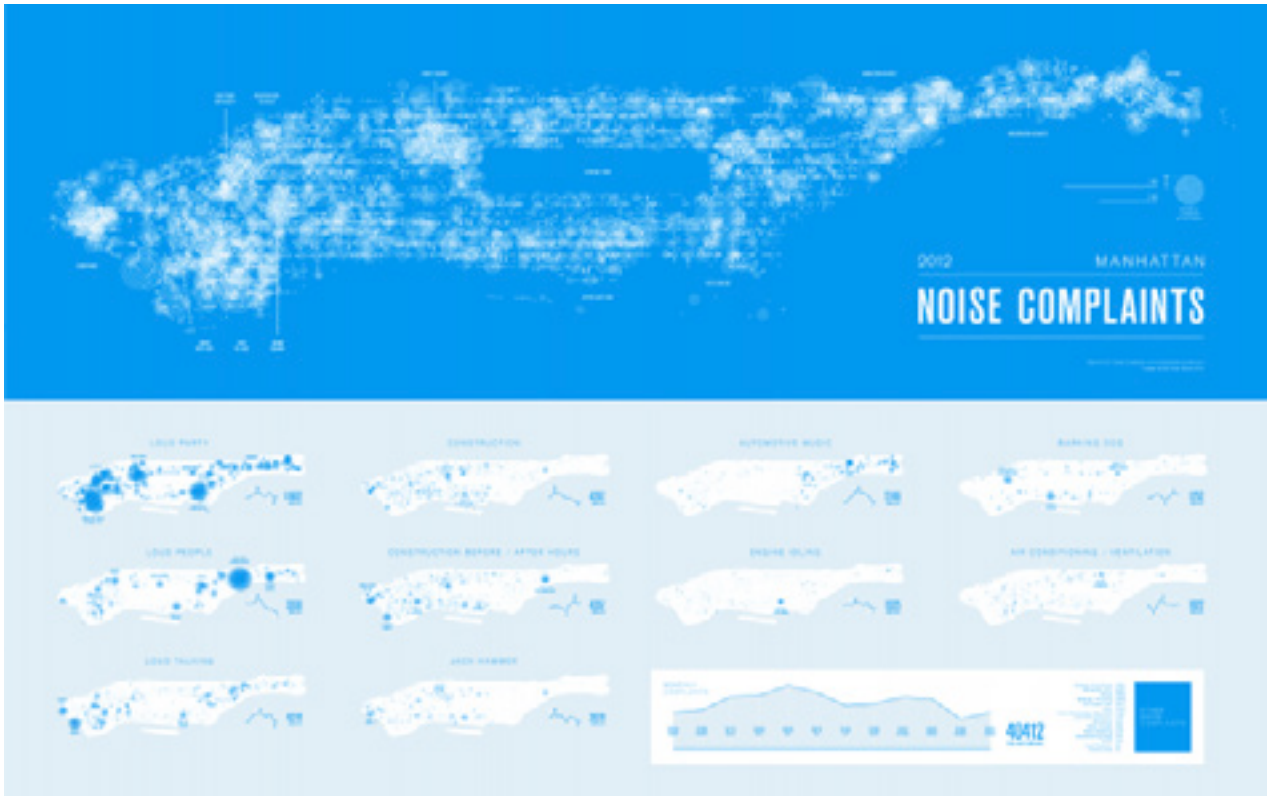
- 3 WORKSHOPS- 5hrs/each
- CURRICULM BY: Meg Studer



Noise Complaints, 2012 by Karl Sluis, 311 data from Open Data NYC, (visualizing.org)

This introduction to Processing focuses on 2D data plotting and visualization. By the end of this course students will know how to a) access, manipulate, and display different data types using Processing, b) export graphics, animations, and interactions for presentation and integration with the Adobe Suite and, with a basic conceptual grasp of code, c) be able to move and choose between GIS, Processing, and Excel for communicating spatio-temporal processes.

NO CODING EXPERIENCE NECESSARY! THIS IS AN INTRODUCTION!

Landscape is awash with parameters and data—climate trends, population distributions (flora, fauna, mineral, human), species attributes, logistical networks, and so on. As designers, we must be able to communicate theses relationships when explaining sited strategies and particular interventions. This introduction to Processing is designed to help you access, explore and think about how to plot relevant, publicly accessible data.

While GIS provides a great interface for spatial analysis, its statistical displays and temporal features are very limited. Processing offers an alternate plotting environment that, while offering x/y or lat/long referencing, enables you to prioritize and feature other types of relationships, time-series, and develop different relational-displays. The end goal is of this workshop is to both know what's possible, given code structures and datatypes, to get started experimenting with data displays for your semester projects, and to understand the conceptual structures of code in order to leap to other freeware and coded environments for communication/visualization like the javascript based d3 or mapbox.

DO NOT BE INTIMIDATED. YOU WORK & THINK WITH THIS INFO ALL THE TIME.

Processing just gives you another, integrated way of imaging it.

WORKSHOP BASICS:

Processing.org:

Download latest version at- https://www.processing.org/download/
Processing 'plug-ins' are call Libraries- http://www.processing.org/reference/libraries/
Grab pieces of code from Examples- http://www.processing.org/examples/
Look up code and functions at References- http://www.processing.org/reference/
Watch Shiffman's video's at- http://hello.processing.org/

Background Readings:

Fry, Ben.
Visualizing Data: Exploring and Explaining Data with the Processing Environment.
1st ed. O'Reilly Media, 2008.

Although we will start with different data and simpler structures, the core visualization processes covered in this workshop are can be found in *Visualizing Data*. The text is useful, as are Fry's cleaned data links, but the code is approximately five years old and thus can be a bit antiquated. Don't imagine that you'll be able to verify Fry's original data sources; they have long since moved.

Reas, Casey, and Ben Fry.
Getting Started with Processing. 1st ed. Make, 2010.

This will give you more background on the overall environment and structure of Processing. If you have questions about the structure of code we are using, this will walk you through the background structures in an easy to understand format. It's a good complement to Shiffman's text/videos.

Shiffman, Daniel.
A Beginner's Guide to Programming Images, Animation, and Interaction
1st ed. Morgan Kaufmann, 2008.

Also videos: https://vimeo.com/channels/introcompmedia

This will give you more background on the overall environment and structure of Processing. If you have questions about the structure of code we are using, this will walk you through the background structures in an easy to understand format. It's a good complement to *Getting Started*.

Sourcing Code & Code Information:

GitHub:
https://github.com/

This is where developers trade info, so sign up for a free account and see what you can find (both free datasets, libraries, and code)...

Stack Overflow:
http://stackoverflow.com/

This is where developers trade questions, so sign up for a free account and see what you can find (both questions and answers on processing)...

OpenProcessing:
http://www.openprocessing.org/

This is where processing users trade sketches, so sign up for a free account and see what you can find (to reverse engineer, comment out, credit, and adapt to your own ends)...

Data Sources & Visualization Inspiration at end of syllabus

Zipped Workshop Processing Sketches for download at
http://www.siteations.com/processing/DataWorkshop.zip

WK 1-BASIC PLOTTING

- examples from the plotting environment

- computation structures
  - references
  - variables
  - conditionals
  - operators
  - syntax
- comparative structures

- basic processing for plotting
  - interface
  - shapes
  - display
  - loops
  - reading tables- core functions

- re-constructing the 311 example
  - finding files: NYC Open data
  - reading tables- core functions
  - geoplotting data
  - symbology
  - internal & external filtering of data
  - parsing/saving new files

WK 2-MATH, INTERACTIONS, & MULTIPLE DISPLAY FORMS

- shifting from 311 dog calls to other NYC Open Data

- simple math from tables
  - StringLists
  - InterDicts
  - Arrays

- legible & interactive information
  - labels and font use
  - mouse and keyboard functions
  - printing and export options for jpg, png, and pdf

- more advanced information & symbology
  - finding files: NYC Open data for Boilers and Oil Use (CSV tables)
  - quantitative thresholds and symbols
  - scaled keys
  - more time-based variables with basic addition/interaction
  - secondary filters- typologies and oil use by zipcode
  - representational forms- data roses
  - combining modulo and raw BTU quantities for final data
  - visually cleaning your geographic 'dashboard'

WK 3-LIVE DATA, JSON & GEOJSONs, CREATIVE COPYING

- Moving toward correllation and arguments with data

- Introduce API and live webdata, json format
  - NYC Tree Count-retrieving data in json form to pair with Boiler emissions
  - Other Socrates examples- simple trash collection numbers as bar graphs
  - Other common sources in json (weather, flickr, data.gov, etc. etc.)
  - Temporal limits and paywalls- workarounds for data retrevial
  - example: historical wind data from weather underground
-
- Additional API sources and other ways to engage json
  - public resources (socrates, nyc open data, data.gov)
  - popular applications (flickr, twitter, yahoo api queries, etc.)
  - easy json and geojson uses (to build off processing)- leaflet, mapbox, d3
  - processing for csv and table conversion to json and geojson

- Copying code, understanding structures
  - open source community and standards
  - resoures: github, stackoverflow, others
  - citation practices, code credits
  - example: Ben Fry's network graphs + NYC trash disposal trajectory json

Useful Data Sources:

Government/Municipal- varied types:

https://nycopendata.socrata.com/ new york city

https://data.ny.gov/ new york state

http://catalog.data.gov/dataset us national clearinghouse

http://data.un.org/ international trade/development databases

http://www.epa.gov/developer/index.html links to epa api's

Non-Profit/Press- varied types:

http://www.theguardian.com/news/datablog/ guardian (uk)

http://developer.nytimes.com/docs new york times

http://www.programmableweb.com/apitag/environment various environmental

http://sunlightfoundation.com/api/ sunlight foundation

and so on...

Commercial- varied types:

http://visualizing.org/data/ visualizing.org

http://developer.yahoo.com/yql/ yahoo system for web queries

http://www.flickr.com/services/api/ flickr

https://dev.twitter.com/ twitter

http://www.wunderground.com/weather/api/ weather underground

https://developer.foursquare.com/ foursquare

https://developers.google.com/maps/ google various api's (maps, etc.)

Visualization Inspiration:

Varied types:

http://www.visualizing.org/explore visualizing dot org

https://vimeo.com search visualization animations

http://www-958.ibm.com/software/analytics/manyeyes/ ibm's many-eyes

http://www.openprocessing.org/ openprocessing

Miscellaneous Data Representation Project samples:

The following list of links comes from the Processing Data Representation Course at NYU's Interactive Telecommunications Program. Most are working with more dynamic data than we'll cover (api twitter feeds, facebook streams, the gps/gpx iphone tracks), but they are working from the same building blocks. Many will have process videos and code for copying/editing. Explore! (Don't be overwhelmed: the post are most recent to least, skim back to the beginning of term for simpler examples).

http://itpcourtney.com/?p=563
http://www.craigprotzel.com/?cat=24
http://supboon.com/blog/?cat=10
http://dougkanter.wordpress.com/category/itp/data-rep/

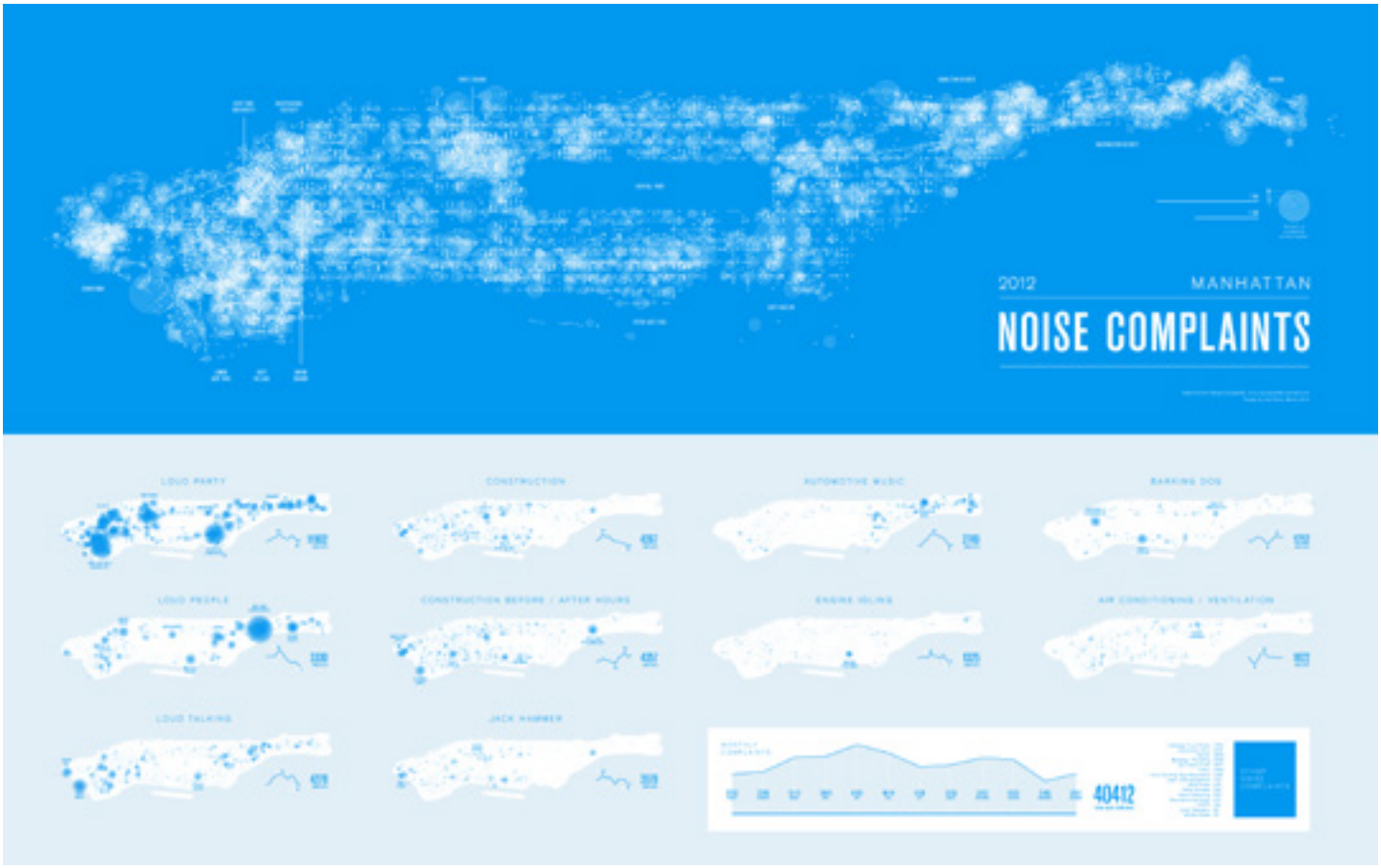
WK 1-BASIC PLOTTING

- examples from the plotting environment

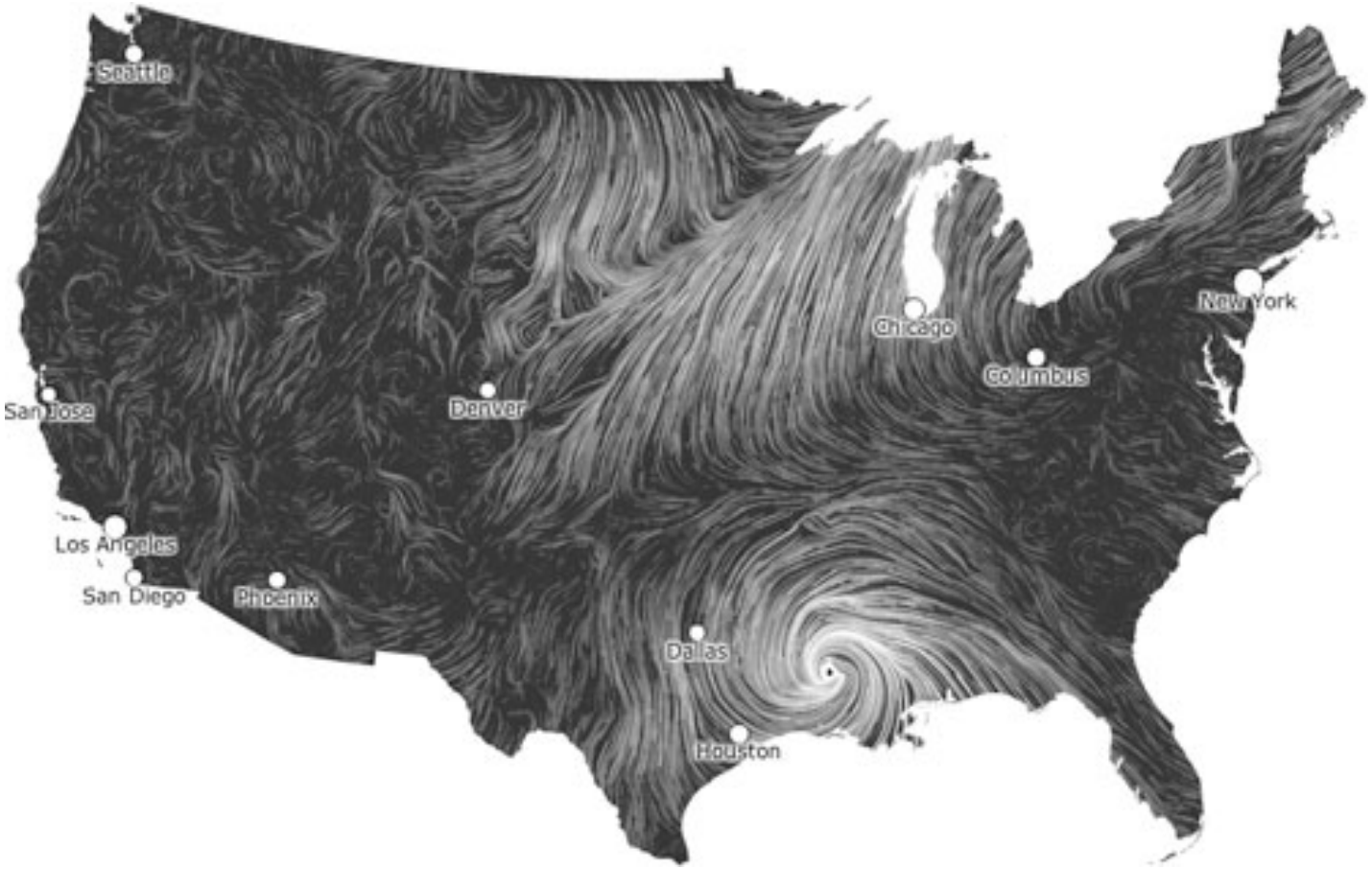
- computation structures
  - references
  - variables
  - conditionals
  - operators
  - syntax
- comparative structures

- basic processing for plotting
  - interface
  - shapes
  - display
  - loops

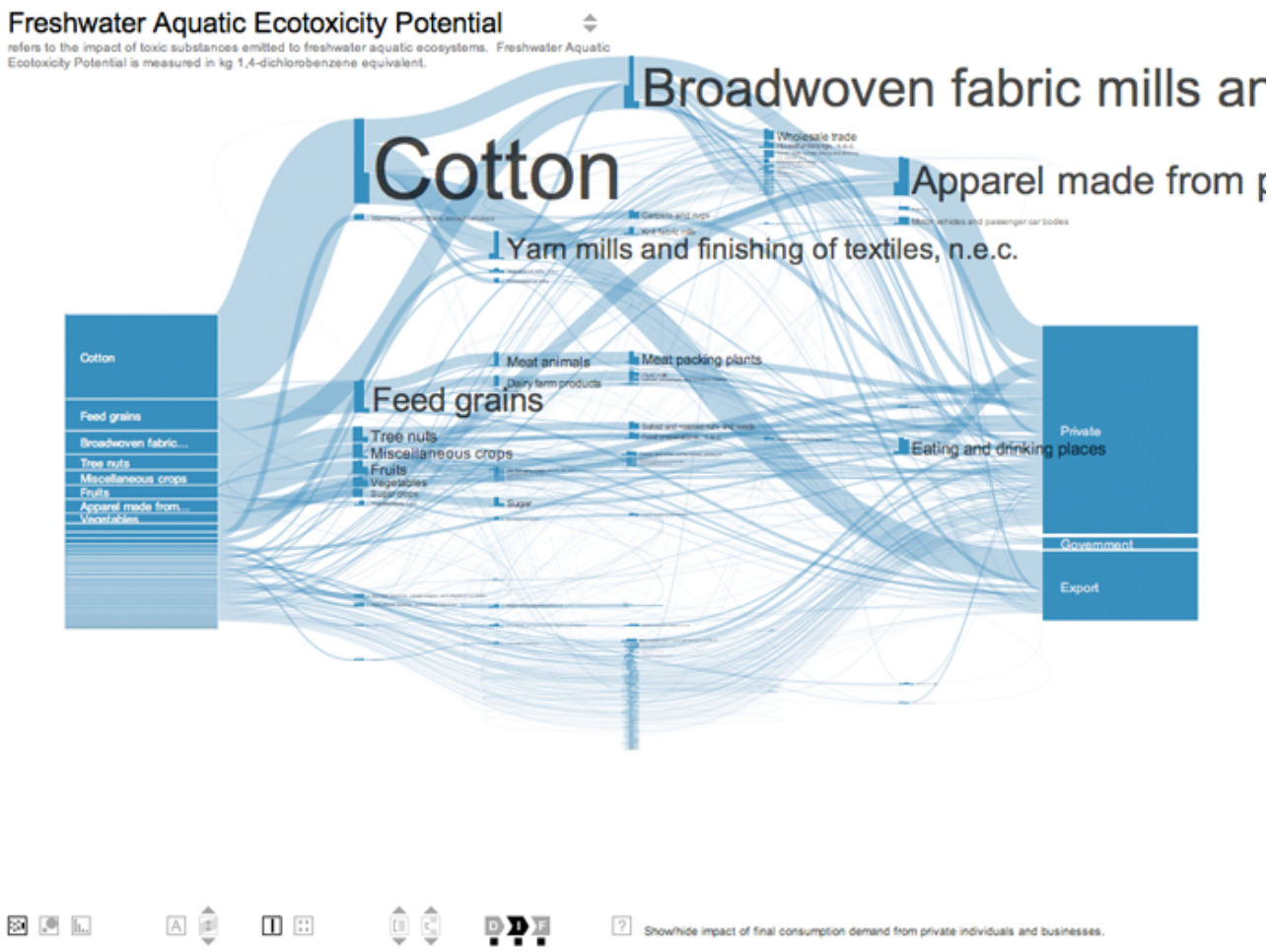
- re-constructing the 311 example
  - finding files: NYC Open data
  - reading tables- core functions
  - geoplotting data
  - symbology
  - internal & external filtering of data
  - parsing/saving new files



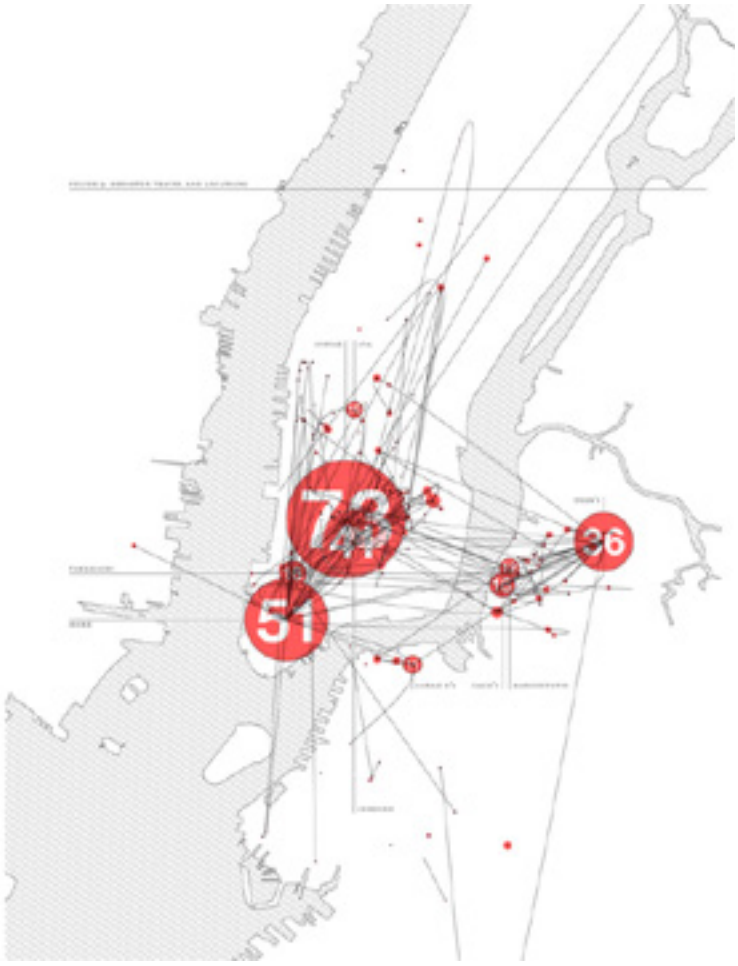
Noise Complaints, 2012 by Karl Sluis
311 data from Open Data NYC (http://www.visualizing.org)



Wind Map by Hint.fm
(http://hint.fm/wind/)



Economy Map by Jason Pearson, TRUTHstudio (http://economymap.org/)



Various Dashboards by Nicholas Feltron (http://feltron.com/)

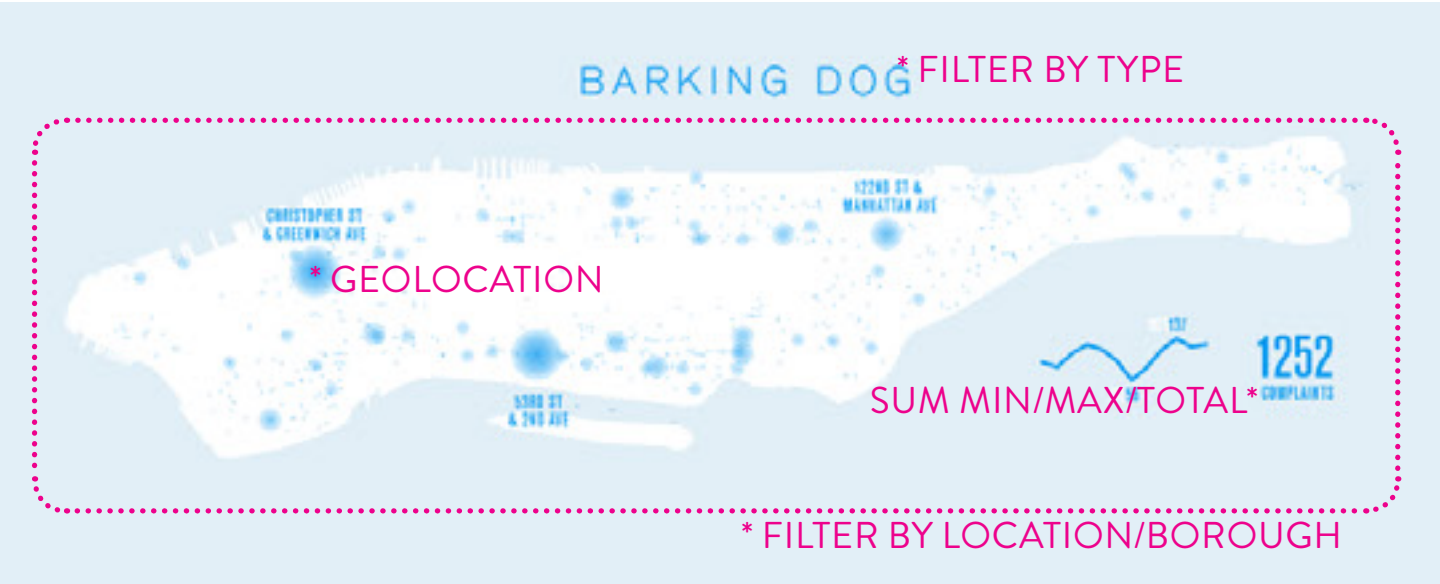
see also the video at: https://vimeo.com/70800507

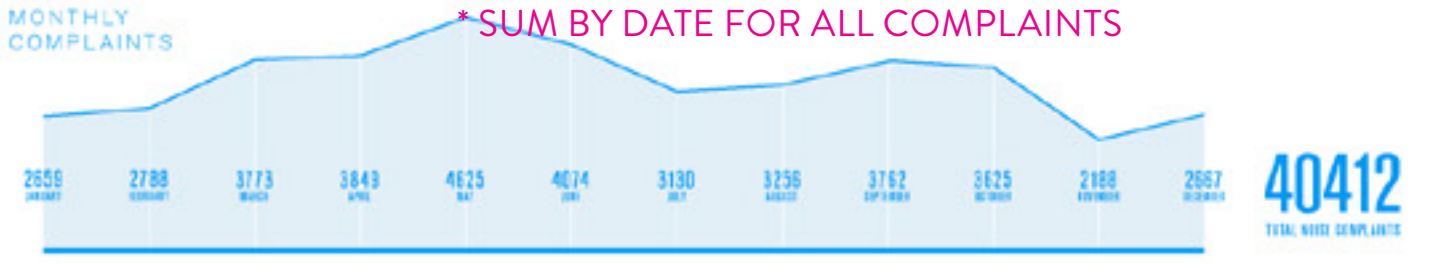
WK 1

COMPUTATION STRUCTURE *(IS FAMILIAR!)*

- references/sources
- variables
- conditionals
- operators
- syntax

- comparative structures



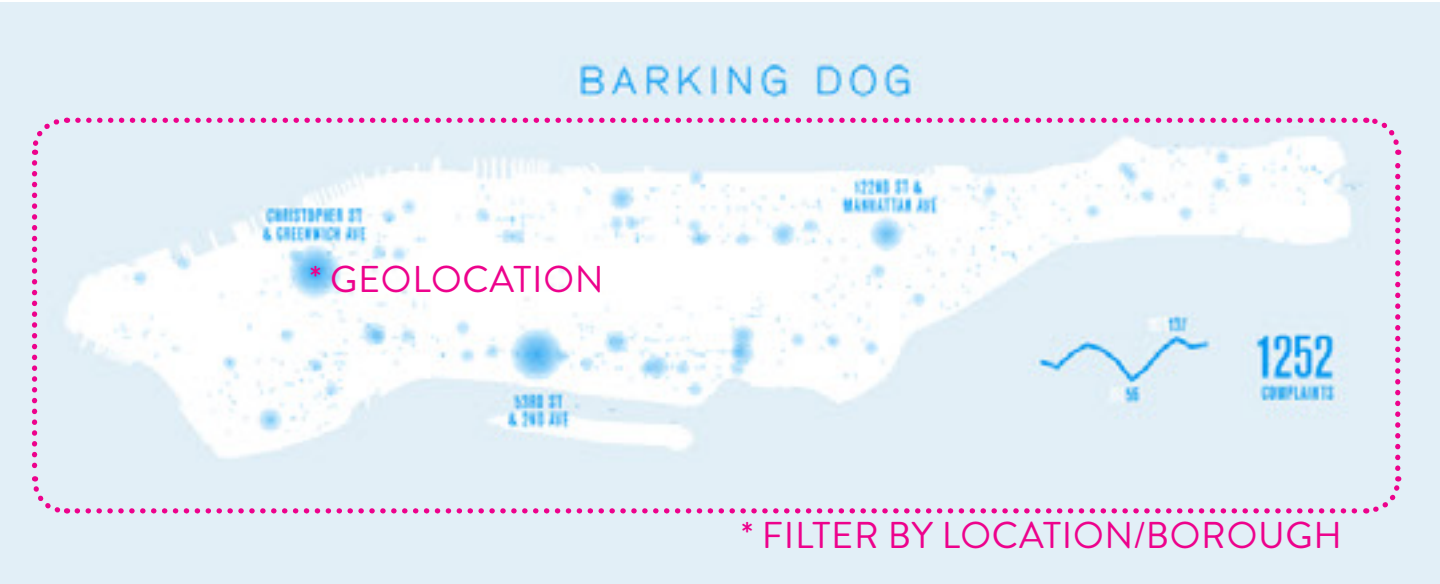


311 EXCERPT: BARKING DOG CALLS FROM 2012

FAMILIAR FEATURES:

- geolocation of complaint calls by lat/long and or NYC street grid
- filtering of all 311 calls to get dog barking complaints for Manhattan

- calculation of calls/location to find:
  - max/min values per site and per date
  - complaint sums across area and type of complaint
  - temporal plots of 311 calls by date for dogs and all calls



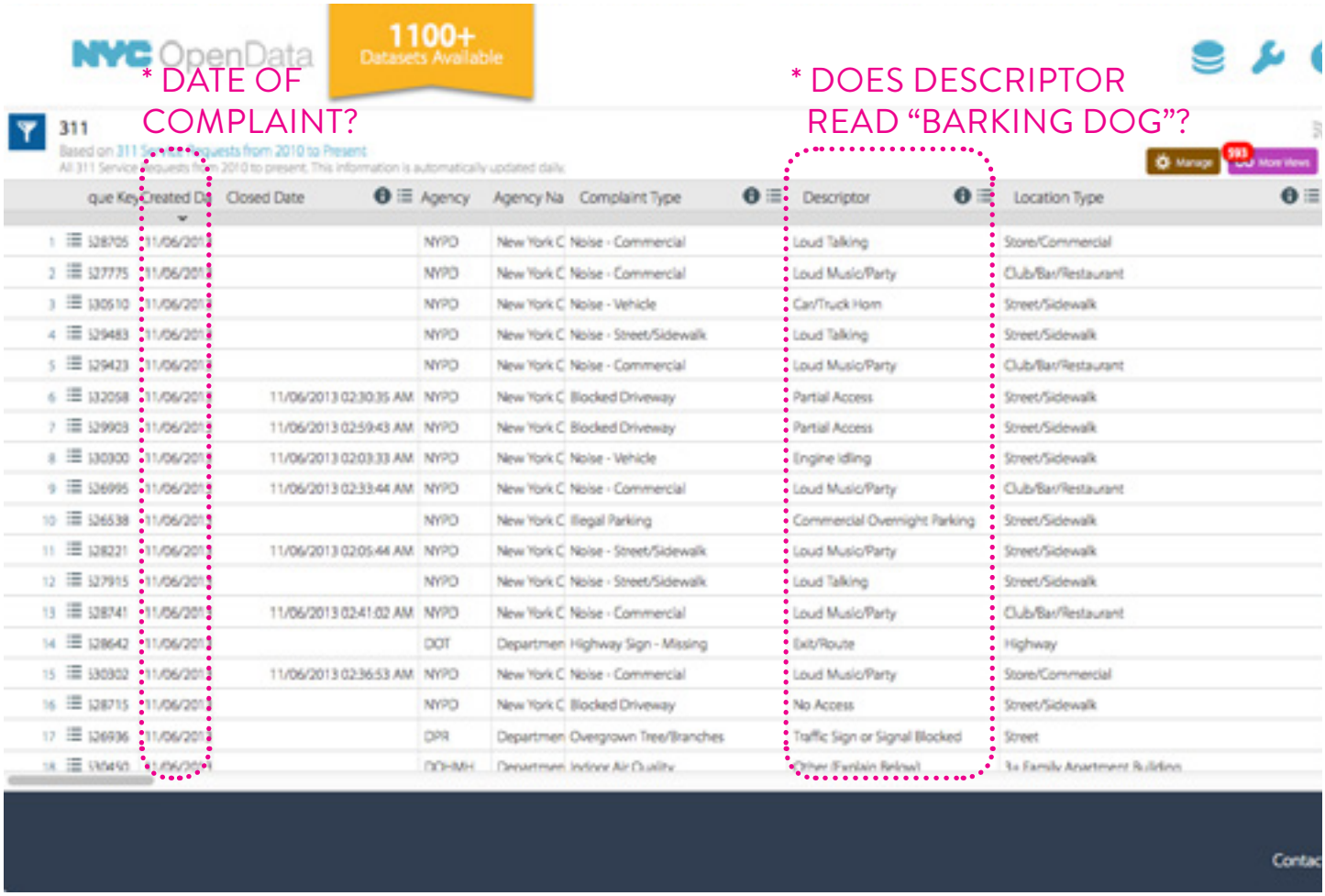
311 EXCERPT: GEOLOCATION & FILTERING

FAMILIAR FEATURES:

- geolocation of complaint calls by lat/long and or NYC street grid

EASILY EXTRACTED FROM NYC OPEN DATA

https://data.cityofnewyork.us/Social-Services/311/wpe2-h2i5



311 EXCERPT: GEOLOCATION & FILTERING

NOTE COLUMNS FOR FILTERING:

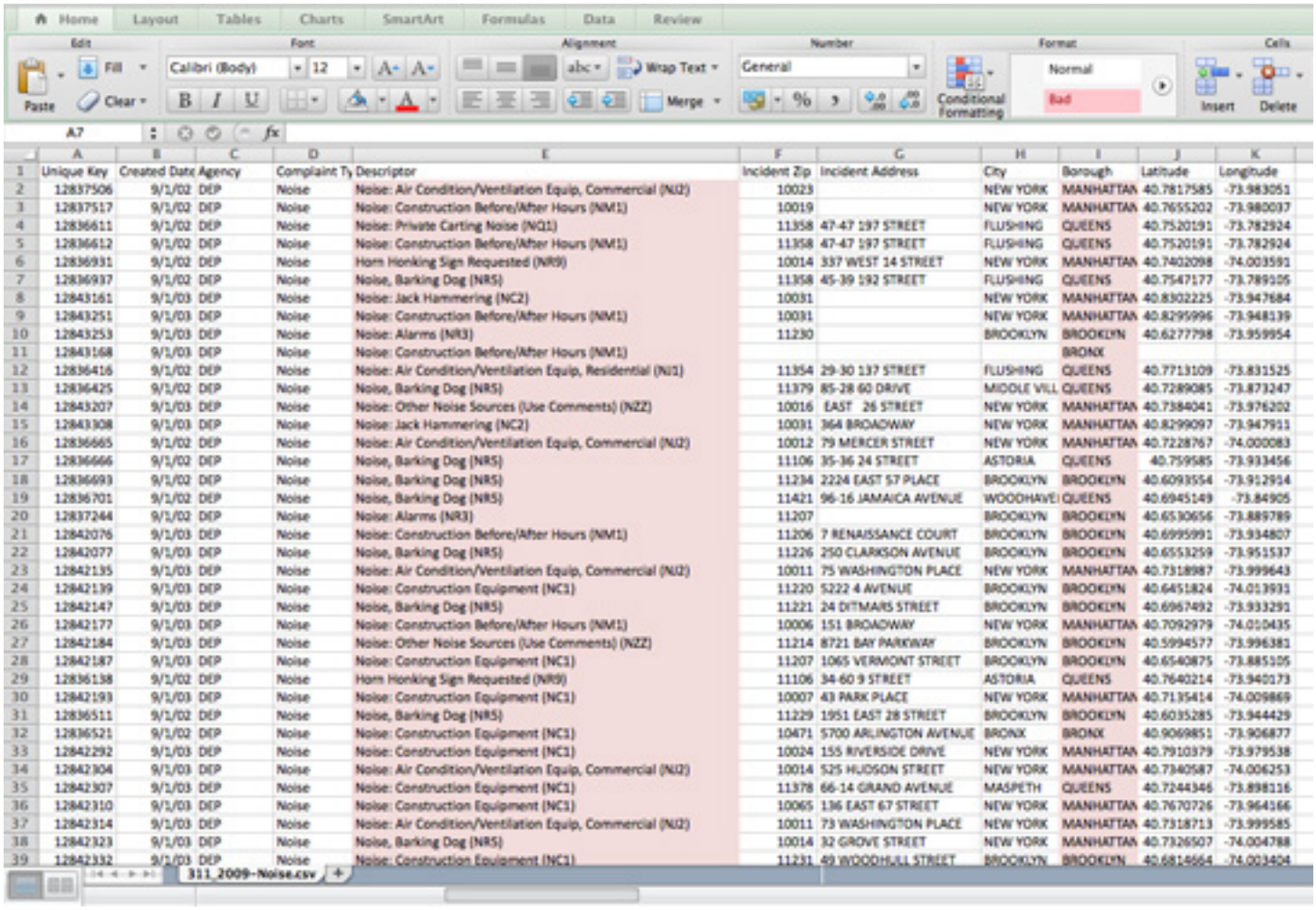
- description is directly tagged "barking dog"
- latittude is give, longitude is given
- borough is given as "manhattan"
- date of call registered under "complaint opened"

311 EXCERPT: GEOLOCATION & FILTERING

IN PROCESSING, GEOLOCATION IS MERELY FILTERING:

a) specifying columns to search
b) specifying rows to read values from
c) setting up VARIABLES to hold those values

Worry not, we'll explore writing those functions during the third section of Week 1.



Unique Key	Created Date	Agency	Complaint Ty	Descriptor	Incident Zip	Incident Address	City	Borough	Latitude	Longitude
12837506	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10023		NEW YORK	MANHATTAN	40.7817585	-73.983051
12837517	9/1/02	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10019		NEW YORK	MANHATTAN	40.7655202	-73.980037
12836611	9/1/02	DEP	Noise	Noise: Private Carting Noise (NQ1)	11358	47-47 197 STREET	FLUSHING	QUEENS	40.7520191	-73.782924
12836612	9/1/02	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11358	47-47 197 STREET	FLUSHING	QUEENS	40.7520191	-73.782924
12836931	9/1/02	DEP	Noise	Horn Honking Sign Requested (NR9)	10014	337 WEST 14 STREET	NEW YORK	MANHATTAN	40.7402098	-74.003591
12836937	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11358	45-39 192 STREET	FLUSHING	QUEENS	40.7547177	-73.789105
12843161	9/1/03	DEP	Noise	Noise: Jack Hammering (NC2)	10031		NEW YORK	MANHATTAN	40.8302225	-73.947684
12843251	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10031		NEW YORK	MANHATTAN	40.8295996	-73.948139
12843253	9/1/03	DEP	Noise	Noise: Alarms (NR3)	11230		BROOKLYN	BROOKLYN	40.6277798	-73.959954
12843168	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)				BRONX		
12836416	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Residential (NJ1)	11354	29-30 137 STREET	FLUSHING	QUEENS	40.7713109	-73.831525
12836425	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11379	85-28 60 DRIVE	MIDDLE VILL	QUEENS	40.7289085	-73.873247
12843207	9/1/03	DEP	Noise	Noise: Other Noise Sources (Use Comments) (NZZ)	10016	EAST 26 STREET	NEW YORK	MANHATTAN	40.7384041	-73.976202
12843308	9/1/03	DEP	Noise	Noise: Jack Hammering (NC2)	10031	364 BROADWAY	NEW YORK	MANHATTAN	40.8299097	-73.947911
12836665	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10012	79 MERCER STREET	NEW YORK	MANHATTAN	40.7228767	-74.000083
12836666	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11106	35-36 24 STREET	ASTORIA	QUEENS	40.759585	-73.933456
12836693	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11234	2224 EAST 57 PLACE	BROOKLYN	BROOKLYN	40.6093554	-73.912914
12836701	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11421	96-16 JAMAICA AVENUE	WOODHAVEI	QUEENS	40.6945149	-73.84905
12837244	9/1/02	DEP	Noise	Noise: Alarms (NR3)	11207		BROOKLYN	BROOKLYN	40.6530656	-73.889789
12842076	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11206	7 RENAISSANCE COURT	BROOKLYN	BROOKLYN	40.6995991	-73.934807
12842077	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	11226	250 CLARKSON AVENUE	BROOKLYN	BROOKLYN	40.6553259	-73.951537
12842135	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10011	75 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318987	-73.999643
12842139	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11220	5222 4 AVENUE	BROOKLYN	BROOKLYN	40.6451824	-74.013931
12842147	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	11221	24 DITMARS STREET	BROOKLYN	BROOKLYN	40.6967492	-73.933291
12842177	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10006	151 BROADWAY	NEW YORK	MANHATTAN	40.7092979	-74.010435
12842184	9/1/03	DEP	Noise	Noise: Other Noise Sources (Use Comments) (NZZ)	11214	8721 BAY PARKWAY	BROOKLYN	BROOKLYN	40.5994577	-73.996381
12842187	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11207	1065 VERMONT STREET	BROOKLYN	BROOKLYN	40.6540875	-73.885105
12836138	9/1/02	DEP	Noise	Horn Honking Sign Requested (NR9)	11106	34-60 9 STREET	ASTORIA	QUEENS	40.7640214	-73.940173
12842193	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10007	43 PARK PLACE	NEW YORK	MANHATTAN	40.7135414	-74.009869
12836511	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11229	1951 EAST 28 STREET	BROOKLYN	BROOKLYN	40.6035285	-73.944429
12836521	9/1/02	DEP	Noise	Noise: Construction Equipment (NC1)	10471	5700 ARLINGTON AVENUE	BRONX	BRONX	40.9069851	-73.906877
12842292	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10024	155 RIVERSIDE DRIVE	NEW YORK	MANHATTAN	40.7910379	-73.979538
12842304	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10014	525 HUDSON STREET	NEW YORK	MANHATTAN	40.7340587	-74.006253
12842307	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11378	66-14 GRAND AVENUE	MASPETH	QUEENS	40.7244346	-73.898116
12842310	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10065	136 EAST 67 STREET	NEW YORK	MANHATTAN	40.7670726	-73.964166
12842314	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10011	73 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318713	-73.999585
12842323	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	10014	32 GROVE STREET	NEW YORK	MANHATTAN	40.7326507	-74.004788
12842332	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11231	49 WOODHULL STREET	BROOKLYN	BROOKLYN	40.6814664	-74.003404

311 EXCERPT: GEOLOCATION & FILTERING

IN EXCEL, YOU'VE LIKELY DONE SOMETHING SIMILAR:

a) specifying columns to search
b) specifying rows to read 'value' from
c) showing or spatially compacting those rows

311 EXCERPT: **VARIABLES FOR HOLDING INFO**

VARIABLES types: *variable definition*
variable example

int = integers (whole numbers= 0, 1, 2, 3, etc.)
ex. int x=45;

float= floating decimals (numbers with decimal info= 1.235, 3.147, .06892)
ex. float x=45.231;

String= string of characters (words or numbers to be read as words= hello, 10005)
ex. String x="forty five point two three one"; or String x="45.231";

char= character (single letter or key-strike= k)
ex. char x="4";

Boolean= a boolean value of true or false (basic binary logic of computing off/on)
ex. Boolean x=true; or Boolean x=false;

color= color variables in rgb alpha (color as (255,0,0,100) which is r,g,b,alpha 0-255)
ex. color x=color(255,0,0,127); or color x is red with 50% opacity

double= a longer version of float with over 8 places of info
ex. double x=45.23186789039326753; (useful for very specific decimal lat/long)

long =a longer version of interger with over 8 places of info
ex. long x=45231867890393267 53;

byte =serial info not decoded into ascii (still binary, not alphabetic/numeric)
ex. byte x=00101100; (used when talking w/ microcomputers, other devices)

find @ http://www.processing.org/references/

311 EXCERPT: **VARIABLES, CODING SYNTAX**

VARIABLE USE: *coding syntax*

simple structure

variable type + variable name (your choice) = initial value;
[variable for use, name is fixed & unique] = [value for dynamic redefinition]

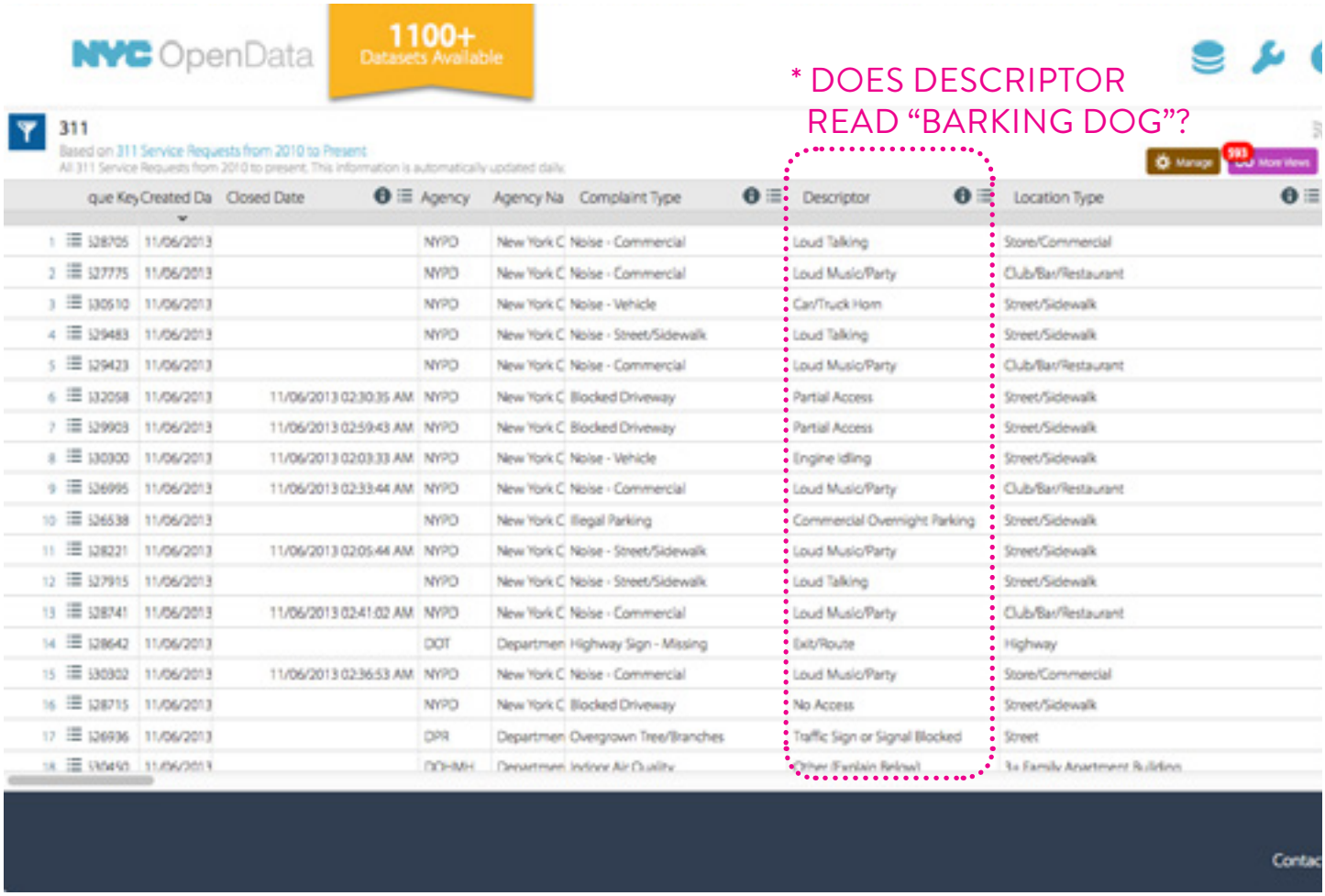
ex.

```
int x=1;
int y=3;
int z= x + y;
// here  int z= 4;
// are comment marks
/* for multilines use backslash asterick and its mirror */
x++; // ++ means adding one to an exsiting value
z=x+y;
// here int z=5;
```

a few notes on syntax:

- as seen above //, /*, and */ allow for comments in code
- all lines of code must end with a semi-colon ;
- the first time a variable is used it must be initialized with the variable type
- after initialization it can be used without the variable type
- each block of code is read top to bottom, hence dynamically defined variables

variables nest and can be dynamically redefined to build complexity



311 EXCERPT: **CONDITIONALS, CODING USE**

CONDITIONAL USE: *coding syntax*

In order to filter data for variables,
we specify conditions by asking questions with CONDITIONALS:

Does the column title read "Longitude", if so...?

Is there a value for "Closed Date", row 75, if so...?

311 EXCERPT: **CONDITIONALS FOR FILTERING INFO**

CONDITIONAL types: *conditional definition*
conditional example

if = tests a statement (if statement is true (x is less than 6), do something)

ex.

```
if (x<6){
String y="true";
}
```

else = covers false responses to 'if'(statement is true (x is less than 6), do A, else B)

ex.

```
if (x<6){
String y="true";
} else {
String y="false";
}
```

else if= covers false responses to 'if' in advance of 'else'

ex.

```
if (x<6){
String y="true";
} else if (x>6 && x<10){
String y="ambiguous";
} else {
String y="false";
}
```

switch= choses between finite options (not often used)

find @ http://www.processing.org/references/

311 EXCERPT: **OPERATORS FOR CONDITIONALS**

OPERATOR types: *operator definition*

operator examples (true statements)

RELATIONAL: *operators here define statements in relation to values*

> great than
ex. int x=45;
if (x>25){. . .}

< less than
ex. int x=45;
if (x<65){. . .}

=> equal to or great than
ex. int x=45;
if (x=>45){. . .}

=< equal to or less than
ex. int x=45;
if (x=<45){. . .}

== is or equality
ex. int x=45;
if (x==45){. . .}

=! Is not or inequality
ex. int x=45;
if (x=!65){. . .}

LOGICAL: *operators here link statements for finer accessment in conditionals*

&& And
ex. int x=45;
if (x==45 && x=>25){. . .}

|| Or
ex. int x=45;
if (x=!65 || x==45){. . .}

! Not
ex. int x=45;
if (x==45 ! x==65){. . .}

find @ http://www.processing.org/references/

311 EXCERPT: **MORE ON CODING SYNTAX**

CODING SYNTAX: *more basics*

( ) (parentheses) mathematical use, brackets for variables and statements
ex. float x=(32-21.45)/5.25;

[ ] (square brackets) for array use, allows more than one variable
ex. float [ ] x=new float[3];
x[0]=.25;
x[1]=.50;
x[2]=.75;

alternate form float [ ] x = { .25 , .50 , .75 };

{ } (curly brackets) encloses a block of code, like conditionals, arrays, functions
ex. if (x==45){
String y="true";
} // conditional

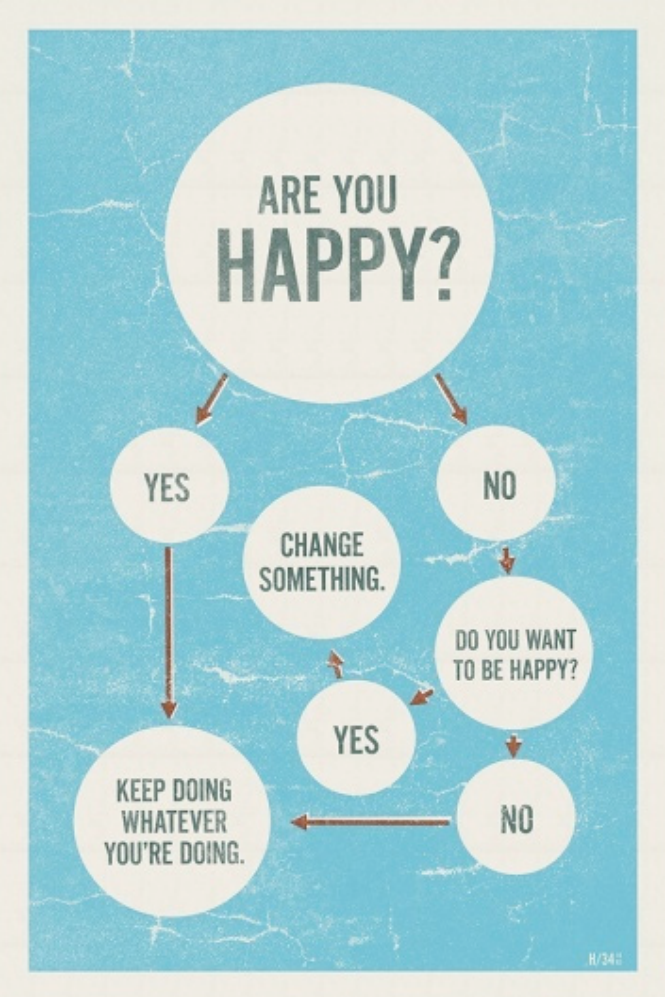
ex. void setup ( ) {
// all setup code goes here
} // functions

, (comma) separates values of variables, values in arrays, see above

// & / */ (single & multiline comments)* notes in your code

; (semicolon) end of line *= (assign)* set variables

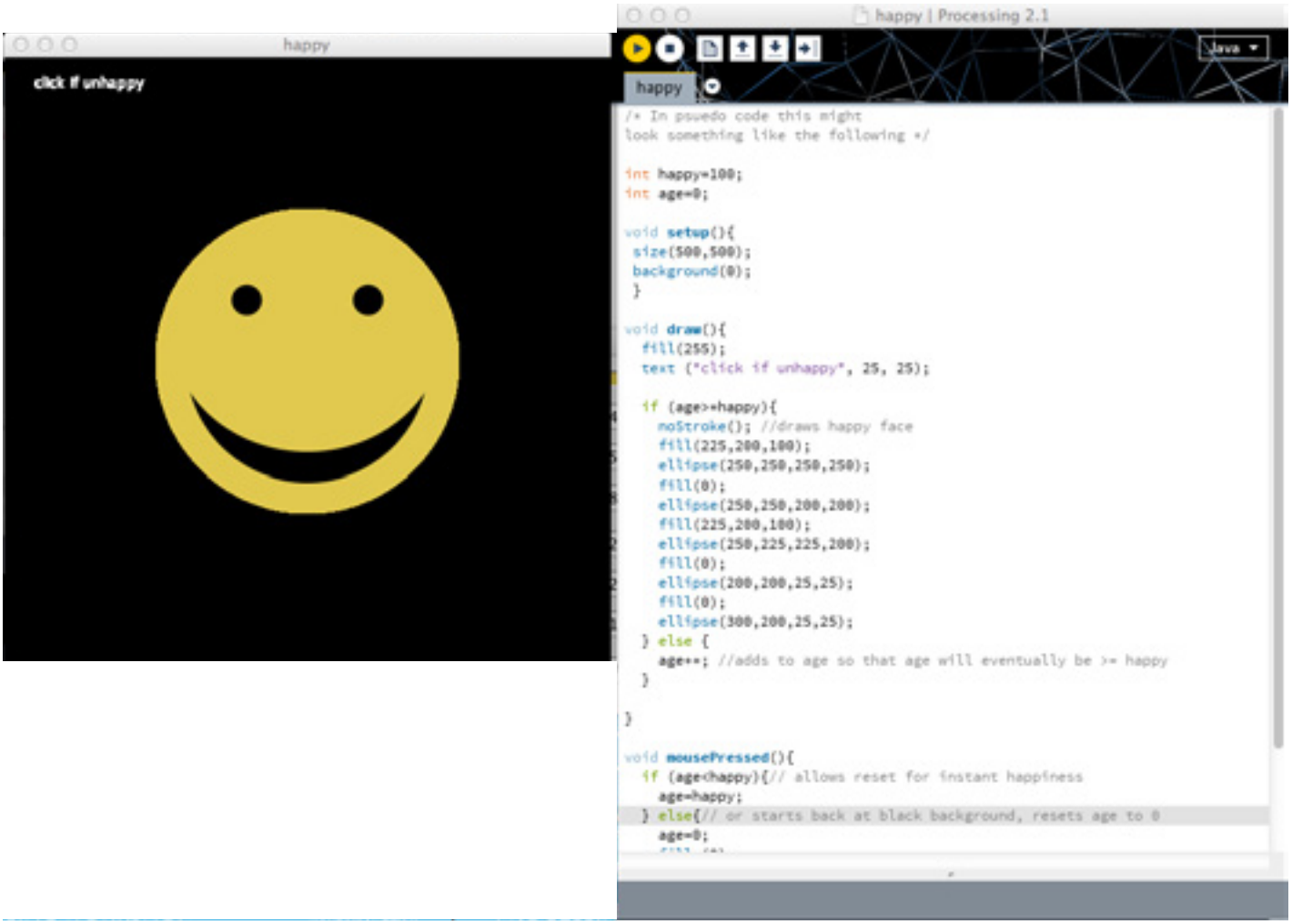
find @ http://www.processing.org/references/



CONCEPTUAL STRUCTURE: **CODE STRUCTURE**

Processing is designed to give visual feedback for learning code.

Basic geolocation and data visualization is only one of many uses for Processing. Before starting on those lessons, it's necessary to consider, first, how common parametric thinking is and, second, how to create basic forms.



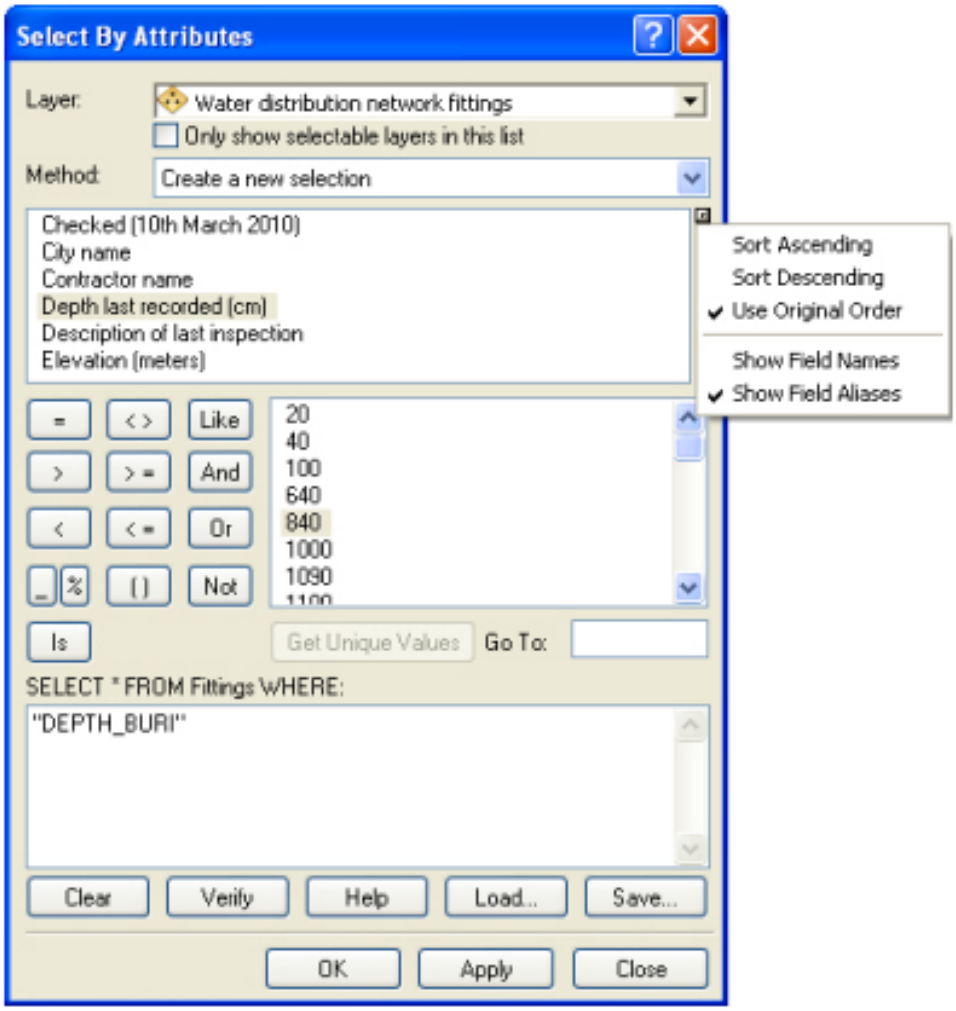
INPUT: **THE CODE BEHIND THE VISUALS**

From here on, coded sections will be shown as screenshots.

Explanations will by typed in this area below.

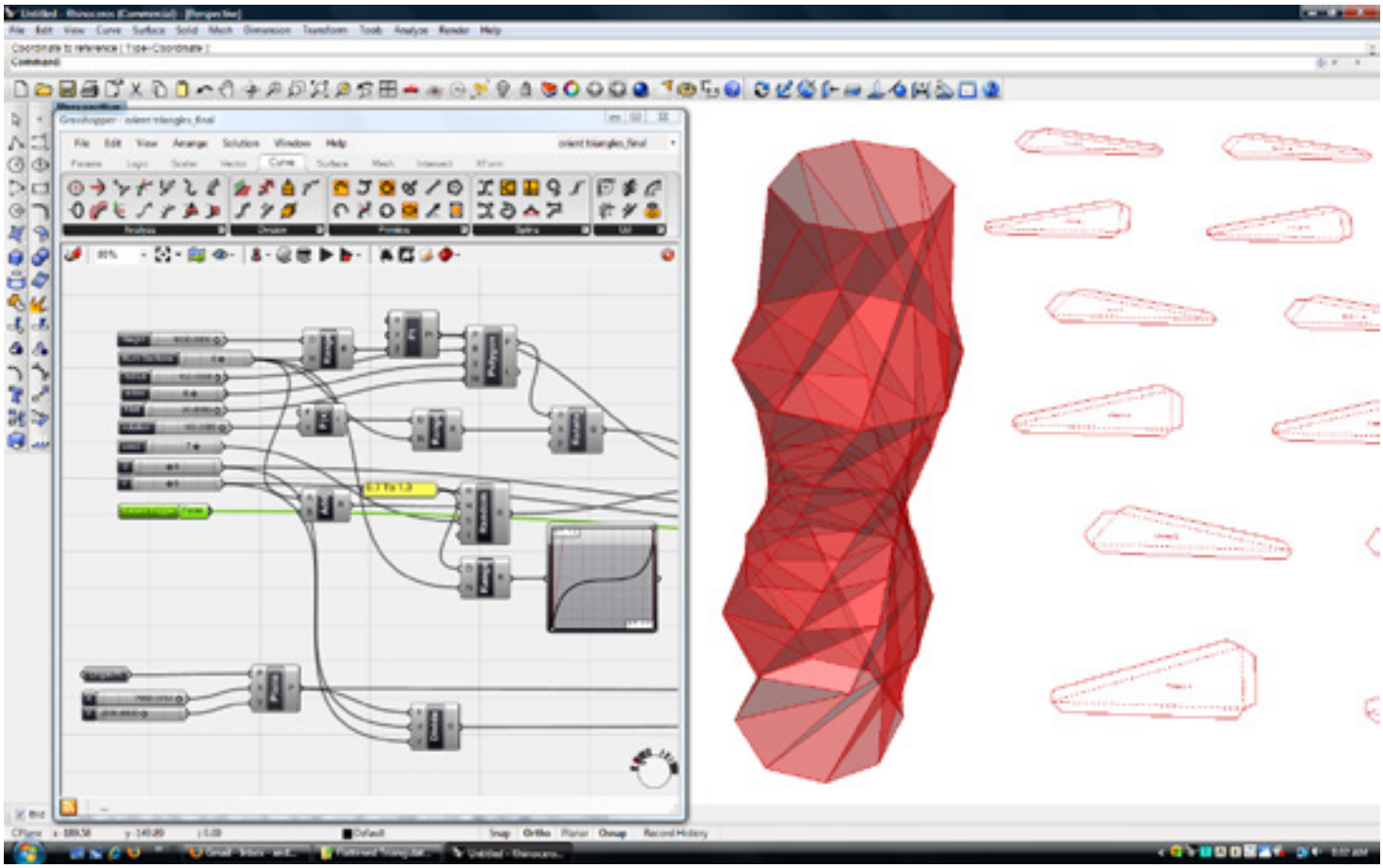
This has a few benefits, despite being a pain:

a) by re-typing code, you'll get use to being cautious with syntax,
b) by re-typing code, you'll get use to debugging your typos,
and
c) you'll get used to translating between psuedo-code (below) and code (above).



FIELD FILTERING **SECTIONS FROM TABLES**

Basic table sorting for pre-join/pre-analysis selections.



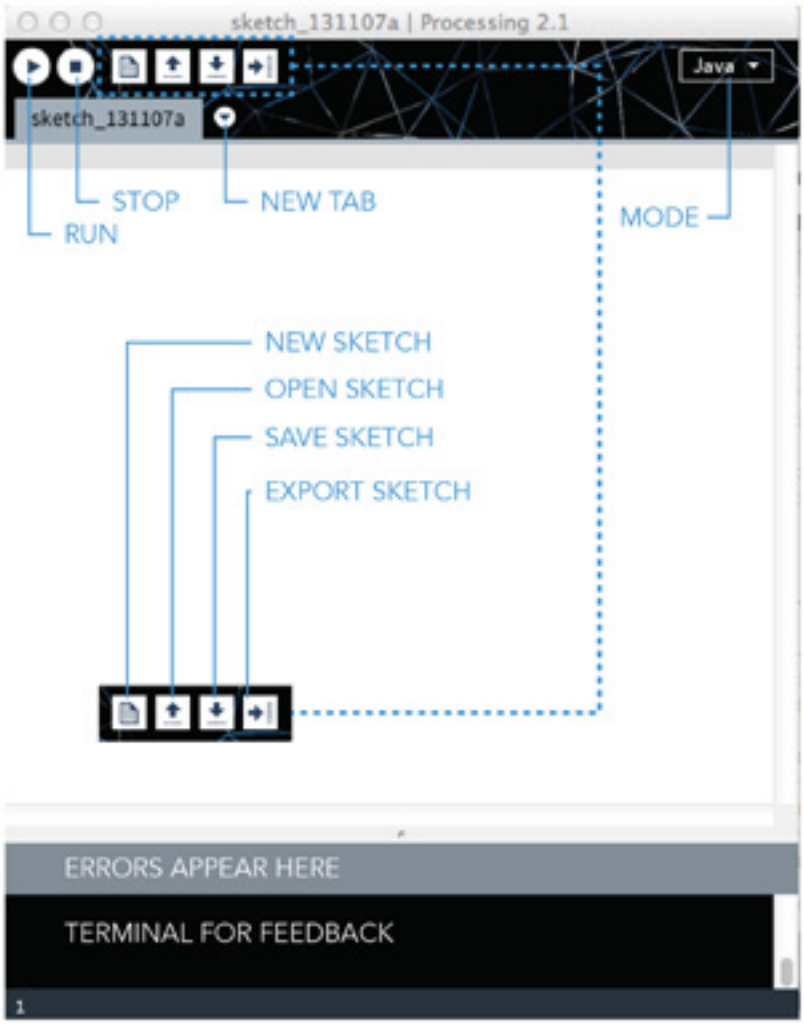
PARAMETER ADJUSTMENT **GRASSHOPPER OBJECTS**

Basic re-working of object parameters for formal manipulations.

WK 1

BASIC PROCESSING FOR PLOTTING

- interface
- shapes
- display
- loops



FILE MENUS: **USEFUL OPTIONS (EXPLORE)**

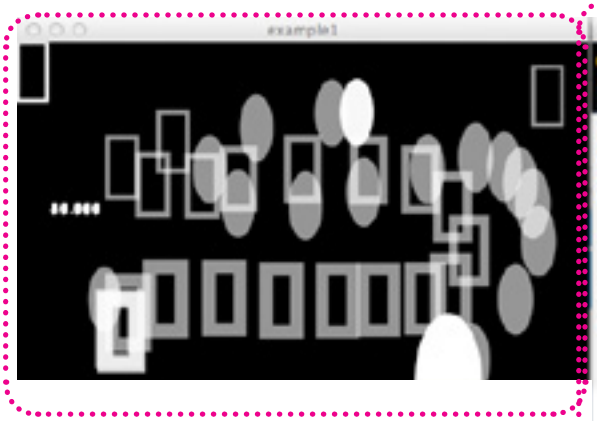
PROCESSING/PREFERENCES

FILE/SKETCHBOOK FILE/SAMPLES

EDIT/FORMAT EDIT/COMMENT

SKETCH/IMPORT LIBRARY

TOOLS/CREATE FONT TOOLS/COLOR SELECTOR



* APP OUTPUT

HIGHLIGHTS*

ORANGE-VARIABLE TYPES

BLUE- PREDEFINED FUNCTIONS

PINK- PREDEFINED SPATIAL/TEMPORAL PARAMETERS

ADJUST IN PREFERENCES

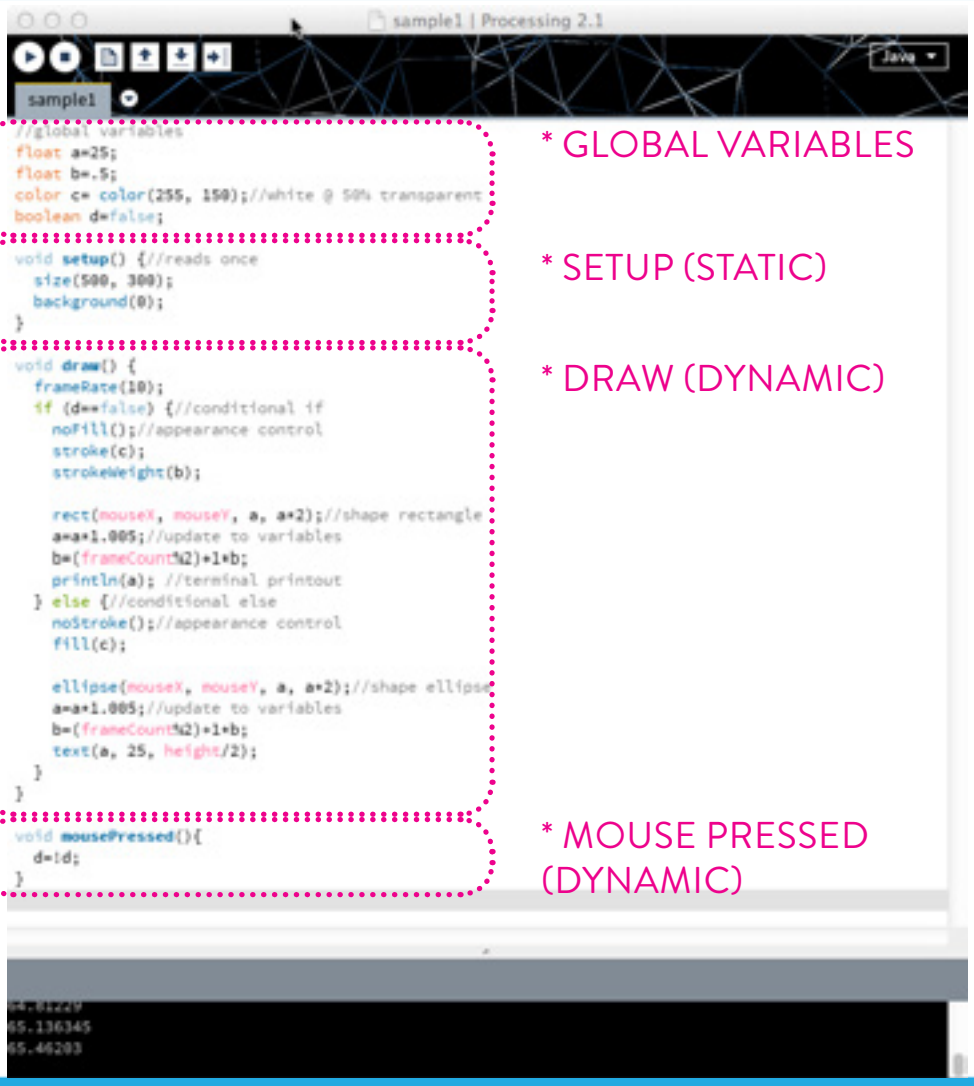
* PDE PROCESSING DEVELOPMENT ENVIRONMENT

SKETCH STRUCTURE **WORKING IN PDE**

You must 'run' the code to see the visual output in a second window.

Since you work on the PDE side, there are a number features that help make code legible:

a) color-coding of significant components (see above),
b) auto-indentation for visual ease of reading (command+T),
and
c) naming convention of lowerCase for multiword functions or variables.



SKETCH: EXAMPLE 1 **PARTS/ORDER**

Type the above. Test and run. A few notes on general structure:

Global variables are initialized (annonced) at the tops so that any function can read them. If boolean d was initialized within draw as a local variable, the function mousePressed would not be expecting it and would error out.

Functions are the code blocks beginning with void such as setup, draw, and mousePressed. They define a unit of work for the computer. They are contained within the widest { }. Within functions, code is read top to bottom.

Setup is static and will be read once, while draw and mousePressed are dynamic, meaning the computer will loop through them at 60 frame/second simultaneously (or a re-defined rate).

SKETCH: EXAMPLE 1 **SHAPES**

This simple sketch has two shapes drawn against a background.

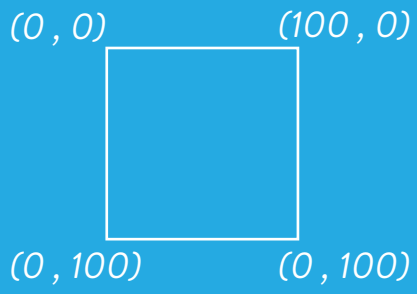
size (width, height); defines the window for the drawing. Background (0) defines its' color (black). These are static and do not refresh.

rect(starting pt x, starting pt y, width, height); draws a rectangle when the boolean d is false. Here the intial corners are defined as mouseX, mouseY which processing recognizes as dynamically defined by your mouse position.

ellipse(center pt x, center pt y, width, height); works similarily to draw an ellipse.

SKETCH: EXAMPLE 1 **PRIMITIVE SHAPES**

SPACE in processing: *(x coordinate , y coordinate)*



to define this rectangle, we'd write: rect (0 , 0 , 100, 100);

x increases from left to right
y increases from top to bottom

SHAPE types: *shape definition*

These will become the foundation for more complex graph-types and symbolization/display choices :

point (x, y); *point, size effected by strokeWeight() and stroke()*

line (x, y, x1, y1); *line*

rect (x, y, width, height); *rectangle, default upper left*

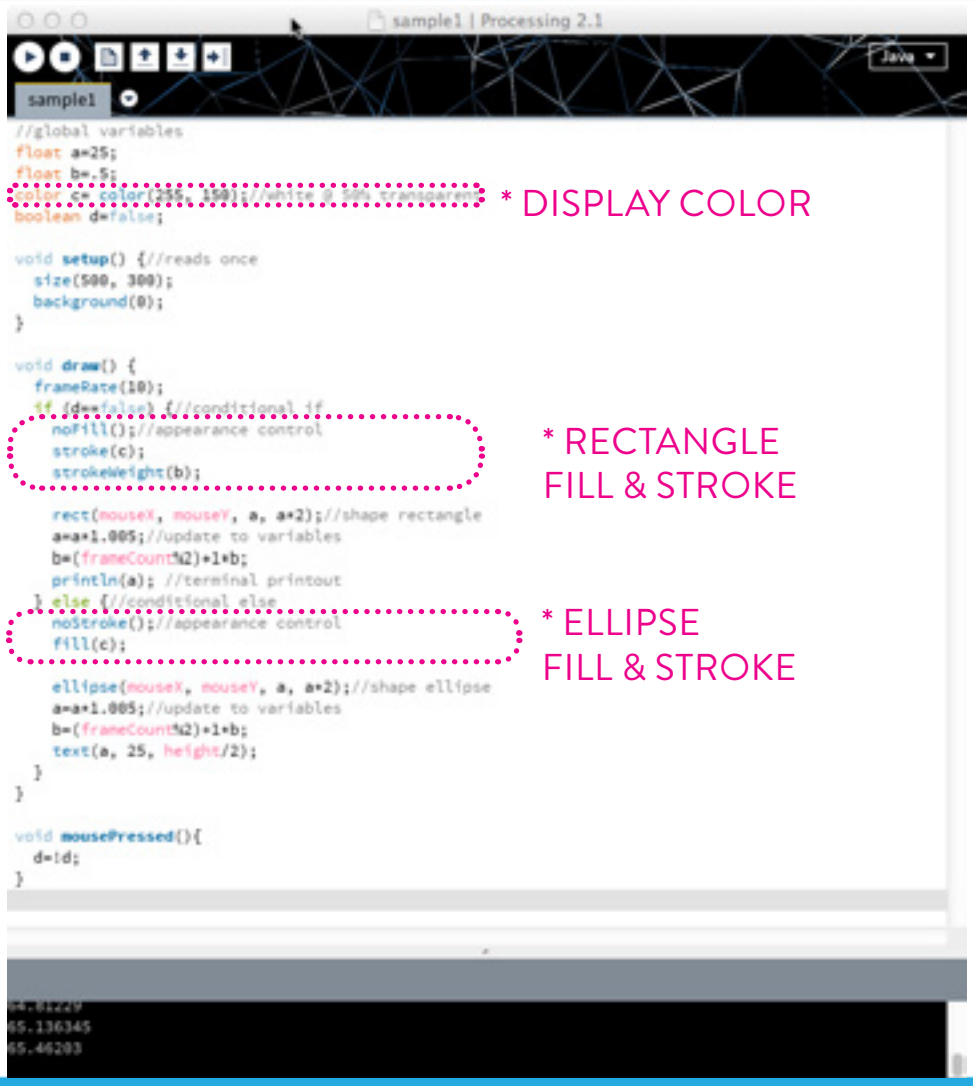
ellipse (x, y, diameter, diameter); *ellipse, default centered*

triangle (x, y, x1, y1, x2, y2); *triangle defined by 3 points*

quad (x, y, x1, y1, x2, y2, x3, y3); *quadrilateral defined by 4 points*

arc (x, y, diameter, diameter, start radians, stop radians, mode); *an arc defined by center, diameter, start & stop angles, and edge modes*

find @ http://www.processing.org/references/



SKETCH: EXAMPLE 1 **DISPLAY VARIABLES**

This simple sketch defines color globally and then uses that color locally in combination with fill and stroke. Each shape display parameter is defined in advance of the shape itself.

noFill( ); or noStroke( ); in either case, you eliminate fill (internal color) or stroke (outline). If these are not set processing defaults to a white fill and a 1pt black stroke outline.

stroke(color); sets the color of an outline stroke.
strokeWeight(width in pixels); sets the width of an outline stroke, defaults to the center of line.
fill (color); sets the color of an internal fill.

SKETCH: EXAMPLE 1 **DYNAMIC UPDATES**

As the sketch runs within each block, the variables a, b, and d are set to update through simple math or, for d, in reaction to user mouse-pressing. When a variable is updated, it is then used the next time that block of code runs. Here, those changes accumulate is strokeWeight and figure size, but you could also reset variables back to their original value for a stable drawing.

Follow the arrows above to understand those interactions. Note the use of d = ! d, which turns the boolean from true to false or false to true, acting as a button to shift between the two conditional options in if/else.

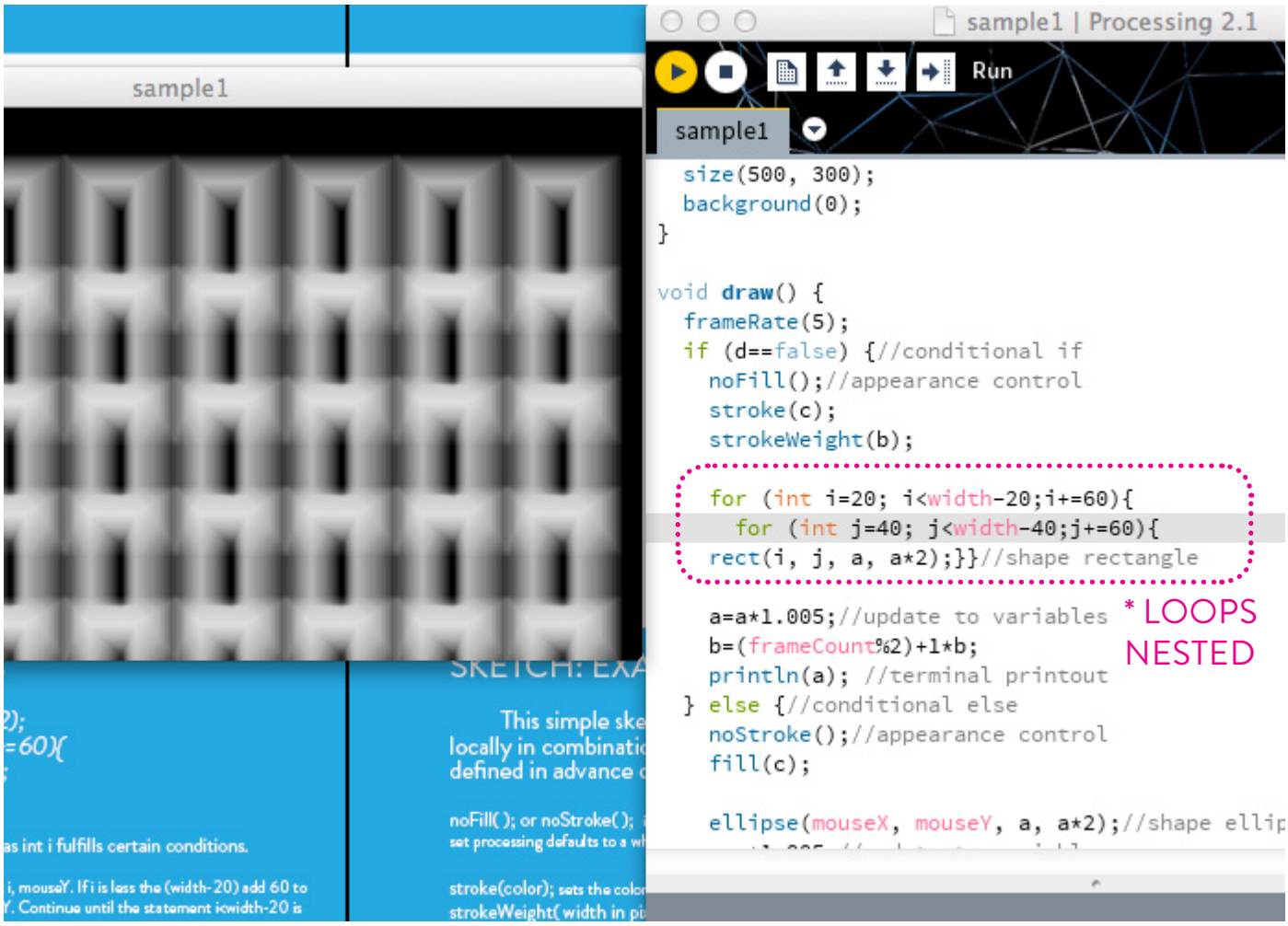
SKETCH: EXAMPLE 1B **LOOPS**

change to

```
rect(mouseX,mouseY, a, a*2);
for (int i=20; i<width-20;i+=60){
        rect(i, mouseY, a, a*2);
}
```

This 'for loop' tells the program to repeat a task as long as int i fulfills certain conditions.

Here, it says, 'Given that i=20, draw a rectangle with an origin at i, mouseY. If i is less the (width-20) add 60 to make i=80 and draw another rectangle with an origin at i, mouseY. Continue until the statement i<width-20 is false. Thus it draws a row of rectangles offset by 60.



SKETCH: EXAMPLE 1B **LOOPS NESTED**

Here the loops defined according to i and j are nested.

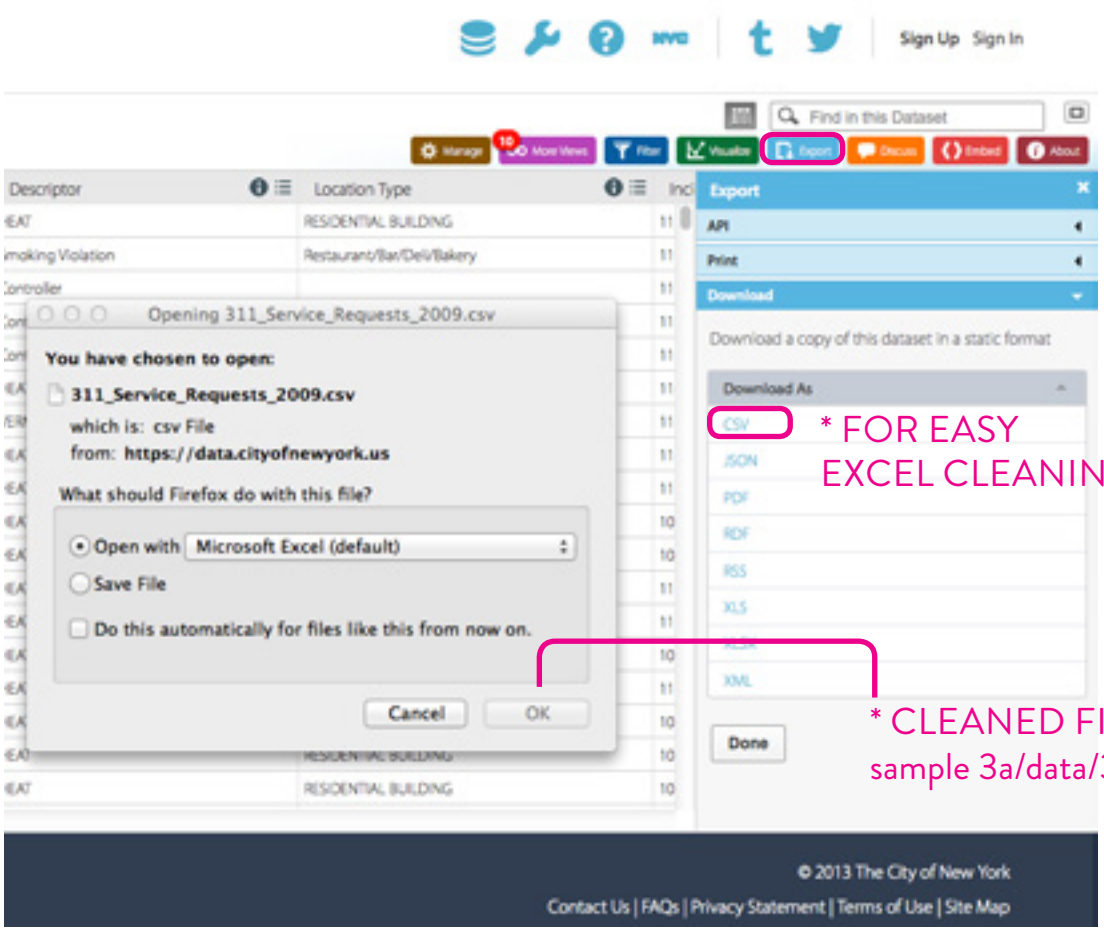
The grid of rectangles that results draws a series of columns of rectangle across the page. For i=20, it executes all at j positions and then moves to the next i variable, i=80... and so on.

For tables, we will use loops and conditionals to find matching values (description = dog barking) and then extract the lat and long information in those rows, looping through all rows in the process.

WK 1

RECONSTRUCTING 311 EXAMPLE

- NYC Open Data: CVS download
- reading tables
- geoplotting data
- symbology
- internal filtering w/ conditionals
- parsing/creating new files of filtered data



SKETCH: EXAMPLE 3A **ORIGINAL & CLEANED SOURCES**

https://data.cityofnewyork.us/Social-Services/311-Service-Requests-2009/3rfa-3xsf

To replicate the initial 311 example, we'll begin with the dataset above.

Downloading is as easy as selecting export/download/csv to get a file for simple cleaning.

I've cleaned a file, merely deleting extra columns that we won't need for representation. It can be found in the data-folder of file "sample 3a/data/311_2009-Noise.csv"

* PShape- initializes use of vector files
* Table- initializes use of csv files

* loadShape( ); - then loads svg files
* loadTable- then loads csv files,
  "header" notes column titles
* .getRowCount( );
  then gives the count of those rows for later use

SKETCH: EXAMPLE 3A **USING/LOADING FILES**

To use outside files, they must be declared globally, then loaded in setup, and are then ready for use/reading in void draw( ).

Code the above. All the files to be loaded must be located in the sketche's data folder. To place files there, they can be 'dragged-and-dropped' on the sketch pde window.

Look up the 'shape' and 'table' commands in reference. For shape, the vector image is declared, as well as parameters identical to rectangles: shape(img, x, y, width, height).

*loop through table rows to get values

* .getFloat (row, column by header)

SKETCH: EXAMPLE 3A **TABLE READING**

The basis of all table use is reading through the rows iteratively, pulling out values by their column position and/or matching values.

In this sample, after calling the shape in void draw(), we want to iterate through the rows to grab the longitude and latitude values. To do this we set up new variables float x and float y. They are then defined as specific row, column positions (if iterative re-definition). This is done with the table.getFloat ( ) function (see above).

ADJUST FROM GEOGRAPHIC
TO PIXEL SPACE OF THE SKETCH

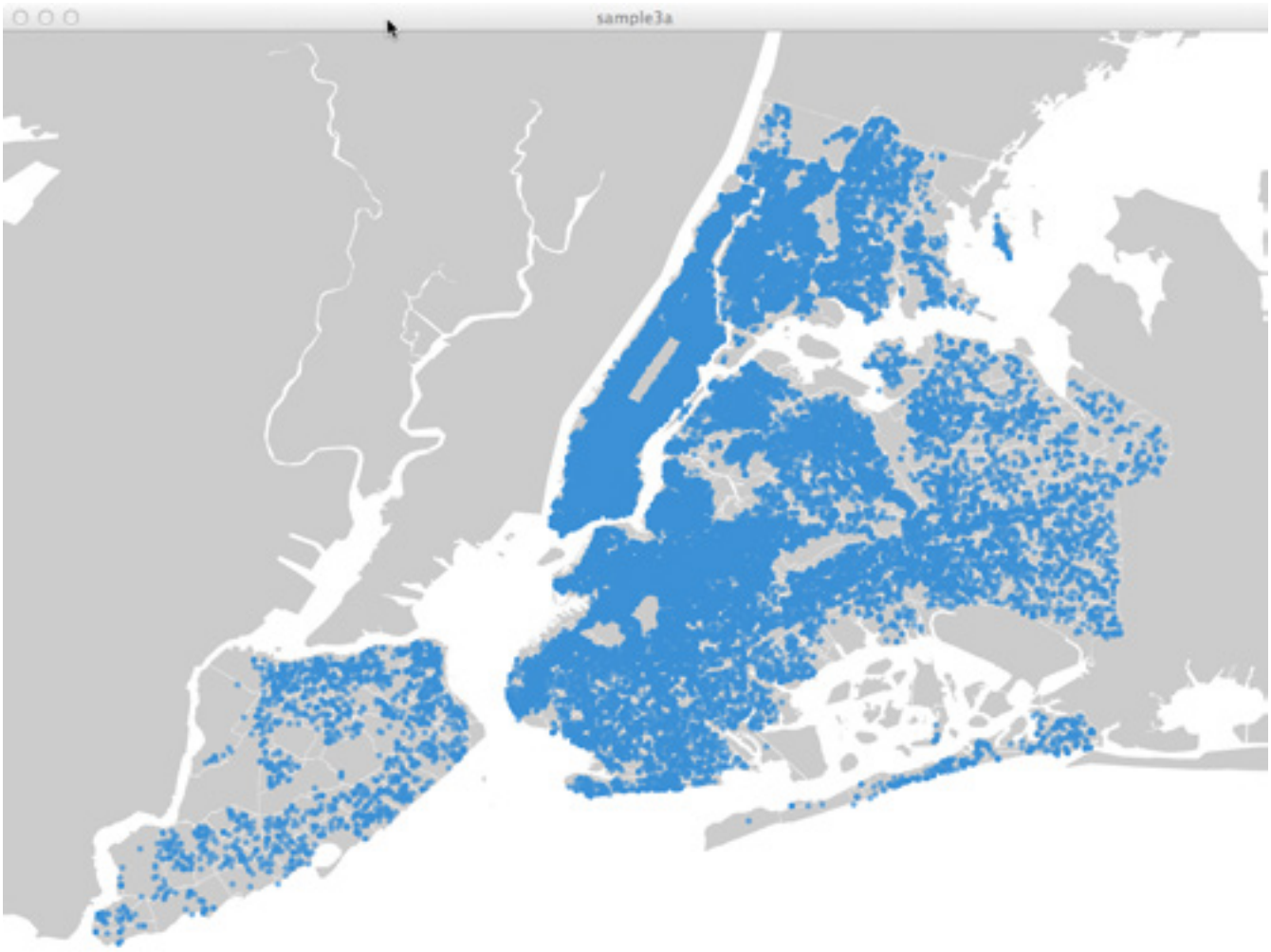
* map (variable, x, y, new x, new y)

* final location of each point
point (lng, lat) ;

SKETCH: EXAMPLE 3A **TABLE READING**

Processing works with a basic sq pixel grid, thus it is easiest to convert latitude and longitude into rectangular projection maps. (Plate Carrée)

As noted above in the comments, the underlay vector map stretches from -74.3 to -73.65 and 40.95 to 40.45. The map function is then used to re-map the original float x and float y to float lng and float lat, shifting from geographic coordinates to the pixel range of the image, as defined in setup.



SKETCH: EXAMPLE 3A **POINTS MAPPED**

The basic result is shown above.

Here are all the Noise complaint points from 311 in 2009, regardless of borough or further complaint type.

TABLES: **MORE ON COMMON READING MECHANISMS**

TABLE FUNCTIONS: *more basics for use with loadTable(), saveTable()*

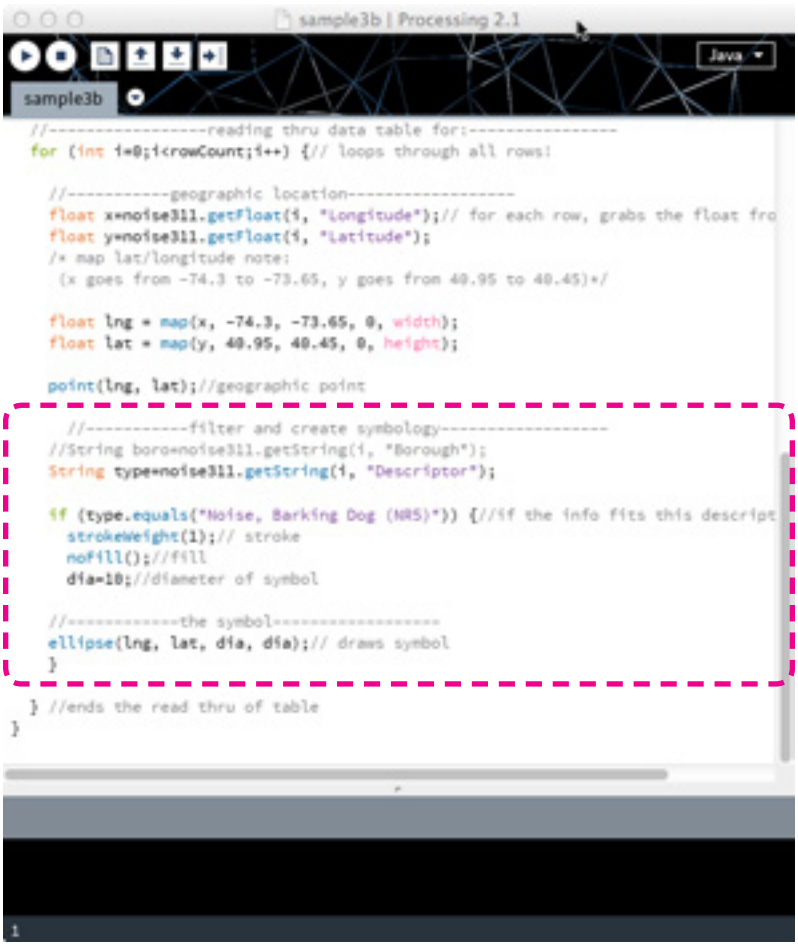
addColumn() Adds a new column to a table
removeColumn() Removes a column from a table
getColumnCount() Gets the number of columns in a table
getRowCount() Gets the number of rows in a table
clearRows() Removes all rows from a table
addRow() Adds a row to a table
removeRow() Removes a row from a table
getRow() Gets a row from a table

rows() Gets multiple rows from a table
getInt() Get an integer value from the specified row and column
setInt() Store an integer value in the specified row and column
same for getFloat(), getString() etc.

findRow() Finds a row that contains the given value
findRows() Finds multiple rows that contain the given value
matchRow() Finds a row that matches the given expression
matchRows() Finds multiple rows that match the given expression

removeTokens() Removes characters from the table
trim() Trims whitespace from values

find @ http://www.processing.org/references/



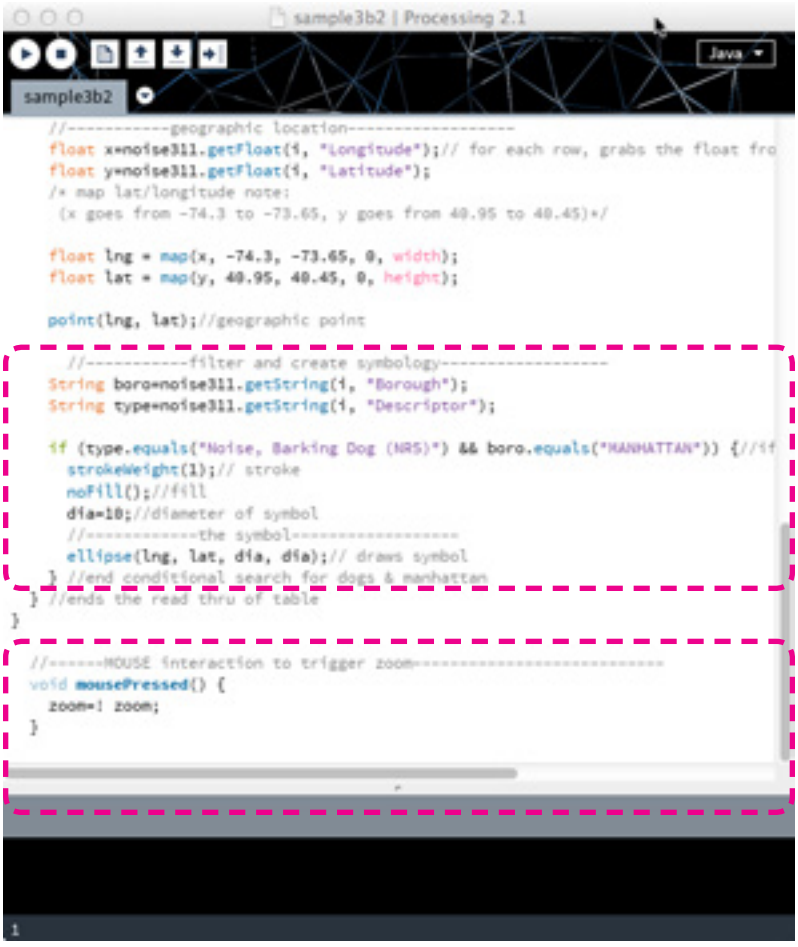
* .getString(row, column by header)
works like .getFloat( )

*variable.equals("selected phrase")
is used to match/filter words

SKETCH: EXAMPLE 3B2 **SYMBOLIZE & FILTER**

Within the same loop, set up a variable to grab to string in the "Description" column. This can then be used to filter for specific matches with a desired term, "Noise, Barking Dog (NR5)."

By setting up the ellipse to only show under these conditions, we will still have all 311 dots, with larger dog circles.



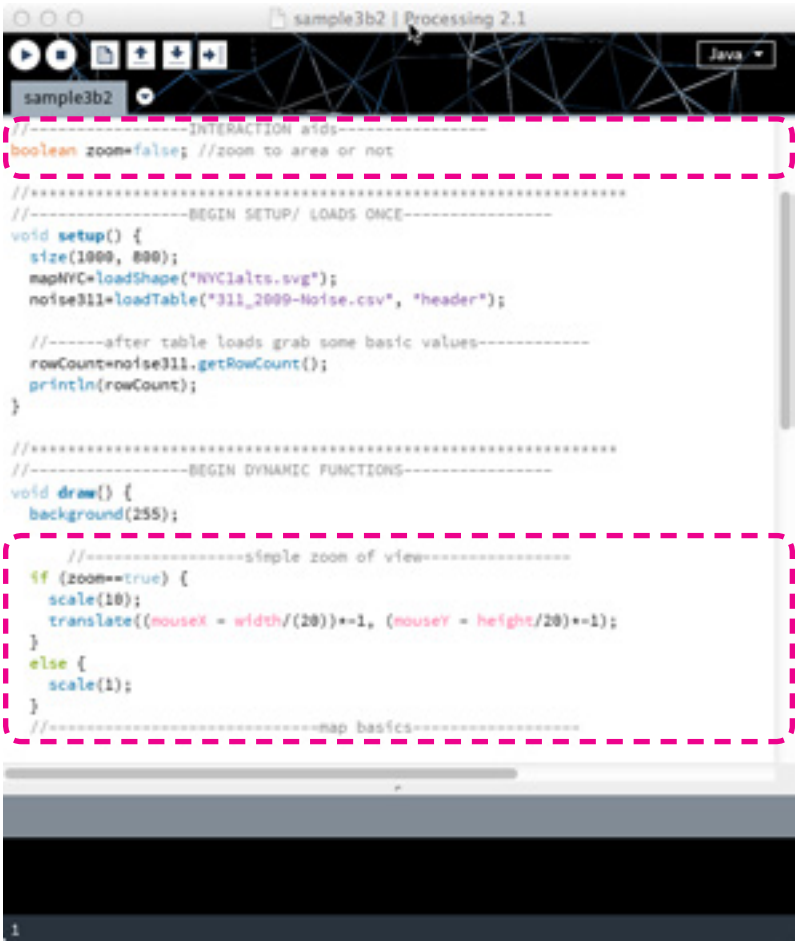
Create another variable boro and add to type.equals with a logical && to filter for “MANHATTAN” as well as Dog Barking

Add the function void mousePressed() in order to create a way to zoom

SKETCH: EXAMPLE 3B2 **FILTER MORE & INTERACT**

Additional variables can be added to further filter and refine which data is symbolized.

Simple interactions can be added to maneuver through the graphic.



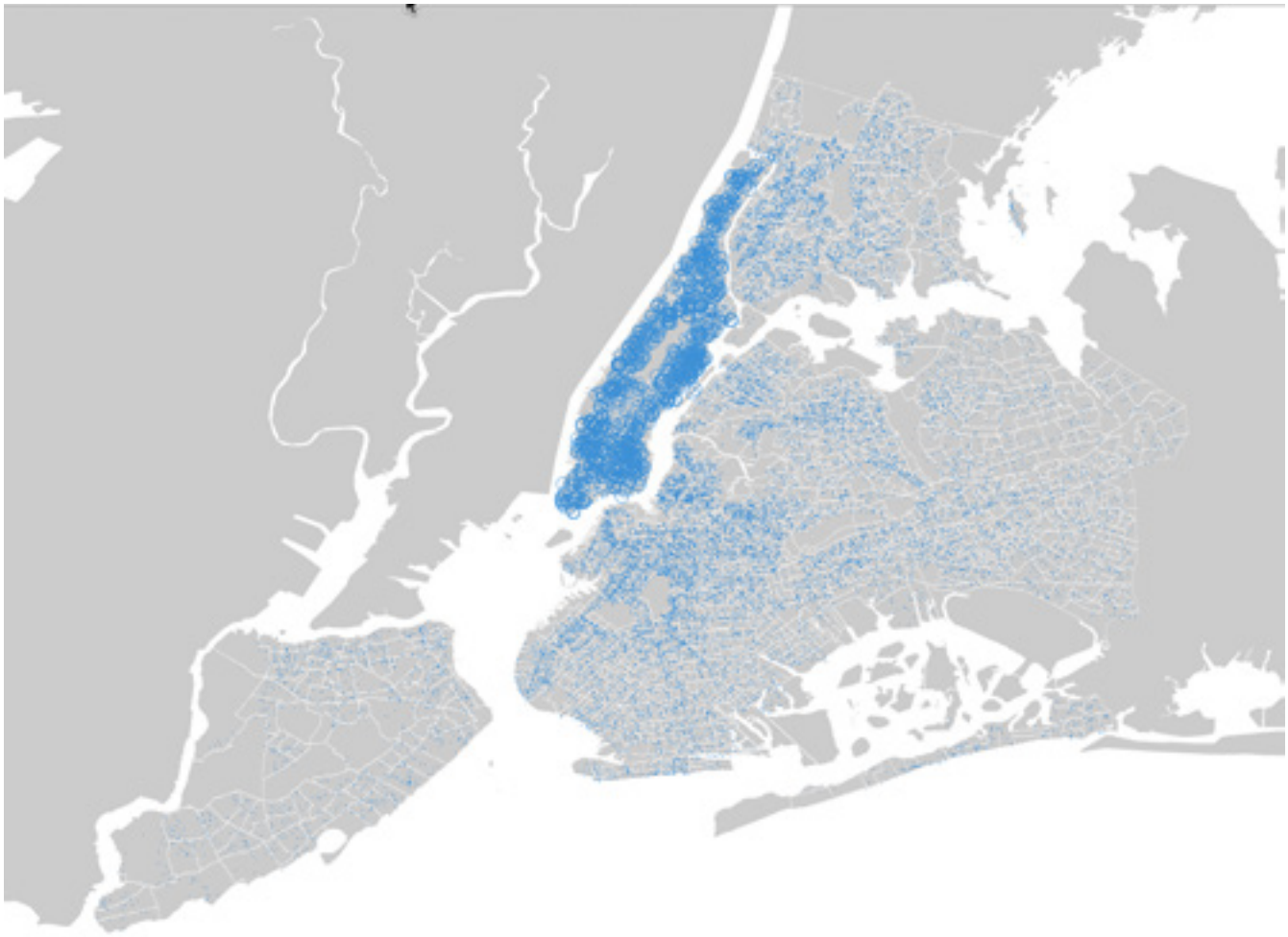
Add “Boolean zoom=false;” up at the top

With the upper are of void draw( ), this if/else statement check to see if mouse is pressed (zoom==true).

The function scale( ); multiples while translate(x,y); sets up a new origin x,y position for the full sketch.

SKETCH: EXAMPLE 3B2 **INTERACT**

Add the Boolean variable to the global declaration area and the if/else statement above shape( ) in the void draw( ) function.



SKETCH: EXAMPLE 3B2 **UNZOOMED**

This code generates the above map when the mouse is not pressed.

SKETCH: EXAMPLE 3B2 **ZOOMED**

This code generates the above map when the mouse is pressed.

Declare new tables in global variables, call them in void setup( )

*TableRow row: table.findRows(matching phrase, column);

*TableRow newRow= table.addRow(row);

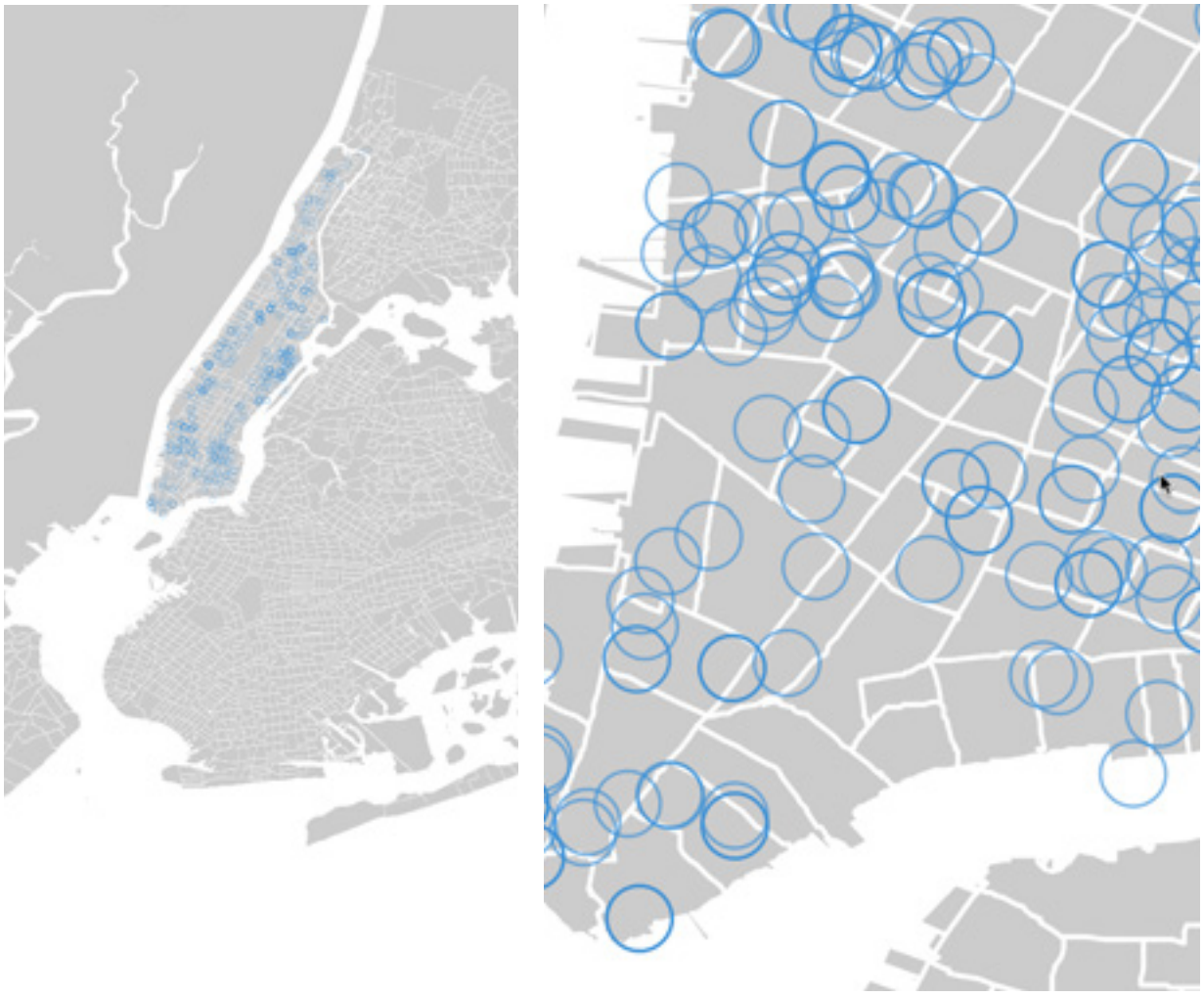
*saveTable(table, “filename”);

SKETCH: EXAMPLE 3B3 **CREATING SORTED TABLES**

Another way to sort data is to filter in setup and create new tables.

As in using existing files, new tables must be declared and then activated through the function new Table( );. Here, we are filtering the data using a ‘for’ TableRow functions instead of iterating through each row. The function .findRows will grab the data from all matching rows and then add to our new tables. After each of these .findRow loops, the new table with its sorted rows is saved into a back-up csv file.

manhattanTable will then replace all older noise311 tables. Instead of column name “Descriptor,” the column number should be used because manhattan table has no headers.



SKETCH: EXAMPLE 3B3 **RESULTS**

Note, by linking only to the new file, filtered to hold only Manhattan Dog Barking complaints, we no longer have the residual dots of all 311 Noise complaints.

And that is it, for the first week. . .

WK 2-MATH, INTERACTIONS, & MULTIPLE DISPLAY FORMS

- shifting from 311 dog calls to other NYC Open Data

- simple math from tables
  - StringLists
  - InterDicts
  - Arrays

- legible & interactive information
  - labels and font use
  - mouse and keyboard functions
  - printing and export options for jpg, png, and pdf

- more advanced information & symbology
  - finding files: NYC Open data for Boilers and Oil Use (CSV tables)
  - quantitative thresholds and symbols
  - scaled keys
  - more time-based variables with basic addition/interaction
  - secondary filters- typologies and oil use by zipcode
  - representational forms- data roses
  - combining modulo and raw BTU quantities for final data
  - visually cleaning your geographic 'dashboard'

WK 3-LIVE DATA, JSON & GEOJSONs, CREATIVE COPYING

- Moving toward correllation and arguments with data

- Introduce API and live webdata, json format
  - NYC Tree Count-retrieving data in json form to pair with Boiler emissions
  - Other Socrates examples- simple trash collection numbers as bar graphs
  - Other common sources in json (weather, flickr, data.gov, etc. etc.)
  - Temporal limits and paywalls- workarounds for data retrevial
  - example: historical wind data from weather underground

- Additional API sources and other ways to engage json
  - public resources (socrates, nyc open data, data.gov)
  - popular applications (flickr, twitter, yahoo api queries, etc.)
  - easy json and geojson uses (to build off processing)- leaflet, mapbox, d3
  - processing for csv and table conversion to json and geojson

- Copying code, understanding structures
  - open source community and standards
  - resoures: github, stackoverflow, others
  - citation practices, code credits
  - example: Ben Fry's network graphs + NYC trash disposal trajectory json