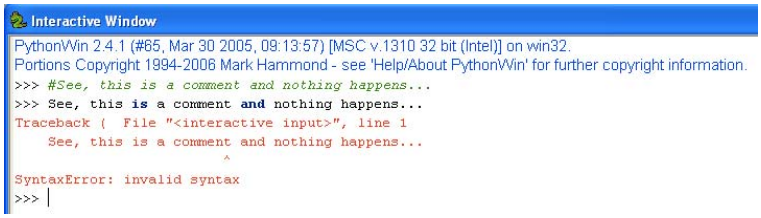
- You will learn how to use your programming skills for **sampling** in space
- You will learn why the **Geometry** object is at the core of spatial sampling and data extraction
- You will learn how to implement different **sampling designs** that can be used for extraction in RS data
- …how to sample **randomly** or **systematically** in relation to existing features (large data context)
- You will learn how to **debug** your code

Remember

- => Tool functions (ArcToolbox)
- => Cataloguing, organizing and listing spatial data: Batching
- => Describing spatial data and their properties
- => Creating, editing and manipulating spatial data (data access)
- => More complex tasks combining above (e.g., Sampling in space w/ geometry)

About Errors…

- **Syntax** errors (writing errors)
- **Runtime** errors (illegal operations)
- **Semantic** errors (working properly but wrong output)



And how to approach them…

- **Compile** errors: Syntax check befor execution:
  - Syntax check for pure programming syntax and indentation

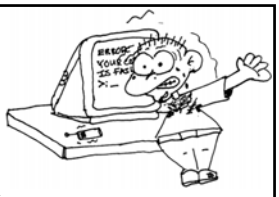
```
Python and the TabNanny successfully checked the file 'testGDALNumerics.py'
```

- **Run-time** (to be caught during execution)
  - `Print` statements at critical points
  - `Try/Except` statements to raise exception if something's wrong
  - `gp.getmessages()`

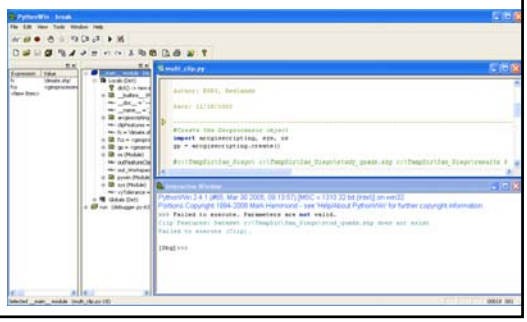
- **Logic errors** (think about it… and … Debugg!):
  - Run the program in Debugging mode

```
>>> Failed to execute. Parameters are not valid.
Clip Features: Dataset c:\TempDir\San_Diego\stud_quads.shp does not exist
Failed to execute (Clip).
```

Debugging Environment



- While running: Interrogate variables, check object validity, evaluate expressions
- **Stack view window**: Check scope of **variables** and contents of **modules** imported
- **Watch window**: Display values of certain variables
- Step through each line of the program
- **Breakpoints**: Execute the program until this point



Class01_03_guessGame, then class05_02_debugg

Sampling and Selection in Space for Information Extraction

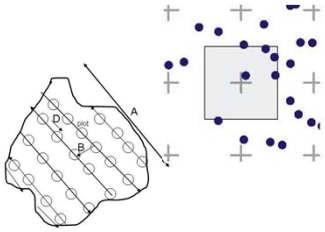
- **Representative** subset of an underlying population
- Collect data in the field or in remote sensing images or combined – data are BIG!
- Spatial sampling for **field surveys** or as **data input** to modeling (training data)
- Strategies/Design depend on

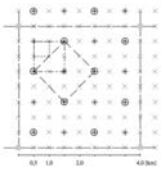
  -Objective of the survey (error, costs, variance)

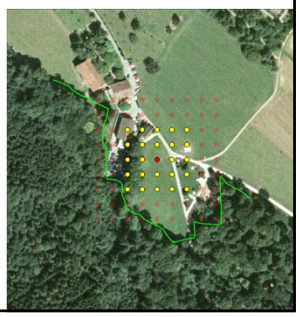
  -Geographical extent to be surveyed (scale)

Some Sampling Approaches

- **Random Sampling** as the most representative approach
- **Systematic sampling** is an approach to sample if random sampling is difficult to be realized (field conditions, efficiency…)
- **Stratified sampling**
- **Area frames**
- **Multi-stage sampling** (RS/field surveys)





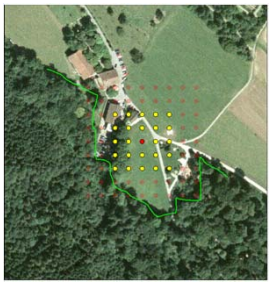


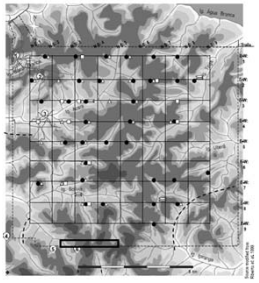
How to Spatially Sample to Extract Information

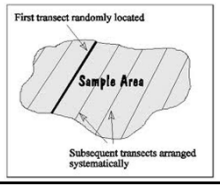
- Define **spatial extent** within which you want to sample: (sub)area, population, along/across features (lines, boundaries)
- Define a **sampling design** (or strategy) HOW you want to collect information in the field or from imagery (**random**, **systematic**, **distances** between locations, **shapes**)
- Determine **locations** for sampling
- Extract the information at the considered **points** (point-based) or in **local environments** (e.g., concentric circles)

Some Examples

- **Predictive Habitat Modeling** using Remote Sensing data sources
- Forest or landscape **inventories** based on field surveys and Remote Sensing
- **Land use change** analysis based on RS time series or statistical assessments
- **Landscape diversity assessment** (edges along transects)
- **Machine Learning**







What you need to know

- You need to be able to **describe area and sampling locations schematically**
- Know how to use feature **properties** (extent, lengths, shape types, first/last points ect.)
- You have to be able to create **new objects** in space (points, areas, lines)
- Implement a given sampling design using features
- You need to know how to **derive (extract) information** and make this information available (attribute table or spatial layer)

Random … ?

- First rule to make a **random sampling** is to **generate** random numbers!
- Python allows you to use the **random module** to do this (plus methods provided):

Floats [0,1]	**Ints** [0,100]
`random()`	`randint(0,100)`

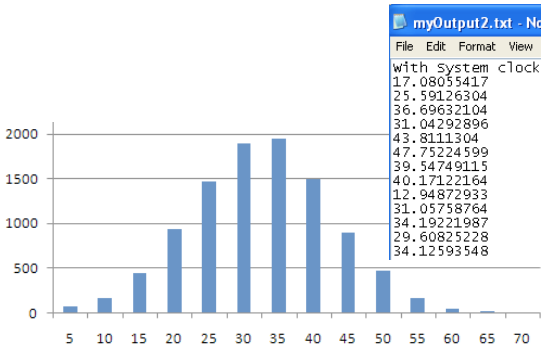
- Distributions:

  `normal(…)`

  `poisson(…)`

Defining Distributions

- You can use **predefined distributions** for creating random numbers
- For samples that should follow certain criteria to represent a **population of expected characteristics**
- **Simulation tasks**
- Random!=Random



Geometry and Sampling

- Sampling based on **spatial information** means: Sampling using geometry
- Properties of the **geometry object** are useful to determine the “extent” or locations at which sampling has to be done
- … or the local sampling area

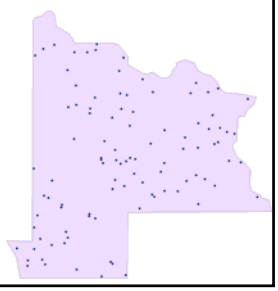
Geometry

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*



Flexibility through work with Geometry

Point

METHODS

clone(point_object)
contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- ID–*Integer*
- M–*Double*
- X–*Double*
- Y–*Double*
- Z–*Double*

Polygon

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Polyline

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolea*
disjoint(second_geometry)→*Boole*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boole*
touches(second_geometry)→*Boolea*
within(second_geometry)→*Boolean*

PROPERTIES

- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Geometry

METHODS

contains(second_geometry)→*Boolean*
crosses(second_geometry)→*Boolean*
disjoint(second_geometry)→*Boolean*
equals(second_geometry)→*Boolean*
getPart(index)→*Array*
overlaps(second_geometry)→*Boolean*
touches(second_geometry)→*Boolean*
within(second_geometry)→*Boolean*

PROPERTIES

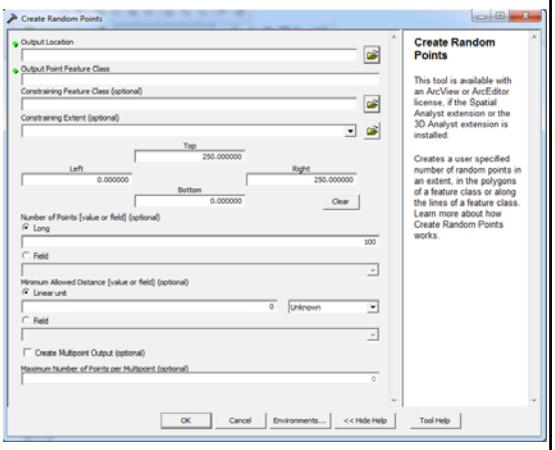
- area–*Double*
- centroid–*Point*
- extent–*Extent*
- firstPoint–*Point*
- hullRectangle–*String*
- isMultipart–*Boolean*
- labelPoint–*Point*
- lastPoint–*Point*
- length–*Double*
- partCount–*Integer*
- pointCount–*Integer*
- trueCentroid–*Point*
- type–*String*

Geometry and Sampling?

- Sampling **along lines** (every x meter, in equal distances, randomly)
- Sampling **within polygons** (system, rand)
- Sampling **within areas around points**
- Sampling at **predefined coordinates** based on random and systematic design

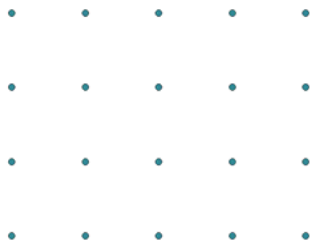
The Random Points Tool

- ArcToolbox provides a tool for **generating random points** along a line or within polygons
- These standard mechanisms can thus be used if sufficient
- If you need something else …



Systematic Sampling in 2D

- Extent of the area (Rectangle around your polygon!)
- Mesh sizes
- Number of points

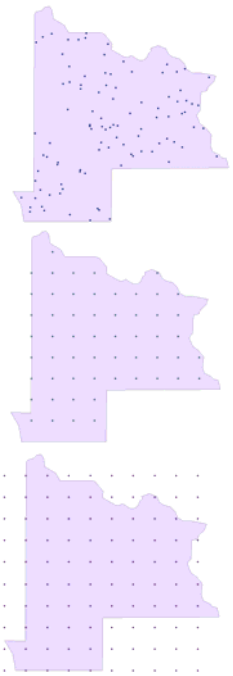


```
for i in range(0,10):
    g += 1
    if xurStr > xllStr: x[i*(10)] = xllStr + i*partTransectX
    if xurStr < xllStr: x[i*(10)] = xllStr - i*partTransectX
    if yurStr > yllStr: y[i*(10)] = yllStr
    if yurStr < yllStr: y[i*(10)] = yllStr
    id[i*(10)] = g
    for j in range(0,9):
        g += 1
        if xurStr > xllStr: x[j+(i*10)+1] = x[(i*10)]
        if xurStr < xllStr: x[j+(i*10)+1] = x[(i*10)]
        if yurStr > yllStr: y[j+(i*10)+1] = yllStr + (j+1)*partTransectY
        if yurStr < yllStr: y[j+(i*10)+1] = yllStr - (j+1)*partTransectY
        id[j+(i*10)+1] = g
```

Show syst sample fct

Where to go from here

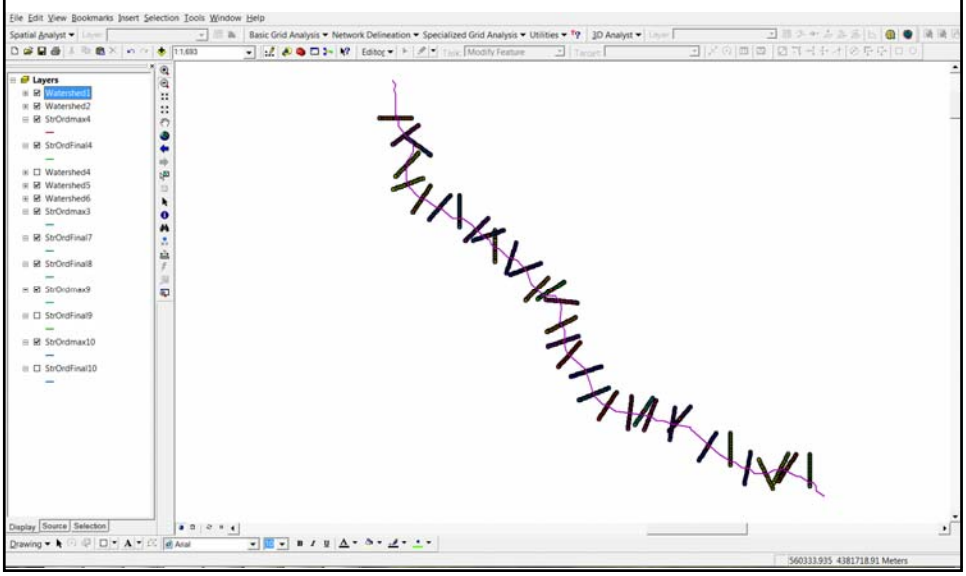
- Varying **density** in different subregions
- First–stage **subsample**
- **Varying** local sampling **areas** for extraction
- Refining sample design by **masking** (e.g., water/external data)
- Finally **shape adjustment** for non-circularity



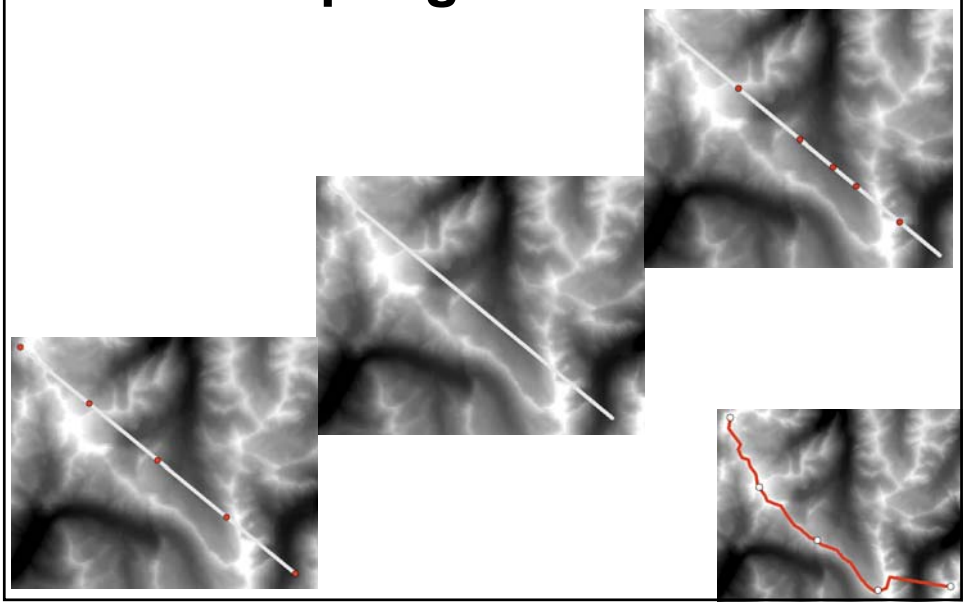
Show geom modify fct for sys sampleadjustment

Let's look at sampling strategies along a line segment

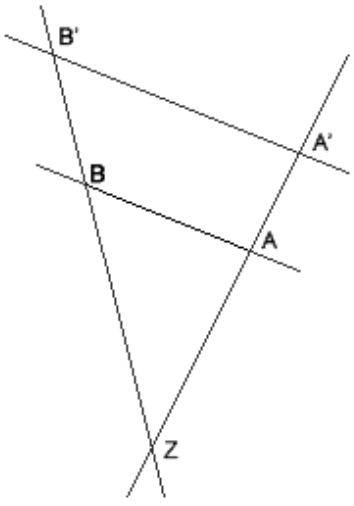
Cross Sections (final project by Francis Rengers)



Sampling a Transect



Theorem on Intersecting Lines (Intercept Theorem)



$$\overline{ZA'} : \overline{ZA} = \overline{ZB'} : \overline{ZB}$$
$$\overline{ZA} : \overline{AA'} = \overline{ZB} : \overline{BB'}$$
$$\overline{ZA'} : \overline{AA'} = \overline{ZB'} : \overline{BB'}$$

$$\overline{A'B'} : \overline{AB} = \overline{ZA'} : \overline{ZA}$$
$$\overline{A'B'} : \overline{AB} = \overline{ZB'} : \overline{ZB}$$

Summary

- Sampling in space as a fundamental approach for field **survey planning** or **RS based information extraction** in different disciplines (Biogeography, land use, Landscape Ecology, Urban geography,…)
- We can make use of the **geometry** (object) to define the strategy how we would like to sample in space
- **Sampling schemes** are flexible and can be adjusted to your needs (along lines, within certain areas, varying densities of sampling points, depending on autocorrelation, random/systematic, etc…) and include time!!!

Next time

- **Raster** data processing
- Working with 2D raster matrices
- Numpy
- Remote sensing context