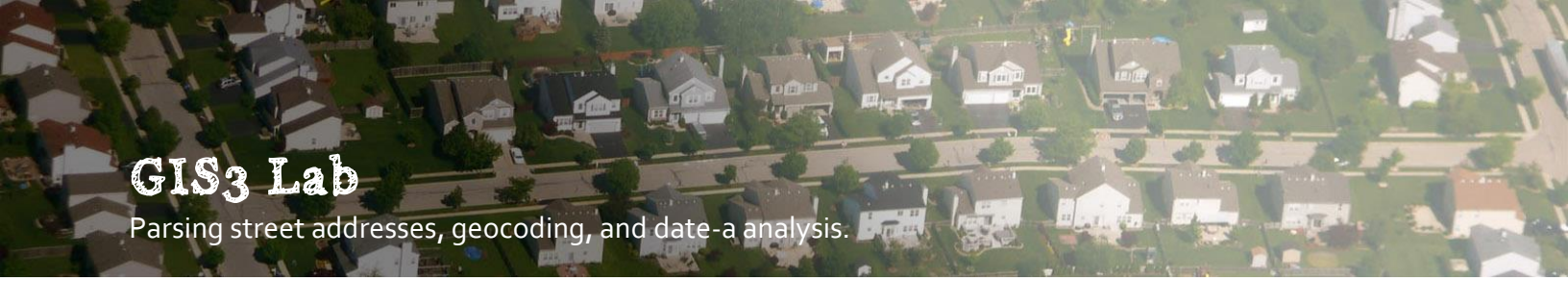


GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

Introduction and background

Street addresses and geocoding **(Bolstad, p395-397, 301-303)**

An important element of any GIS is knowing how your data are spatially referenced. In many applications, street addresses are the preferred way to spatially reference other data. Customer and client data, crime incident reports, and emergency 911 responses, are just a few examples of data that can be tied to street addresses. Although most users can query and summarize street names or zip code values, they will eventually need coordinates to plot their data and visualize them with other spatial data layers.

Every time you've ever searched for an address using Google Maps, Bing Maps, Apple Maps, OpenStreetMap, Yahoo Maps, etc., you've used some kind of geocoding. Geocoding is a workflow that creates point features (spatial data with coordinates and attributes) from street address data (text data without coordinates). Geocoding one address with your browser is easy. Geocoding in bulk, however, requires more.

To geocode in bulk you need an "address locator." An address locator is a database object that can be composed of building polygons (with address attributes and coordinates), road centerlines (with address attributes and coordinates), or town or city centroids (place names and coordinates). The more complete and accurate your address locator, the better chance you have of getting a good return when you search.

Simple address locators are composed of one layer of reference data. A composite address locator, however, is composed of multiple datasets and that are searched in a particular order - from your most complete and accurate reference layer to your least. If an address cannot be found in the first layer, then the search is passed to the second layer … until a matching address is found or not found.

Google Maps uses a composite address locator. During map search, the text of your inputted address is compared first against Google's building attributes. If Google finds a matching text address, then it grabs the building's coordinates and returns the location as a pin (red balloon symbol). Figure 1 shows "350 Maple Street" was successfully geocoded (address matched ✓, coordinates assigned ✓).

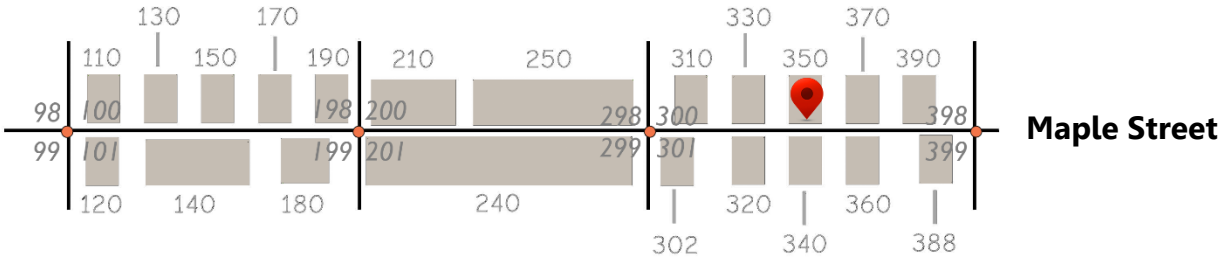


Figure 1. A geocoded address via building polygon search.

If Google doesn't find a matching address in its buildings database, then it will search its road database next. Google's road segments are attributed with address ranges (from this # to that #). Figure 2 shows the address "1113 County Road" can be found along the line segment attributed with name = "County Road" AND from = "994" AND to = "1129." As Bolstad (2016: p326) describes, the coordinates are estimated via linear interpolation. In this example, address number 1113 is an odd number and 88% of the way between 995 and 1129 (Example 1). So, the address is plotted at the spot that marks 88% of the line segment's `Shape_Length`. In this example, the address is plotted on the left side of the road segment because the odd range was attributed to the left side.

$$((1113_{search} - 995_{from}) / (1129_{to} - 995_{from})) * shape\_length = 0.88 * shape\_length \qquad \text{Ex. 1.}$$

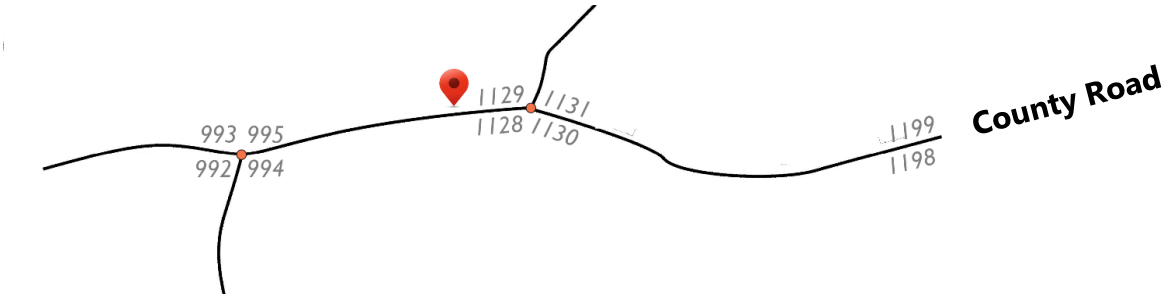
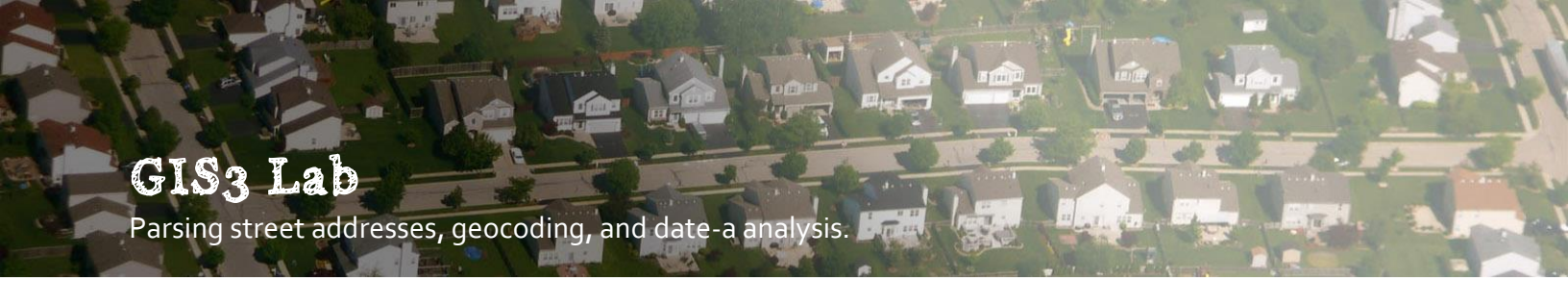


Figure 2. Geocoding via road segment search and linear interpolation.

The typical geocoding workflow has five steps :

1. **Build or obtain a high-quality reference dataset**: often a line feature class representing street centerlines is used (but it could also be a polygon feature class representing buildings or a point feature class representing post offices). The feature class must have a spatial reference system AND street address data in its attribute table. The address data must be conform to one of these standard addressing styles;
    a. **Some standard addressing styles**:
        i. US ADDRESS – DUAL RANGES: Street centerlines with an address range, place name, and ZIP code for each side of every line segment (e.g., a range of odd numbers on one side, a range of even numbers on the other side);
        ii. US ADDRESS – ONE RANGE: Street centerlines with an address range, place, and ZIP code data for both sides of each line segment (left vs. right is ignored);
        iii. US ADDRESS – SINGLE HOUSE: Polygons (or polygon centroids) that represent ownership parcels;
        iv. US ADDRESS – ZIP 5: Polygons (or polygon centroids) that represent 5-digit ZIP code areas;
        v. GENERAL: Polygons (or polygon centroids) that represent places or landmarks.
2. **Build an address locator**: An address locator is a database object that works with the high-quality reference dataset and facilitates both the address-matching and the coordinate assignment processes;
3. **Prepare a table of street address**: Your table of 'raw' addresses should be prepared using the same addressing style that's used by your reference dataset (see 1.a above).
4. **Locate addresses**
    a. **Address matching:** find matches between input table text and locator table text**.**
    b. **Assign coordinates from the reference dataset to the matched addresses** (and don't assign coordinates to unmatched addresses).
5. **Maintain your address locator:** An address locator contains a snapshot of address attributes and indexes provided from the reference data. Whenever changes are made to the reference data that reflect real changes on the ground, you will need to rebuild the address locator to reflect those changes.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

In the USA, street addresses can take many forms. In western states, for example, it is common for street names to reflect the underlying Public Land Survey System. So, you might encounter an address like:

"56 North Townline East Road".

In the Mid-Atlantic region, we might encounter an address like:

"1950 Fruitville Pike" or "5026 Old Sowell Mill Pike"

which don't reflect any kind of surveying system, but landmarks along former toll roads that radiated from central places.

GIS analysts that maintain national address databases need to be aware of every conceivable format that a street address can take. So, each element of an address has a name (see Table 1).

Table 1. Elements of street addresses used around the United States

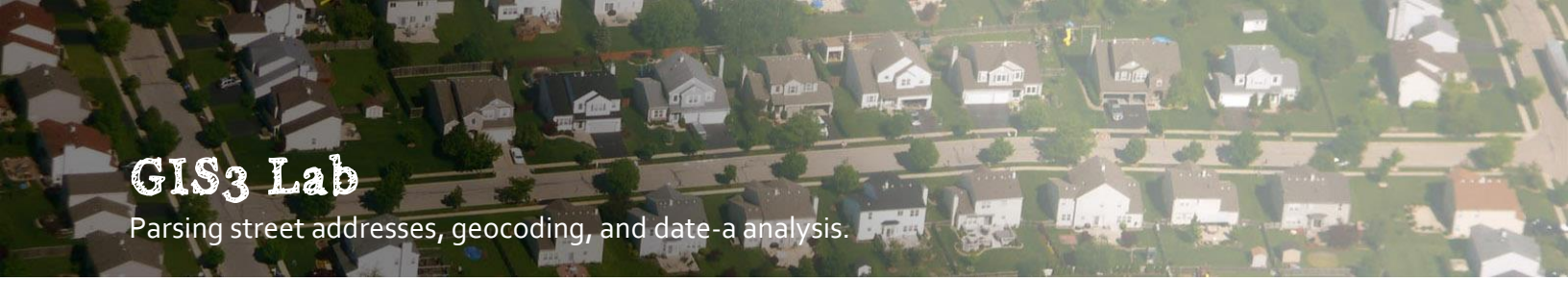
Address element name	Example
Address number	**44** Old North Canal Street
Street name pre modifier [The, Old, New, Alternate, …]	44 **Old** North Canal Street
Street name pre directional [N,NE, … W,NW]	44 Old **North** Canal Street
Street name	44 Old North **Canal** Street
Street type [Rd, St, Dr, Ave, Hwy, …]	44 Old North Canal **Street**
Street name post directional [N,NE, … W,NW]	95 Church Street **West**
Street name post modifier [Bypass, Loop, Spur, …]	95 Church Street **Bypass**
Complete street names	**44 Old North Canal Street** , **95 Church Street Bypass**

Our problem

The number and spatiotemporal[1] distribution of pizza restaurants in any town or city reflects the underlying distributions of likely customers and zoning regulations (if applicable). In other words, people that run pizza businesses want to be open *where* and *when* people are likely to buy pizza, and they will build those businesses as real-estate regulations allow.

The Borough of Carlisle, PA, is home to many restaurants that sell pizza. Some are open for lunch, but not all. Many deliver, but not all. In the aggregate, there's a surprising amount of variability in Carlisle's pizza market. But when the data are sliced temporally and geographically, the distribution(s) make perfect sense. The purpose of this lab is to analyze and describe Carlisle's spatiotemporal pizza market.

[1] spatiotemporal [adj.] Having both spatial and temporal qualities. Existing in both space and time.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

Objectives

To analyze and describe Carlisle's spatiotemporal pizza market, we need the following pieces of information:

1. We need to know where the pizza shops are located.
2. We need to know when each pizza shop opens and closes for business.
3. We need to know when each pizza shop begins and ends deliver service.

To accomplish our first objective, we're going to geocode a table of pizza shop addresses against Cumberland County's street centerline dataset (the high-quality reference dataset).

Table 1: Access and download these datasets from the course website.

Data originator	Data distributor	Dataset, by name
Dr. Drzyzga	COURSE WEBSITE	carlislePizzaShops.xlsx
Cumberland County GIS		cumberlandCountyGIS.zip

Methods

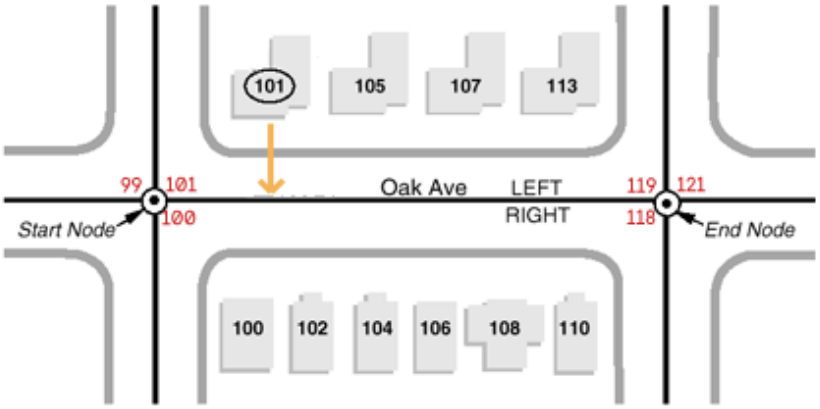
Step 1. Build or maintain a reference dataset

Start in ArcCatalog. **Create** a folder for this lab and, in it, **create** a new file geodatabase called "gis3pizza_<your initials here>.gdb". In the geodatabase, create a new feature dataset called "carlisle" and tie it to the metric (NAD83) State Plane PA-South coordinate system. Import any geospatial data you downloaded from the course website into your "carlisle" feature dataset.

Preview your street centerline data. The lines will look like typical road data, but these lines are attributed with street address elements. Each line segment carries attributes that indicate the street name, street type, the address range on each side of the segment (there's a Left side and a Right side) as well as ZIP code indicators for each side of the segment (there's a Left side and Right side for zip codes, too).

Preview the attribute table. Pay particular attention to the fields STREETNAME, STREETSUF, PREFIXDIR, POSTDIR, LFROM, LTO, RFROM, RTO, LZIP and RZIP. Use Figure 3 below as a guide.

NOTE: If working at home, then you might have to **Customize…** your **ArcCatalog Options** by changing the default **Metadata** style from *Item Description* to **FGDC Metadata**.



Record Type 1 contains separate data fields for both the start and end of each address range.

Record Type 1				Address Range			
				Left side		Right Side	
				Start	End	Start	End
RT	GIS_ID	STNAME	STSUFFIX	LFROM	LTO	RFROM	RTO
1	0007654320	Oak	Ave	**101**	**119**	**100**	**118**

Figure 3: A sketch of a centerline segment representing "Oak Avenue" that's been annotated with street name, start and end nodes, segment side indicators (left vs. right), and both address ranges (left range vs. right range). The implied direction (from the start nodes to the end nodes) does not indicate the direction that traffic flows or that Oak Avenue is a one-way street; rather, it indicates the direction that street numbers increase through both ranges. Also revealed in this sketch is the potential for coordinate displacement during linear interpolation. Notice how the locations of 101 Oak Ave in the buildings dataset and the centerlines dataset are not co-located. In this case, the address can be found in both datasets, but the location that is returned will depend on which dataset is searched first.

Steps 2 and 3. Select a standard address locator style and build an address locator object
Stay in ArcCatalog. Right-click your lab folder and **Create Address Locator….**

Setting up an Address Locator is pretty easy to do. Choose the **US Address – Dual Ranges** style – a nationally recognized standard. Use your projected centerline attribute table as your Primary Table (find the 'Role' parameter); and then map the fields in your centerline attribute table to the awaiting fields (*) in your address locator (see Table 2 below). Your field names do not need to be identical; they just need to carry the expected elements into the address locator. Output your address locator file to your lab folder (and not your geodatabase). **OK**.

Table 2. Mapping field names in the street centerline attribute table to the address locator.

Field Name (in the Address Locator)	Alias Name (in the centerline feature class)
Feature ID	ID
* From Left	LFROM
* To Left	LTO
* From Right	RFROM
* To Right	RTO
Prefix Direction	PREFIXDIR
Prefix Type	<None>
* Street Name	STREETNAME
Suffix Type	STREETSUF
Suffix Direction	POSTDIR
Left City	LCITY
Right City	RCITY
Left ZIP	LZIP
Right ZIP	RZIP

* Indicates a required input.

Step 4. Get familiar with your reference data and your Address Locator

Close ArcCatalog and **open a session of ArcMap**. Set your default geodatabase and add your projected centerline features as a new layer.

Next, **label** your street centerlines. Use this Python **expression** to build a complete label for each road feature:

```
[PREFIXDIR] + " " + [STREETNAME] + " " + [POSTDIR] + " " + [STREETSUF]
```

Next, **symbolize** all line segments that were attributed as Avenues with a light grey line with a light grey end arrow. Symbolize all others with a black line and a black end arrow. (Figure 4).

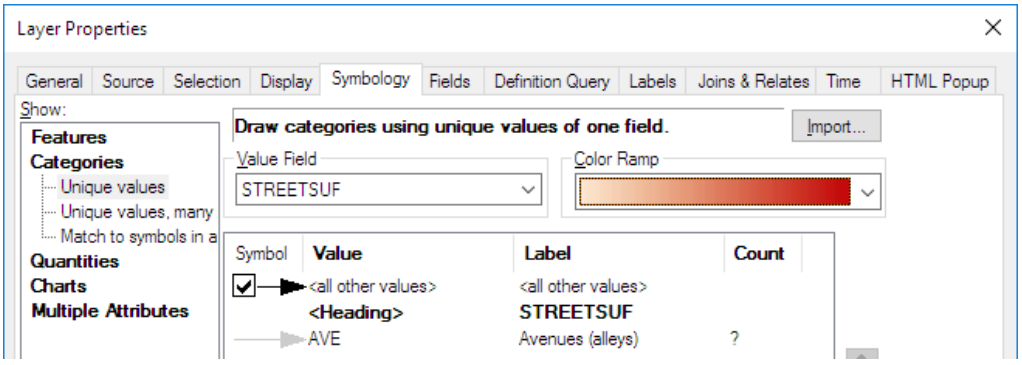
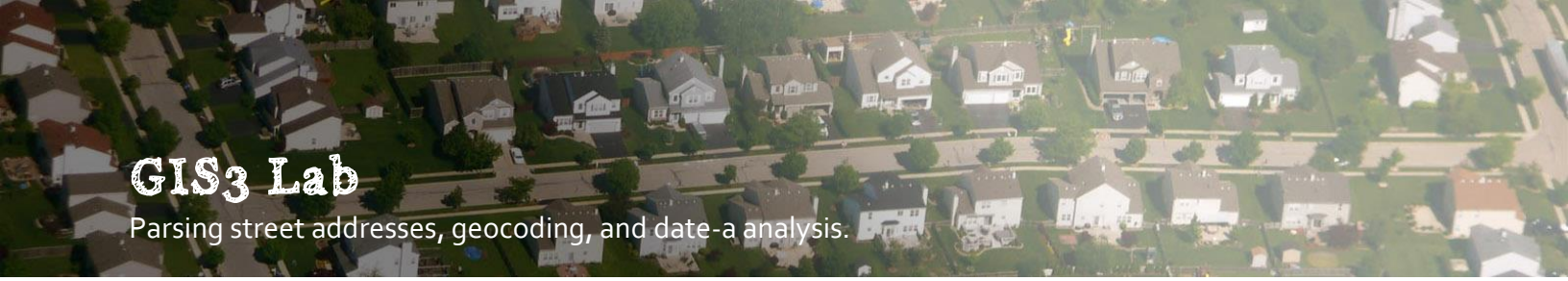


Figure 4: Symbolizing line segment directions. In Carlisle, the alleys are called 'avenues'.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

Next, turn on the **Geocoding toolbar**. Use the toolbar to **manage and add your address locator**; make your address locator the active Address Locator.

Use the toolbar to enter and find "53 West South Street, Carlisle, PA 17013", which is where Carlisle Borough Hall is located. If all went well, the location should flash on screen.

Next, build a two-part query to select all the segments of West Willow Street and the segments of the other streets that touch it.

First, use **Select by Attributes** to select segments where:

PREFIXDIR = 'W' AND STREETNAME = 'WILLOW' AND STREETSUF = 'ST'

The first part of your query should return 7 selected features.

Second, use **Select by Location** to select features from centerlines that are within a distance of 3 meters of the selected set of centerlines. **Create a layer from your 23 selected features** then turn off the original centerlines layer. Re-label and re-symbolize if the labels and symbols don't carry over. Use the **Identify** button/tool to explore the attributes of each line segment.

To help you get familiar with how the Address Locator works, find the **Address Inspector** button on the toolbar and, with your mouse pointer down, trace the line segments that comprise WEST WILLOW STREET. Notice how the address number increases or decreases as you trace up or down the address range. Be sure to try both sides of the street.

Question 1: Use pen and paper to sketch W WILLOW ST and all its adjoining street segments (23). Annotate your sketch with symbols and labels that indicate:

1. complete street names,
2. arrowheads that indicate the end node and direction of increasing address numbers
3. street sides (left vs right),
4. and the address ranges (see Figures 1, 2 and 3 for examples).
5. Also, sketch and label the spot where we should expect to find "369 W Willow Street".

Step 4. Prepare a table of raw street address to be geocoded.
Create a table view by adding the worksheet from your **carlislePizzaShops.xlsx** workbook. Next, export the table view to your geodatabase. Proceed with the new geodatabase table and remove the view of your Excel ® table.

The addresses in your new table are pretty well behaved (save a few typos). The *SITUS* field[2] contains the complete address for each pizza shop, but notice how each street name, place name, state abbreviation, and ZIP code element are stored together in one long text string. We need to parse these long text strings into their address elements (see Table 1).

Add five new fields to your geodatabase table: *Street* (text, 40); *City* (text, 20); *ZIP* (text, 5); *Testing123* (short integer); and *TestingTXT* (text, 70).

You now need to populate your new !Street!, !City! and !ZIP! fields. Yes, you could open an edit session and re-type everything, but you won't learn how to handle big data that way. Instead, I want you to practice using the Python parser in ArcMap's Field Calculator to parse your *SITUS* strings into their *Street*, *City* and *ZIP* elements. The two "Testing" fields you added are extras; they are safe places to try Python expressions BEFORE you try calculating your Street, City and ZIP values. To accomplish this task, you need to recognize and exploit the structure of your data. Use the commas.

Figure 5 will help you remember Python's indexing system for text strings. Table 3 presents some common string methods. String methods can be combined.

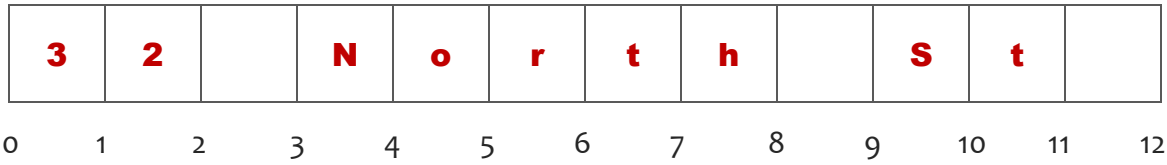


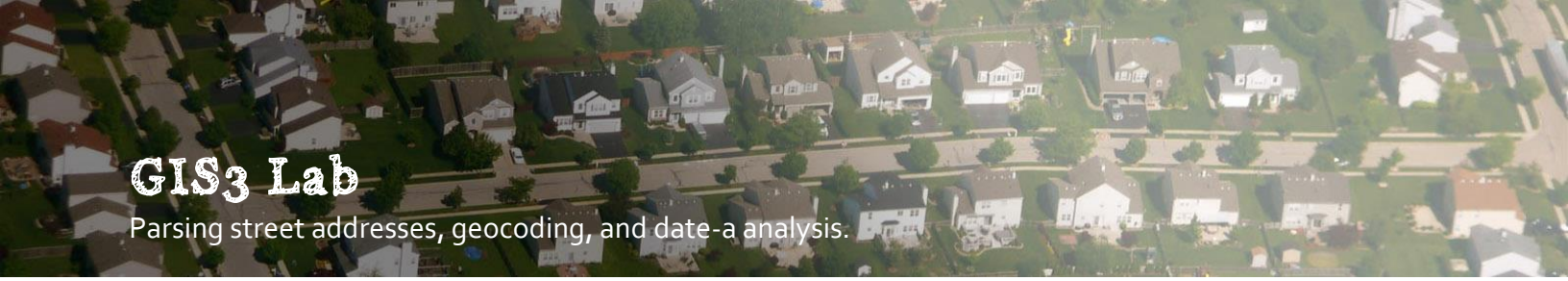
Figure 5. Python's indexing system for text strings.

Table 3. Some common methods for parsing text strings. A complete list of Python methods is available at: **http://docs.python.org/library/stdtypes.html#string-methods**

Assume we have a table that contains one record and a field named STRING.

```
STRING = "32 North St "   # notice the trailing space character in this example
```

[2] A situs address is the industry term for a physical street address, which is commonly used by emergency responders like fire and police. Although often similar, a situs address is not to be confused with a mailing address or post office address.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

You can use integer fields to calculate stuff like:

`len(!STRING!)` Returns the length of (the highest index associated with) the input string, which will include any leading or trailing space characters.

`len(!STRING!)` returns the integer 12

`!STRING!.find("sub")` Returns the highest index where substring "sub" is found.

`!STRING!.find("North")` returns the integer 3

You can use your text fields to try stuff like

`!STRING![i]` Returns the one character in the string that occurs *after* the i^{th} index.

`!STRING![4]` returns "o"

`!STRING![i:j]` Returns the substring between the i^{th} and the j^{th} indexes.

`!STRING![6:11]` returns "th St"

`!STRING![-8:]` returns "orth St "

`!STRING!.strip()` Returns a copy of the input string with all leading or trailing white spaces removed.

`!STRING!.strip()` returns "32 North St"

`!STRING!.replace("old","new")` Returns a copy of the input string with all occurrences of substring "old" replaced by substring "new".

`!STRING!.replace("Nor","Sou")` returns "32 South St "

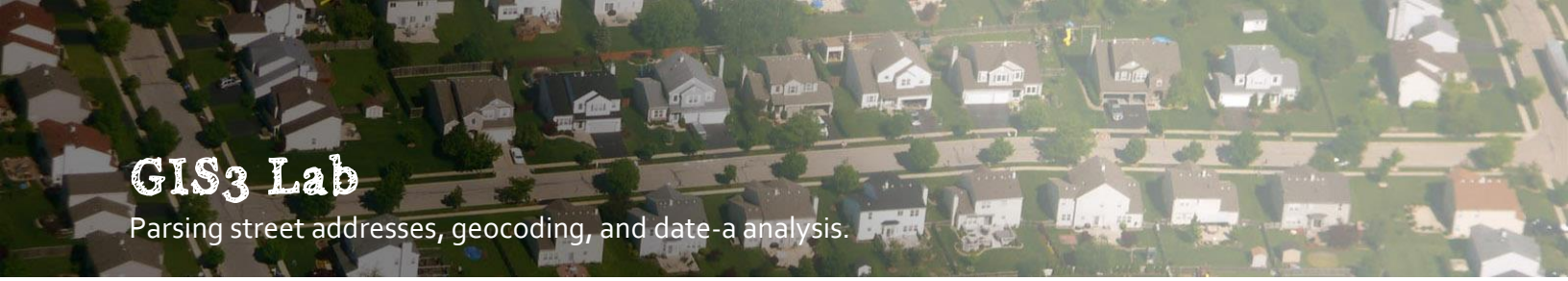
`!STRING!.replace("Nor","Sou").strip()` returns "32 South St"

`!STRING!.partition("sep")[i]` Reading from left to right, this method splits the input string at the first occurrence of "sep", and returns a 3-tuple containing the substring that occurs before the separator [0], the separator itself [1], and the substring that occurs after the separator [2]. If the substring is not found, then it will return a 3-tuple containing the entire input string [0], followed by two empty strings, [1] and [2].

`!STRING!.partition("t")[0]` returns "32 Nor"

`!STRING!.partition("t")[1]` returns "t"

`!STRING!.partition("t")[2]` returns "h St "



`!STRING!.rpartition("sep")[i]` Reading in reverse from right to left, this method splits the input string at the first occurrence of "sep", and returns a 3-tuple containing the substring that occurs before the separator [0], the separator itself [1], and the substring that occurs after the separator [2]. If the substring is not found, then it will return a 3-tuple containing the entire input string [0], followed by two empty strings, [1] and [2].

```
!STRING!.rpartition("t")[0] returns "32 North S"
!STRING!.rpartition("t")[1] returns "t"
!STRING!.rpartition("t")[2] returns " "
```

Question 2: If `STRING = "95 Church Street Bypass"`, then what should the following Python expression return? `!STRING![0:!STRING!.find("Street")].strip()`

Question 3a. Look at your Excel worksheet and find the record for Alfano's Pizza. How is Excel ® *displaying* your TOPEN value in the cell? How is Excel ® *storing* your TOPEN value in the spreadsheet? Also, is midnight stored as the beginning of a day or the end of a day?

Question 3b. Look at your geodatabase table. How are your datetime values *displayed* and *stored* in their cells? Is midnight still stored as the beginning of a day or the end of a day?

The pizza shops in places beyond Carlisle were useful to help us focus on patterns in our text strings, but we already know that our address locator is incapable of finding those addresses. Use your table **Properties** to build a **Definition query** that keeps only those records in Carlisle.

Read steps 5a and 5b before hitting any buttons.

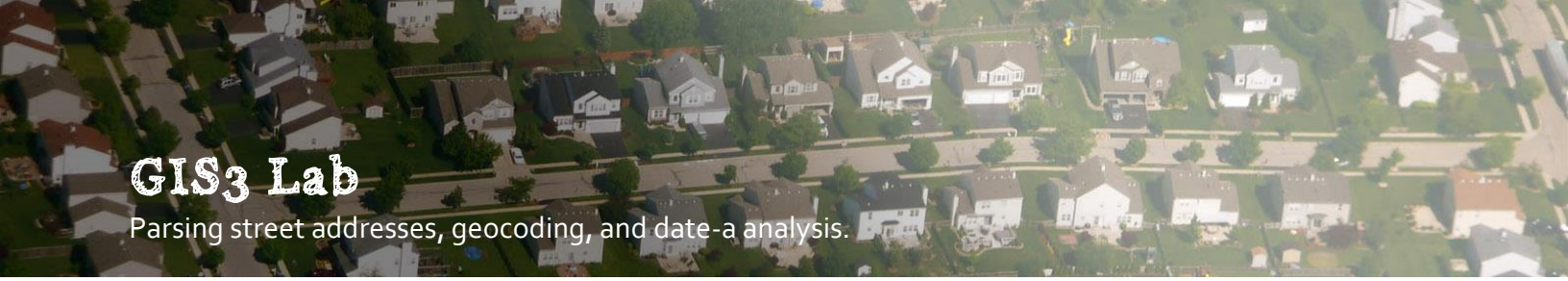
Step 5a. Match address strings in your table against the address strings in the Address Locator.

Use the Geocoding toolbar to:

1. Make sure your Address Locator is still the active Address Locator.
2. Use the **Mailbox** button to **Geocode Addresses** in bulk.
3. Map the fields in your table to the fields in the Address Locator (Street, City, ZIP).
4. Next, ensure your output features will be written to your geodatabase.
5. OK.

Step 5b. Rematch any pizza shop addresses that were not matched.

Now is when you get to see how well your addresses match and, if need be, to make manual adjustments. It is very common to have match rates in the 70-90% range; lower if you don't prepare your data well. Hitting the **[REMATCH]** button will open an interactive tool that lets you deal with any troublesome addresses on the fly.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

Question 5. How many street addresses in Carlisle (# and %) were matched 100%?

Question 6. List the pizza shops by name and address that were either Unmatched OR received a matching Score < 100%. Interrogate your input table AND your address locator to discover why each address received the score it did, then explain.

Question 7. Build a reference map (designed to fit perfectly into your report; TIF format; at least 600 DPI, LZW file compression) that shows Carlisle Borough and the location of every pizza shop in Carlisle. Don't forget to remove the arrowheads from your centerlines – those were just for you.

Question 8. Now it is time to identify the pizza shops that were not plotted correctly on your map. So, which ones?

Figure 3 shows us how a perfectly matched address might not get perfect coordinates. In Figure 3, the centerline that represents "Oak Avenue" is attributed with dual address ranges; one range of odd numbers for the left side and a range of even numbers for the right side. Now, compare the location of the building labeled "101 Oak Avenue" with the location of the 101 attributed to the left-side range; the spots are close but not identical. The best any geocoding software can do is to grab coordinates from the reference data.

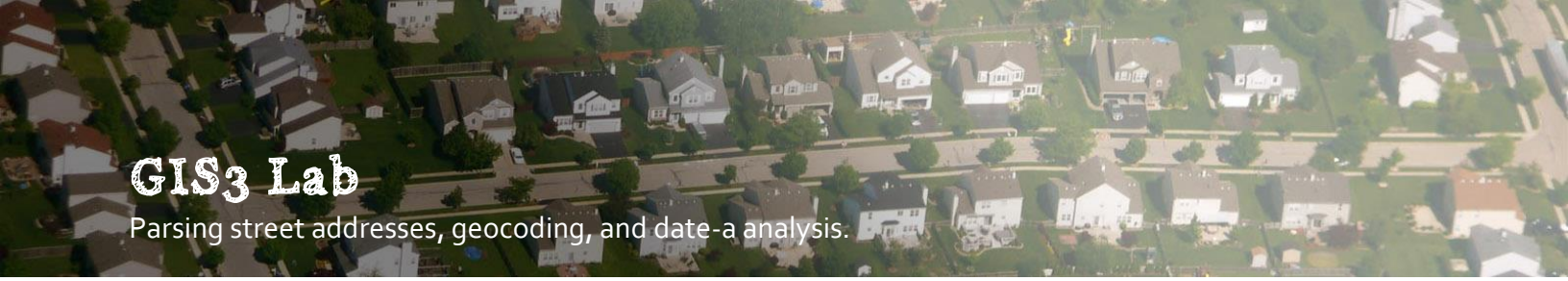
Step 6. Building more definition queries **(Bolstad, p321-331)** *using Datetime fields* **(ESRI)**

Four fields in your pizza shop attribute table are datetime fields (Datetime fields can store dates, times, or dates-and-times). Datetimes are efficient ways to store lots of information. Notice how a colon character is used to separate the three time parts (hour : minute : second). Notice how a slash character is used to separate the three date parts (day / month / year). And notice how a space character separates the date parts from the time parts.

```
10/10/2017 04:00:00
```

Datetime fields are efficient, but they require some care while querying. It is often easier to select by attributes if you reduce the problem. Pull out the just the pieces you need to make your query expressions simpler.

Add some new **short integer fields** to your pizza shop attribute table. Right-click your new integer field and open the **Field Calculator**. Switch the parsing language from VB to Python, **[Load…]** the expression contained in the file named `dataPart_py.cal`, set it up, then calculate. Use the Python expression to pull out the piece(s) of data you need to answer the following questions.



Question 9a. Build a reference map of Carlisle Borough (designed to fit perfectly into your report; TIF format; at least 600 DPI, LZW file compression) that shows where people can get pizza delivery during lunch (i.e., between 11:00 AM and 1:00 PM). After building a user-friendly figure caption, add a sentence that reports your Definition Query expression (syntax counts).

Question 9b. Read the map you built and build an accompanying paragraph that describes the spatial distribution of the lunch-time pizza delivery market.

Question 10a. Build a reference map of Carlisle Borough (designed to fit perfectly into your report; TIF format; at least 600 DPI, LZW file compression) that shows which shops in Carlisle provide dine-in service for dinner (i.e., between 5:00 PM and 8:00 PM). After building a user-friendly figure caption, add another sentence that reports your Definition Query expression (syntax counts).

Question 10b. Read the map you built and build an accompanying paragraph that describes the spatial distribution of the dinner-time pizza service market.

Question 11a. Build a reference map of Carlisle Borough (designed to fit perfectly into your report; TIF format; at least 600 DPI, LZW file compression) that shows which shops in Carlisle will deliver pizza *after* 11 PM. After building a user-friendly figure caption, add another sentence that reports your Definition Query expression (syntax counts).

Question 11b. Read the map you built and build an accompanying paragraph that describes the spatial distribution of the late-night delivery market.

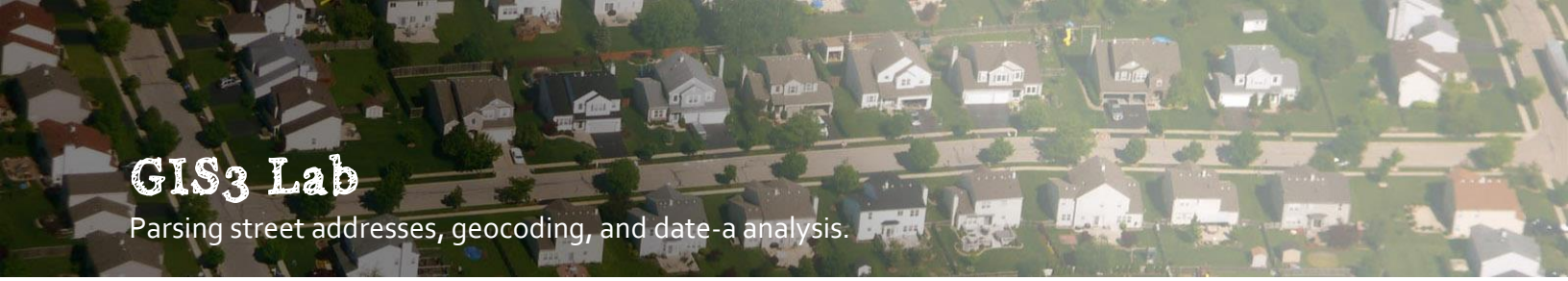
Question 12. Consult and interpret your data and your maps. Describe in words how the spatiotemporal pizza market changes throughout the day in Carlisle, from lunch, through dinner, and into the late night.

Step 7. Build URLs to Yelp!

This last step has nothing to do with geocoding.

You likely noticed that there's an attribute called "PartialURL" in your pizza shop attribute table; it contains parts of the hyperlinks that can take you to the review site Yelp.com. We can construct complete URLs from the partial URLs by adding the missing pieces of text. Start by **add**ing a **new text field**: *URL* (text, 70).

Next, right-click your new **URL** field and open the **Field Calculator**. Switch the parsing language from VB to Python, **[Load…]** the expression contained in the file named `makeURL.cal`, then calculate.



GIS3 Lab

Parsing street addresses, geocoding, and date-a analysis.

Deliverables

Complete a well-written report of the lab exercise. Include your name, date, and page numbers. Your report should include five sections and headings: Purpose, Objectives, Methods and Data, Results and Answers, and Summary. Provide concise purpose and objectives using your own words, and describe the important methods and input data you used to address the tasks listed above. In the Results and Answers section, address any issues or questions prompted during the lab. Include tables and figures in the same order you refer to them. Summarize your work in the *Summary* section. Be sure to identify anything that you learned, found interesting or difficult, or anything that remains problematic. Use the summary to talk to me – or to ask lingering questions.

All lab reports should be typed and printed on 8.5" by 11" stock. Before drafting your report, set your right and bottom page margins to be 0.7"; set your left margin to 1.2"; set your top margin to 1.2"; and set your header margin to 0.7". Put your name and the lab title in the document header. Set the normal font face to be Candara, Bookman Antiqua, or Georgia; never use Times New Roman or any kind of *decorative font*. Also, set the normal font size to be 11 points. Use 1.5 line spacing. Include page numbers on every page. Major section headings should be in **bold face**. All figures and tables must be inserted into the body of your report and conform to the formatting and margin requirements. Table columns that hold numeric values should be right-justified with decimal points aligned; pad values with zeroes if necessary.