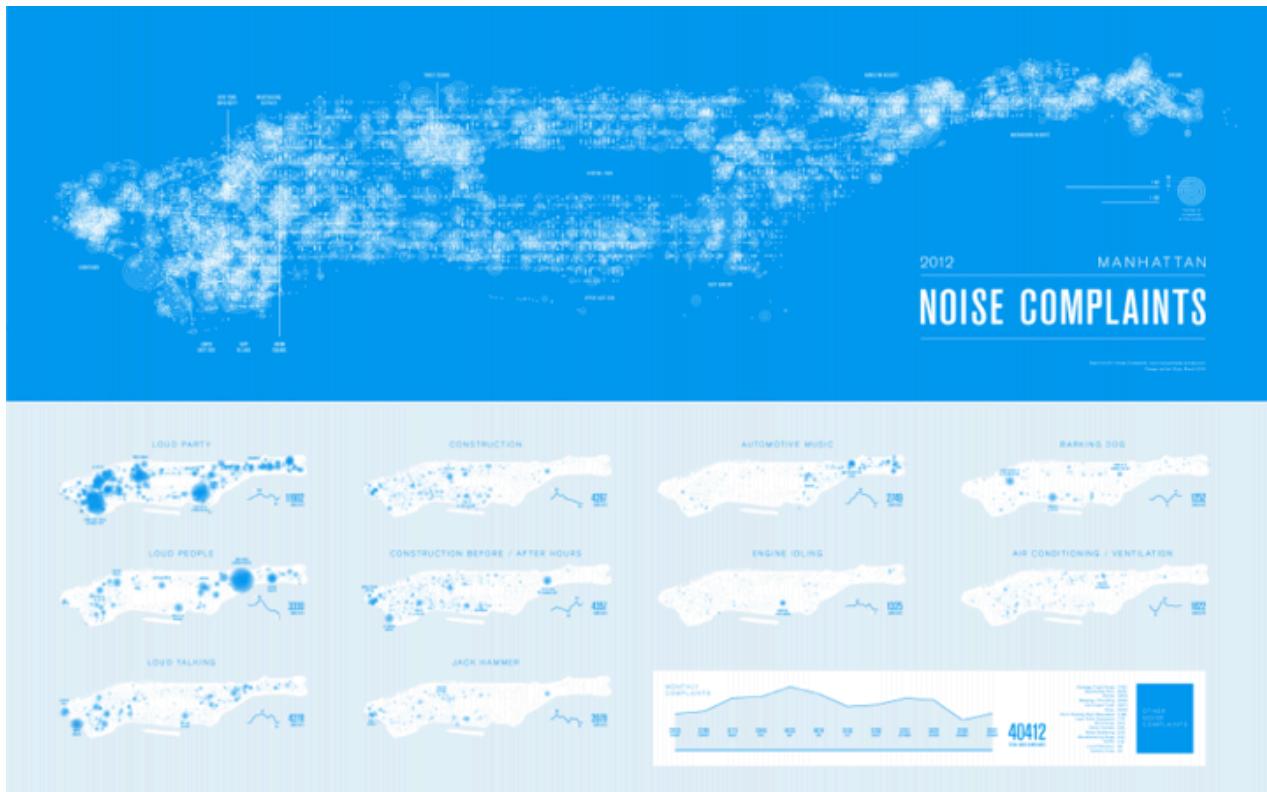


- 3 WORKSHOPS- 5hrs/each
- CURRICULM BY: Meg Studer



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 these 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 free-ware 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, &amp; 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 &amp; GEOJSONs, CREATIVE COPYING

- Moving toward correlation 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 retrieval
  - 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
  - resources: 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/maneyes/> 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/>



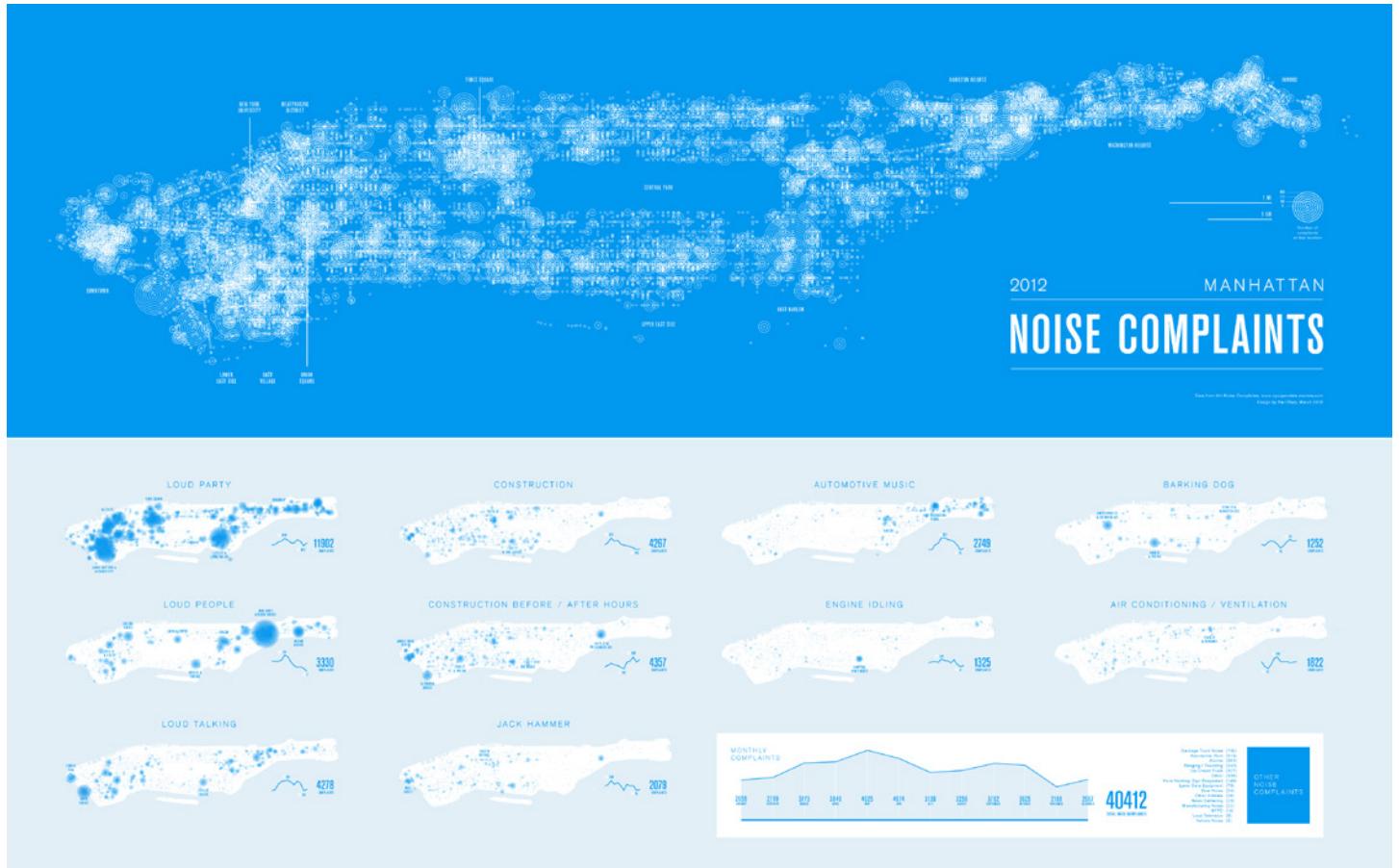
## 9 WK 1-BASIC PLOTTING

59 WK 2-MATH, MULTIPLE FORMS, INTERACTION

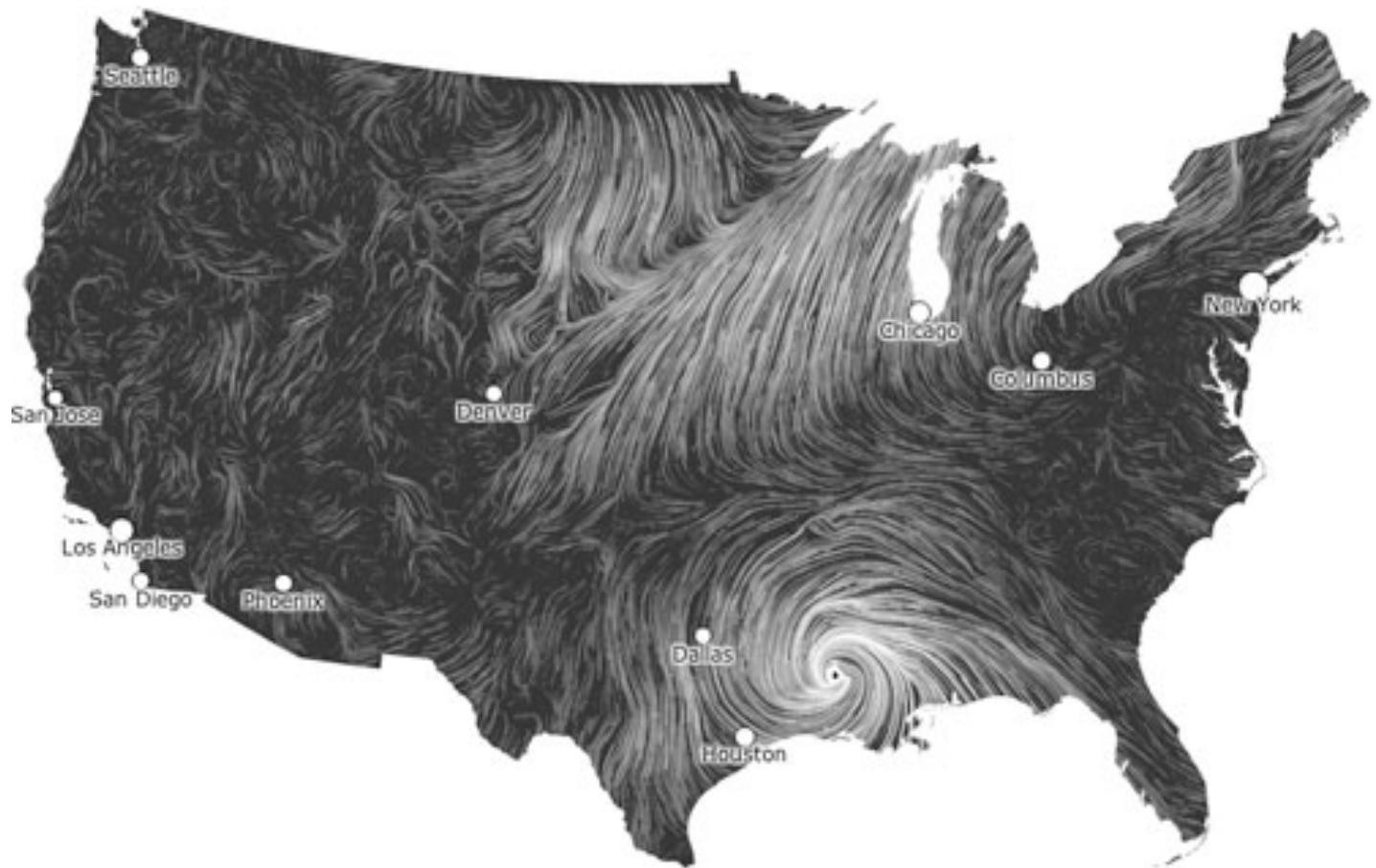
96 WK 3-LIVE DATA, JSON & GEOJSONs, CREATIVE COPYING

## 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

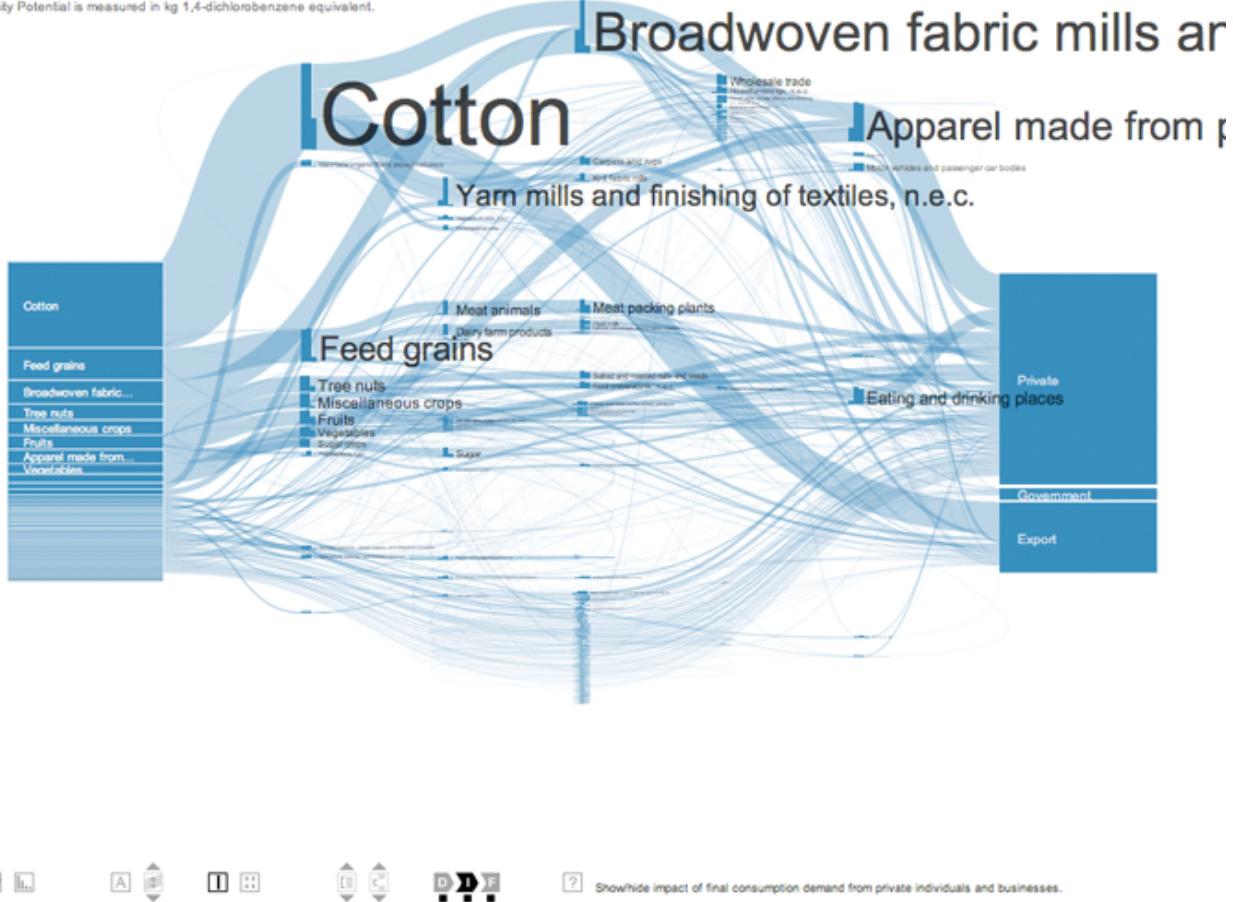


Noise Complaints, 2012 by Karl Sluis  
311 data from Open Data NYC (<http://www.visualizing.org>)



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

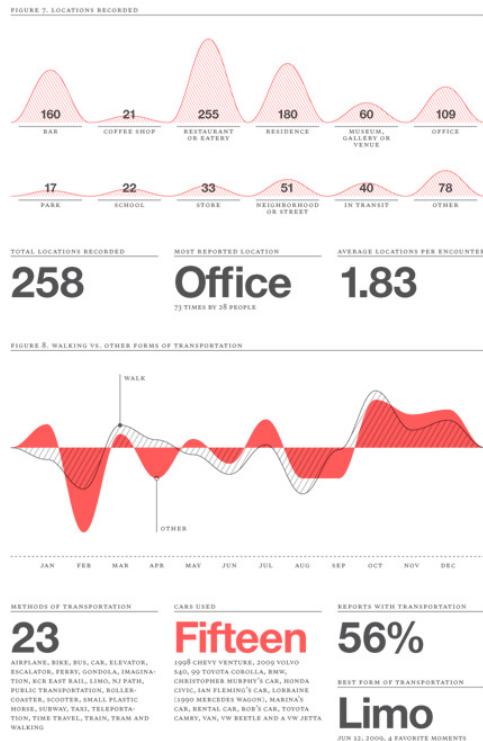
**Freshwater Aquatic Ecotoxicity Potential**  
refers to the impact of toxic substances emitted to freshwater aquatic ecosystems. Freshwater Aquatic Ecotoxicity Potential is measured in kg 1,4-dichlorobenzene equivalent.



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

## Where

Location and methods of transportation.



QUESTION 5. WHERE WERE YOU AND NICHOLAS?

*Shebeen (surprise).*  
MARTIN P, JANUARY 28

*Foocamp at MSFT offices (NERD) in Cambridge, Ma.*  
MATT P, MARCH 28

*AP and SK's roof.*  
RICHARD, JULY 4

*Red Hook pool.*  
JESSICA R, JULY 25

*Ace Bar for Gigi's birthday.*  
REBECCA, NOVEMBER 20

QUESTION 6. DID YOU GO ANYWHERE ELSE TOGETHER?

*Into the conference room.*  
MARIE-CLAIRE, APRIL 1

*To Momofuku-Ko.*  
DANIELLYS, JUNE 8

*Walked out to pick up lunch.*  
MATTHEW G, APRIL 28

*The beach.*  
MICHAEL N, JULY 11

*Olga's for champagne to celebrate, someone's house for the dance party.*  
RYAN, DECEMBER 11

QUESTION 7. WAS ANY FORM OF TRANSPORTATION USED?

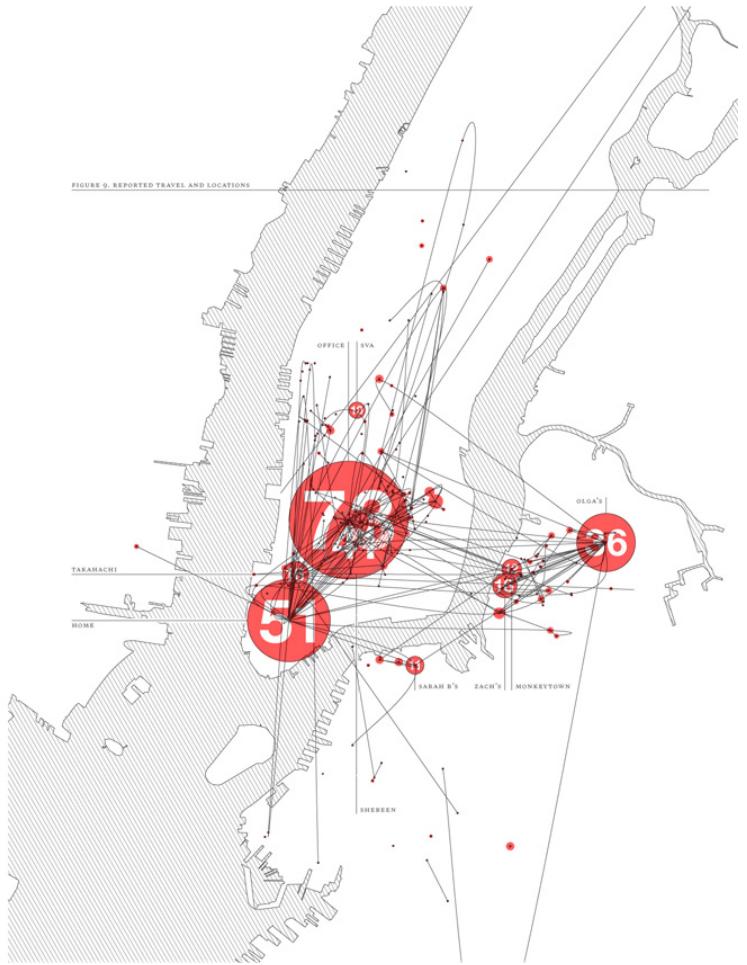
*I believe Nick drove.*  
LEWIS, MAY 17

*Limo. Best limo ride ever.*  
MARIANA, JUN 12

*Yes, type unknown.*  
ZACH, AUG 19

*Peds, JMZ, 4/5/6 trains.*  
OLGA, OCT 17

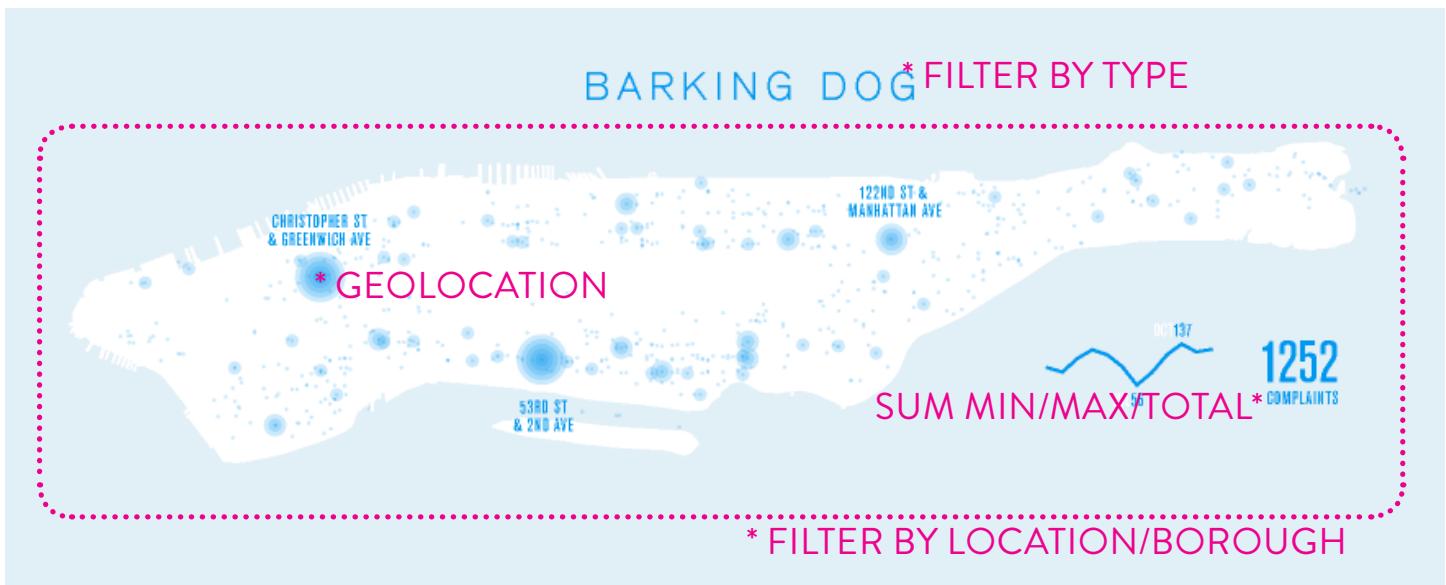
*Airplane.*  
KACIE, OCT 25



# WK 1

## COMPUTATION STRUCTURE (IS FAMILIAR!)

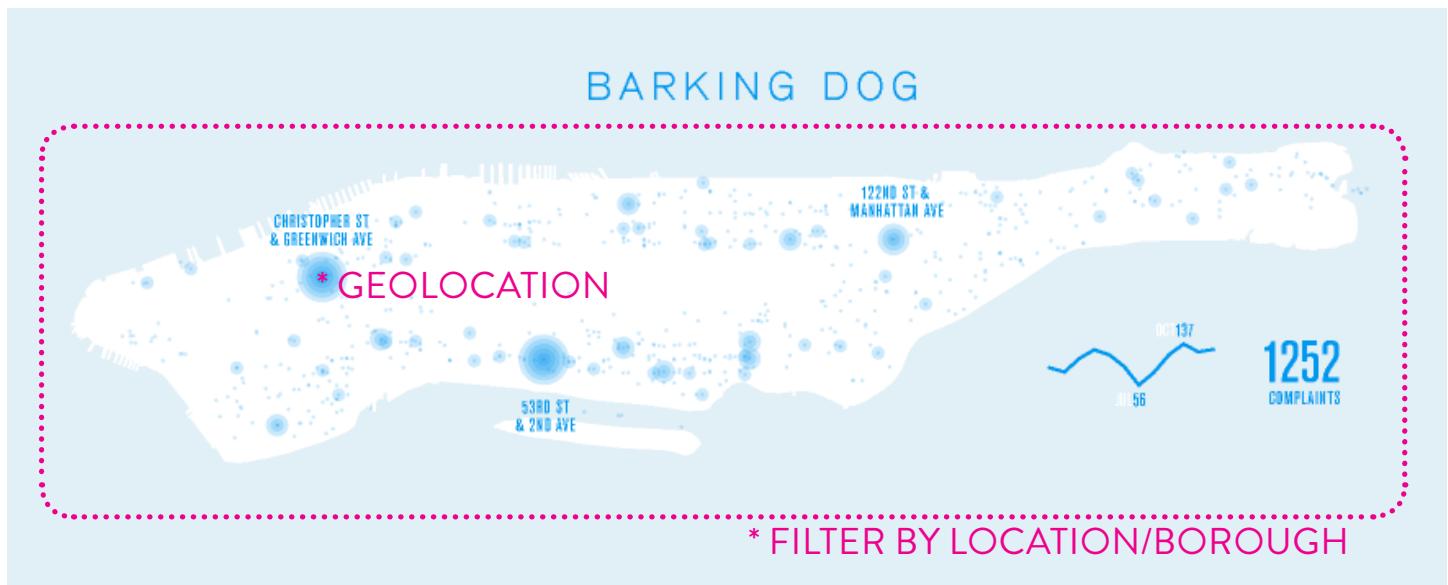
- references/sources
- variables
- conditionals
- operators
- syntax
- comparative structures



## 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>

**NYC OpenData** \* DATE OF COMPLAINT?

Based on 311 Service Requests from 2010 to Present  
All 311 Service Requests from 2010 to present. This information is automatically updated daily.

1100+ Datasets Available

\* DOES DESCRIPTOR READ “BARKING DOG”?

Manage 593 More Views

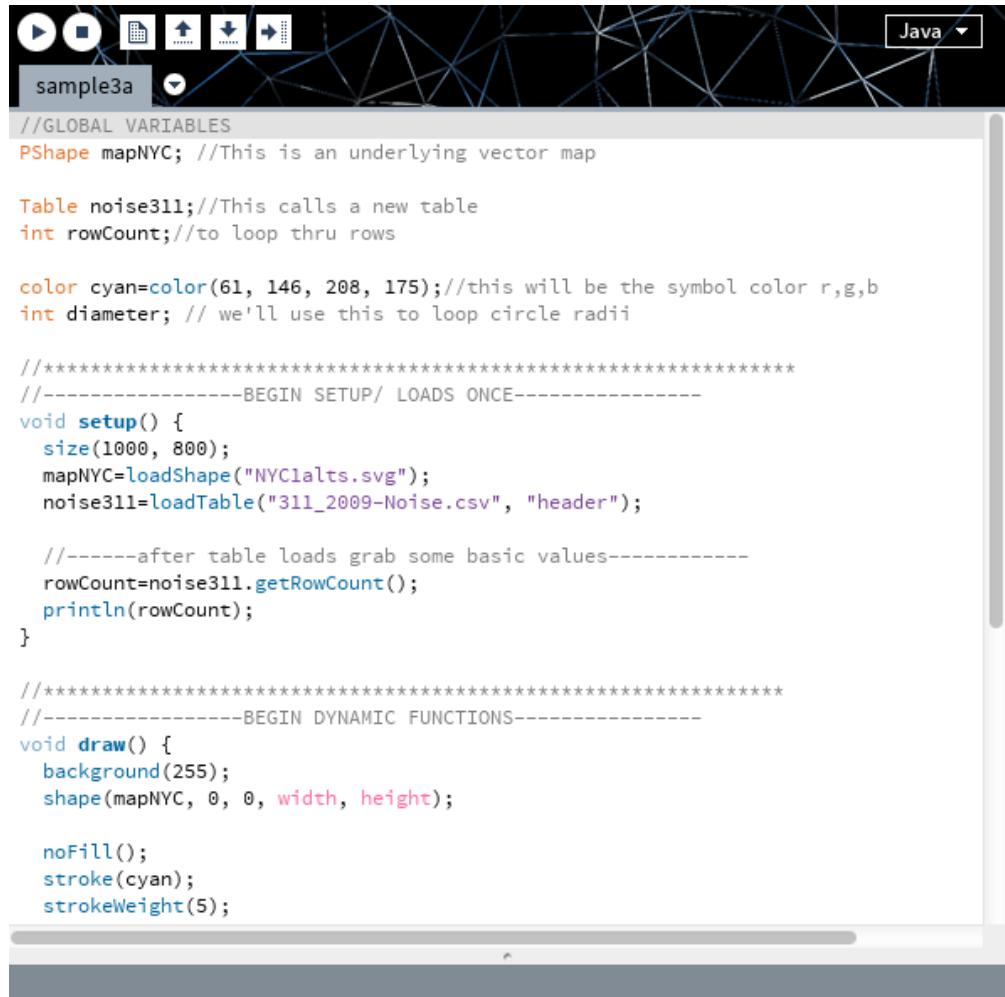
que Key	Created Da	Closed Date	Agency	Agency Na	Complaint Type	Descriptor	Location Type
1	528705	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Talking	Store/Commercial
2	527775	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
3	530510	11/06/2013	NYPD	New York C	Noise - Vehicle	Car/Truck Horn	Street/Sidewalk
4	529483	11/06/2013	NYPD	New York C	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk
5	529423	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
6	532058	11/06/2013	NYPD	New York C	Blocked Driveway	Partial Access	Street/Sidewalk
7	529903	11/06/2013	NYPD	New York C	Blocked Driveway	Partial Access	Street/Sidewalk
8	530300	11/06/2013	NYPD	New York C	Noise - Vehicle	Engine Idling	Street/Sidewalk
9	526995	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
10	526538	11/06/2013	NYPD	New York C	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk
11	528221	11/06/2013	NYPD	New York C	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk
12	527915	11/06/2013	NYPD	New York C	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk
13	528741	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
14	528642	11/06/2013	DOT	Department	Highway Sign - Missing	Exit/Route	Highway
15	530302	11/06/2013	NYPD	New York C	Noise - Commercial	Loud Music/Party	Store/Commercial
16	528715	11/06/2013	NYPD	New York C	Blocked Driveway	No Access	Street/Sidewalk
17	526936	11/06/2013	DPR	Department	Overgrown Tree/Branches	Traffic Sign or Signal Blocked	Street
18	530450	11/06/2013	DOHMH	Department	Indoor Air Quality	Other (Explain Below)	3+ Family Apartment Building

Contact

## 311 EXCERPT: GEOLOCATION &amp; FILTERING

## NOTE COLUMNS FOR FILTERING:

- description is directly tagged “barking dog”
- latitude is given, longitude is given
- borough is given as “manhattan”
- date of call registered under “complaint opened”



```
//GLOBAL VARIABLES
PShape mapNYC; //This is an underlying vector map

Table noise311;//This calls a new table
int rowCount;//to loop thru rows

color cyan=color(61, 146, 208, 175);//this will be the symbol color r,g,b
int diameter; // we'll use this to loop circle radii

//*****-----BEGIN SETUP/ LOADS ONCE-----//
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYC1alts.svg");
  noise311=loadTable("311_2009-Noise.csv", "header");

  //----after table loads grab some basic values-----
  rowCount=noise311.getRowCount();
  println(rowCount);
}

//*****-----BEGIN DYNAMIC FUNCTIONS-----//
void draw() {
  background(255);
  shape(mapNYC, 0, 0, width, height);

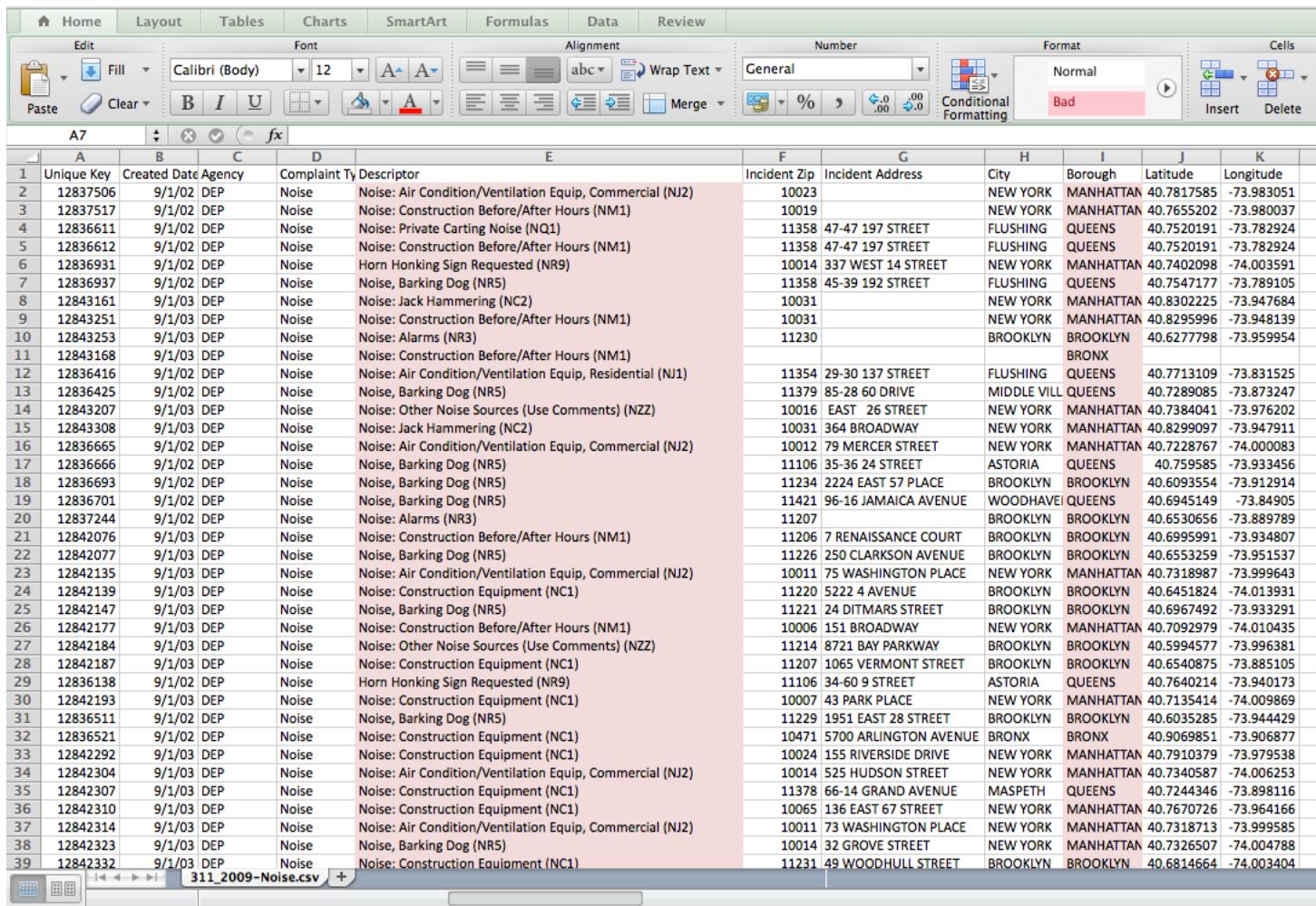
  noFill();
  stroke(cyan);
  strokeWeight(5);
```

## 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 Type	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 (NC1)	11358	FLUSHING	QUEENS	40.7520191	-73.782924	
12836612	9/1/02	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11358	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 (NRS)	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	10031	BROOKLYN	BROOKLYN	40.6277798	-73.959954
12843168	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11354	29-30 137 STREET	FLUSHING	QUEENS	40.7713109	-73.831525
12836416	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Residential (NJ1)	11379	85-28 60 DRIVE	MIDDLE VILL	QUEENS	40.7289085	-73.873247
12836425	9/1/02	DEP	Noise	Noise, Barking Dog (NRS)	10016	EAST 26 STREET	NEW YORK	MANHATTAN	40.7384041	-73.976202
12843207	9/1/03	DEP	Noise	Noise: Other Noise Sources (Use Comments) (NZZ)	10031	364 BROADWAY	NEW YORK	MANHATTAN	40.8299097	-73.947911
12843308	9/1/03	DEP	Noise	Noise: Jack Hammering (NC2)	10012	79 MERCER STREET	NEW YORK	MANHATTAN	40.7228767	-74.000083
12836665	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	11106	35-36 24 STREET	ASTORIA	QUEENS	40.759585	-73.933456
12836666	9/1/02	DEP	Noise	Noise, Barking Dog (NRS)	11234	2224 EAST 57 PLACE	BROOKLYN	BROOKLYN	40.6093554	-73.912914
12836693	9/1/02	DEP	Noise	Noise, Barking Dog (NRS)	11421	96-16 JAMAICA AVENUE	WOODHABE	QUEENS	40.6945149	-73.84905
12836701	9/1/02	DEP	Noise	Noise, Barking Dog (NRS)	11207	200-202 104 AVENUE	BROOKLYN	BROOKLYN	40.6530656	-73.889789
12837244	9/1/02	DEP	Noise	Noise: Alarms (NR3)	11206	7 RENAISSANCE COURT	BROOKLYN	BROOKLYN	40.6995991	-73.934807
21	12842076	9/1/03	DEP	Noise	11226	250 CLARKSON AVENUE	BROOKLYN	BROOKLYN	40.6553259	-73.951537
22	12842077	9/1/03	DEP	Noise	10011	75 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318987	-73.999643
23	12842135	9/1/03	DEP	Noise	11220	5222 4 AVENUE	BROOKLYN	BROOKLYN	40.6451824	-74.013931
24	12842139	9/1/03	DEP	Noise	11221	24 DITMARS STREET	BROOKLYN	BROOKLYN	40.6967492	-73.933291
25	12842147	9/1/03	DEP	Noise	10006	151 BROADWAY	NEW YORK	MANHATTAN	40.7092979	-74.010435
26	12842177	9/1/03	DEP	Noise	11214	8721 BAY PARKWAY	BROOKLYN	BROOKLYN	40.5994577	-73.996381
27	12842184	9/1/03	DEP	Noise	11207	1065 VERMONT STREET	BROOKLYN	BROOKLYN	40.6540875	-73.885105
28	12842187	9/1/03	DEP	Noise	11106	34-60 9 STREET	ASTORIA	QUEENS	40.7640214	-73.940173
29	12836138	9/1/02	DEP	Noise	10007	43 PARK PLACE	NEW YORK	MANHATTAN	40.7135414	-74.009869
30	12842193	9/1/03	DEP	Noise	11229	1951 EAST 28 STREET	BROOKLYN	BROOKLYN	40.6035285	-73.944429
31	12836511	9/1/02	DEP	Noise	10471	5700 ARLINGTON AVENUE	BRONX	BRONX	40.9069851	-73.906877
32	12836521	9/1/02	DEP	Noise	10024	155 RIVERSIDE DRIVE	NEW YORK	MANHATTAN	40.7910379	-73.979538
33	12842292	9/1/03	DEP	Noise	10014	525 HUDSON STREET	NEW YORK	MANHATTAN	40.7340587	-74.006253
34	12842304	9/1/03	DEP	Noise	11378	66-14 GRAND AVENUE	MASPETH	QUEENS	40.7244346	-73.898116
35	12842307	9/1/03	DEP	Noise	10065	136 EAST 67 STREET	NEW YORK	MANHATTAN	40.7670726	-73.964166
36	12842310	9/1/03	DEP	Noise	10011	73 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318713	-73.999585
37	12842314	9/1/03	DEP	Noise	10014	32 GROVE STREET	NEW YORK	MANHATTAN	40.7326507	-74.004788
38	12842323	9/1/03	DEP	Noise	11231	49 WOODHULL STREET	BROOKLYN	BROOKLYN	40.6814664	-74.003404
39	12842332	9/1/03	DEP	Noise						

## 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=4523186789039326753;`

*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 existing 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*

**NYC OpenData** **1100+** Datasets Available

311  
Based on 311 Service Requests from 2010 to Present  
All 311 Service Requests from 2010 to present. This information is automatically updated daily.

que	Key	Created Da	Closed Date	Agency	Agency Na	Complaint Type	Descriptor	Location Type
1	528705	11/06/2013		NYPD	New York C	Noise - Commercial	Loud Talking	Store/Commercial
2	527775	11/06/2013		NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
3	530510	11/06/2013		NYPD	New York C	Noise - Vehicle	Car/Truck Horn	Street/Sidewalk
4	529483	11/06/2013		NYPD	New York C	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk
5	529423	11/06/2013		NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
6	532058	11/06/2013	11/06/2013 02:30:35 AM	NYPD	New York C	Blocked Driveway	Partial Access	Street/Sidewalk
7	529903	11/06/2013	11/06/2013 02:59:43 AM	NYPD	New York C	Blocked Driveway	Partial Access	Street/Sidewalk
8	530300	11/06/2013	11/06/2013 02:03:33 AM	NYPD	New York C	Noise - Vehicle	Engine Idling	Street/Sidewalk
9	526995	11/06/2013	11/06/2013 02:33:44 AM	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
10	526538	11/06/2013		NYPD	New York C	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk
11	528221	11/06/2013	11/06/2013 02:05:44 AM	NYPD	New York C	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk
12	527915	11/06/2013		NYPD	New York C	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk
13	528741	11/06/2013	11/06/2013 02:41:02 AM	NYPD	New York C	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant
14	528642	11/06/2013		DOT	Departmen	Highway Sign - Missing	Exit/Route	Highway
15	530302	11/06/2013	11/06/2013 02:36:53 AM	NYPD	New York C	Noise - Commercial	Loud Music/Party	Store/Commercial
16	528715	11/06/2013		NYPD	New York C	Blocked Driveway	No Access	Street/Sidewalk
17	526936	11/06/2013		DPR	Departmen	Overgrown Tree/Branches	Traffic Sign or Signal Blocked	Street
18	530450	11/06/2013		DOHMH	Departmen	Indoor Air Quality	Other (Explain Below)	3+ Family Apartment Building

\* DOES DESCRIPTOR READ “BARKING DOG”? Descriptor

Manage 593 More Views

## 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**= chooses 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 assessment 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`

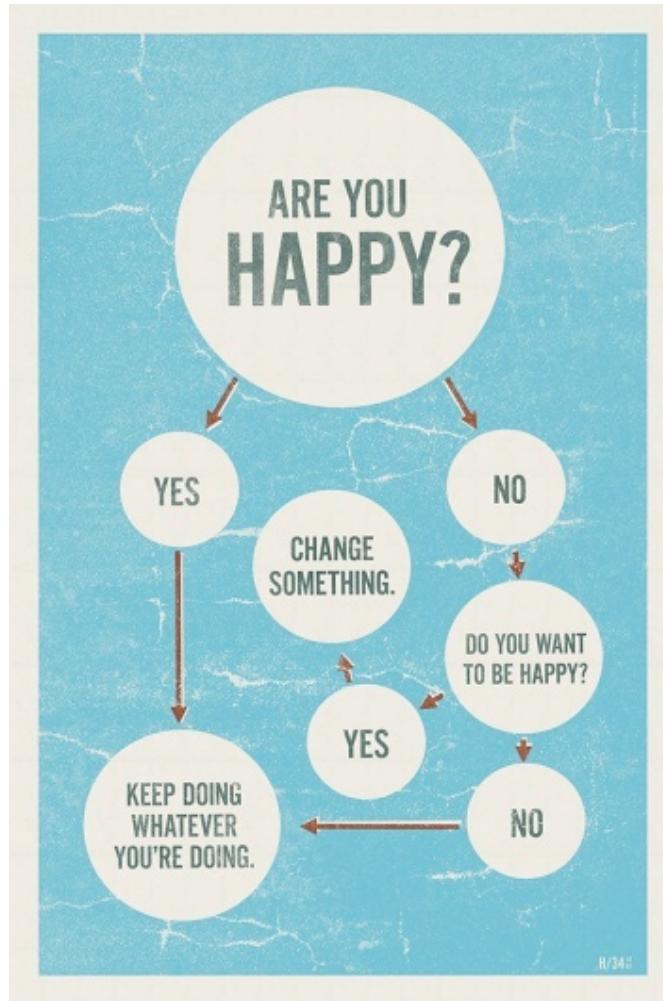
ex. `void setup (){`  
`// all setup code goes here`  
`} // functions`

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

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

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

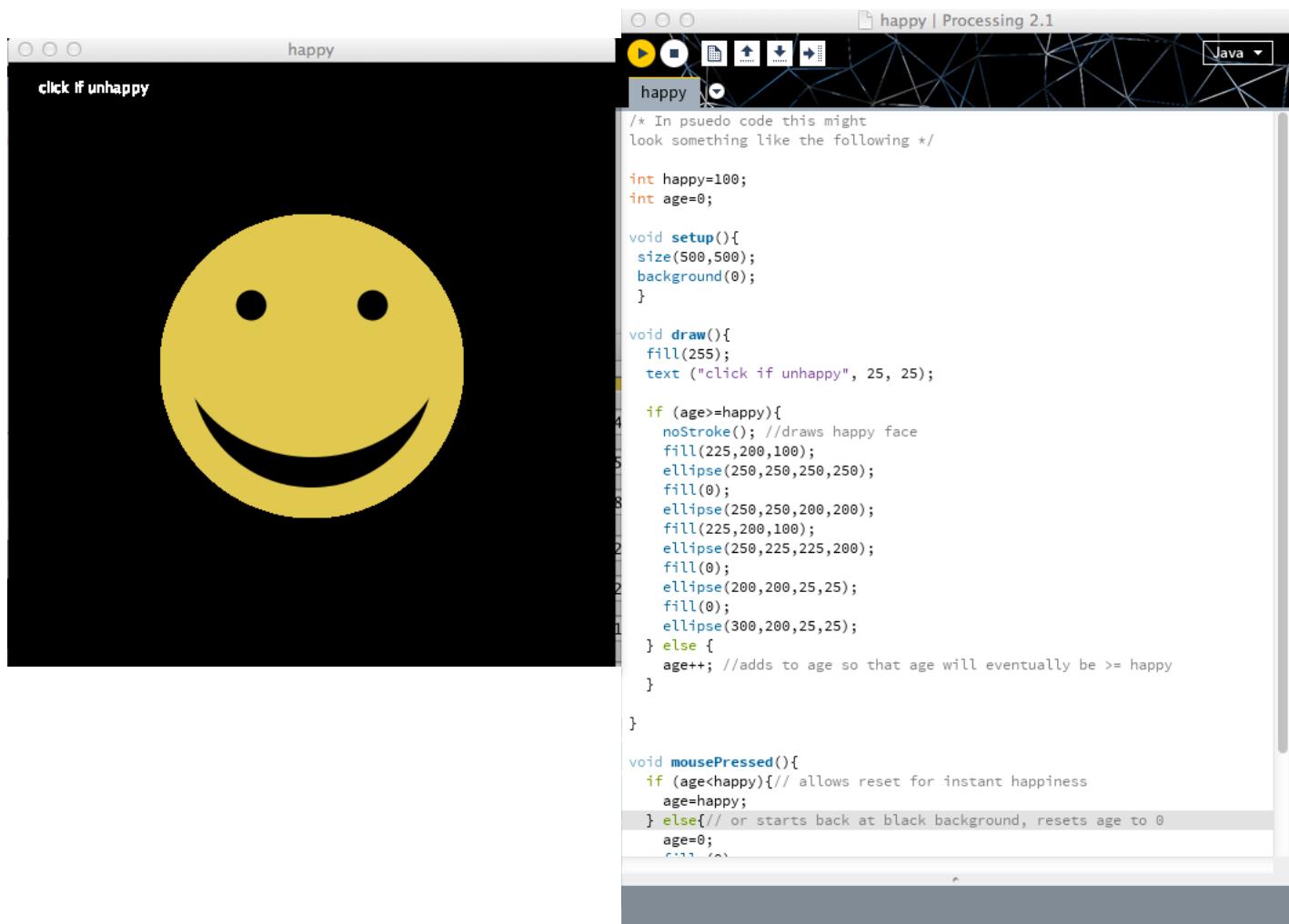
*find @ <http://www.processing.org/references/>*



## 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.



The screenshot shows the Processing 2.1 IDE interface. The title bar says "happy | Processing 2.1". The code editor contains the following Java code:

```
/* In psuedo code this might
look something like the following */

int happy=100;
int age=0;

void setup(){
  size(500,500);
  background(0);
}

void draw(){
  fill(255);
  text ("click if unhappy", 25, 25);

  if (age>=happy){
    noStroke(); //draws happy face
    fill(225,200,100);
    ellipse(250,250,250,250);
    fill(0);
    ellipse(250,250,200,200);
    fill(225,200,100);
    ellipse(250,225,225,200);
    fill(0);
    ellipse(200,200,25,25);
    fill(0);
    ellipse(300,200,25,25);
  } else {
    age++; //adds to age so that age will eventually be >= happy
  }
}

void mousePressed(){
  if (age<happy){ // allows reset for instant happiness
    age=happy;
  } else{ // or starts back at black background, resets age to 0
    age=0;
  }
}
```

The visual output on the left shows a yellow smiley face with a black outline and a wide smile, centered on a black background. The text "click if unhappy" is displayed in purple at the top left of the screen.

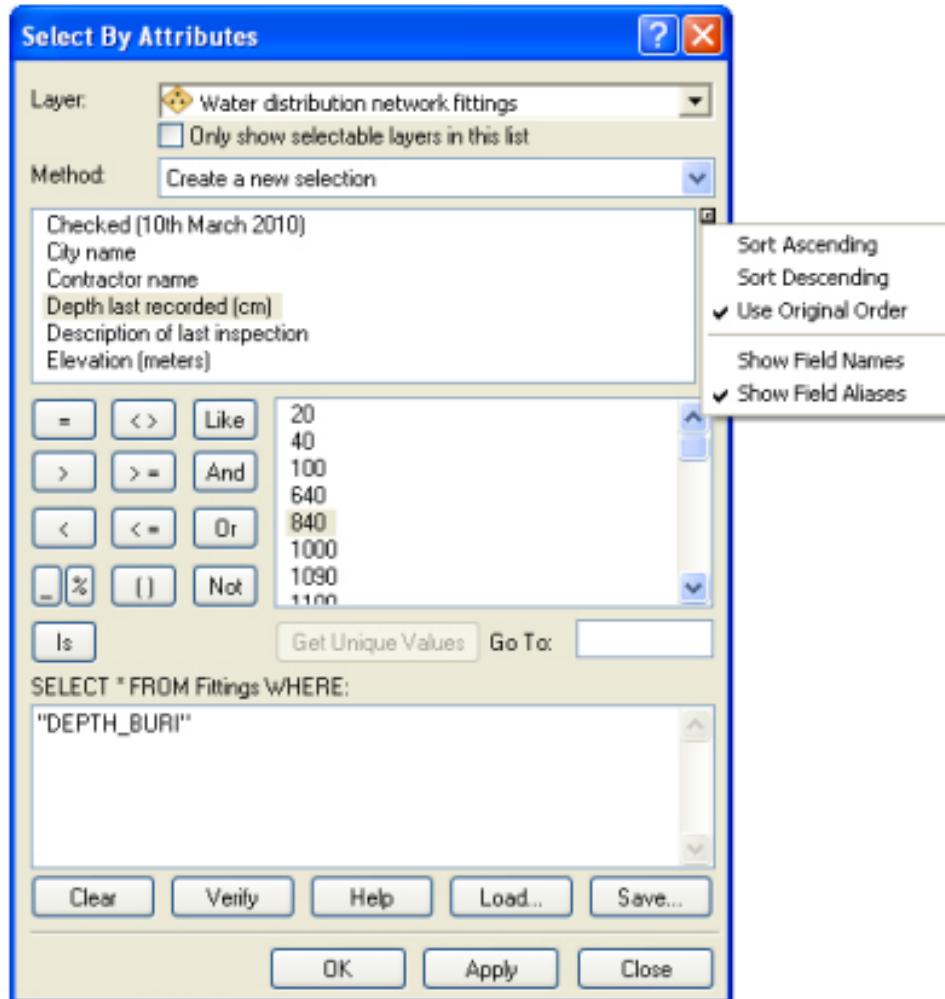
## INPUT: THE CODE BEHIND THE VISUALS

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

Explanations will be typed in this area below.

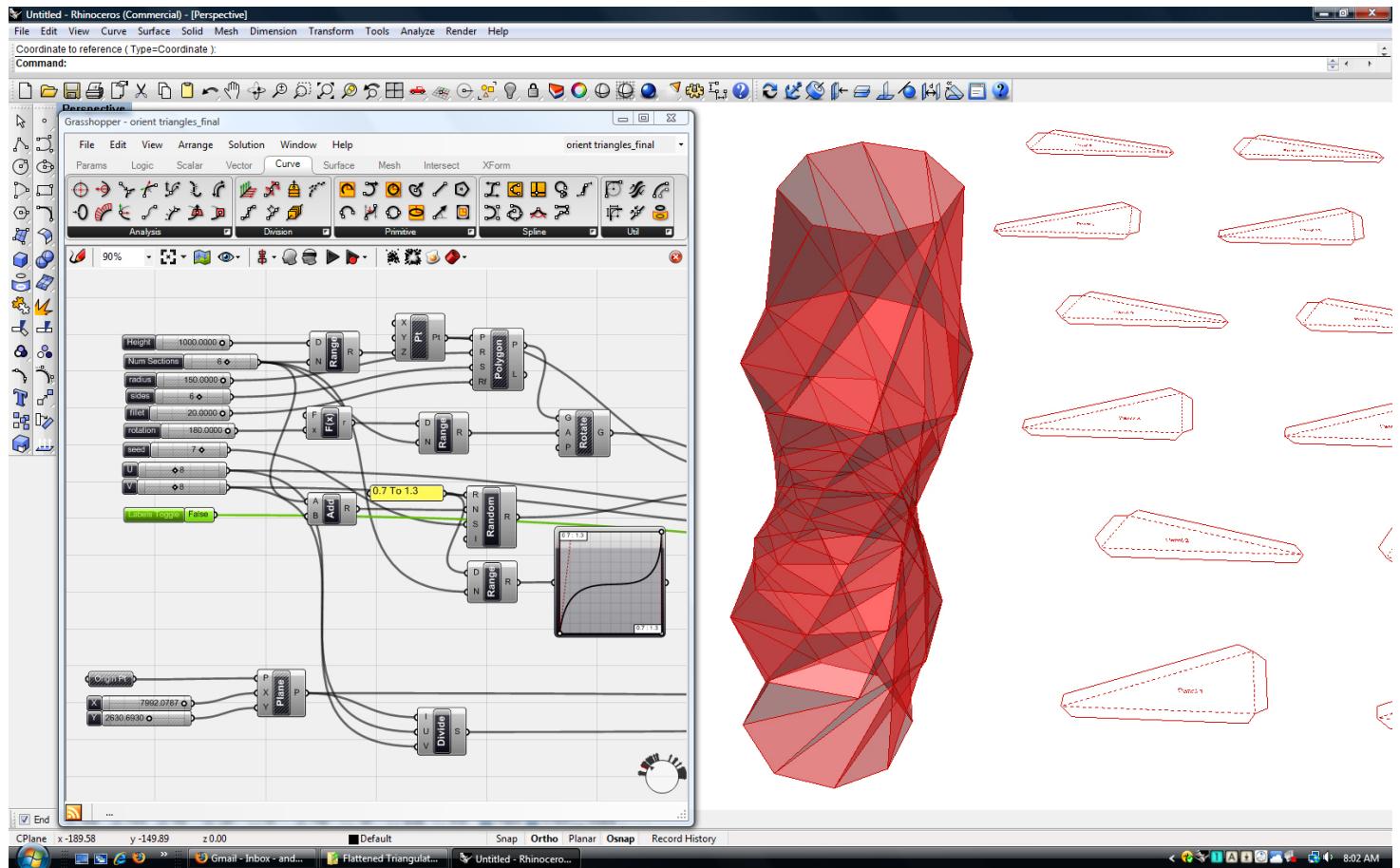
This has a few benefits, despite being a pain:

- by re-typing code, you'll get use to being cautious with syntax,
- by re-typing code, you'll get use to debugging your typos,  
and
- 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.



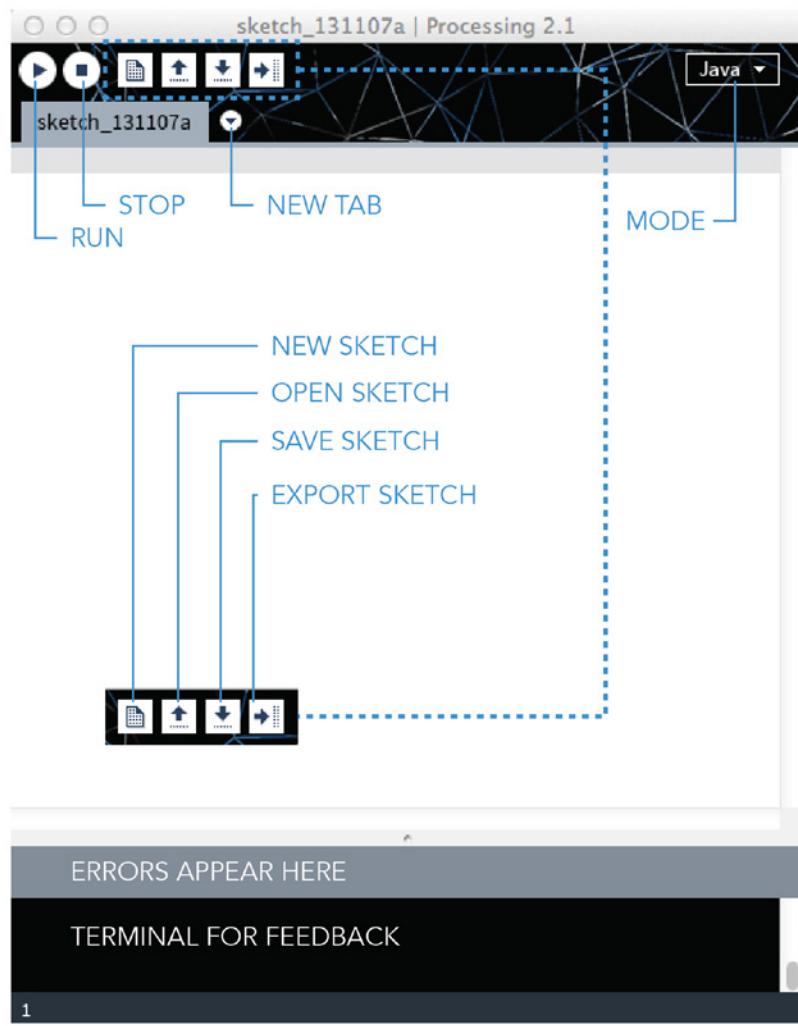
## PARAMETER ADJUSTMENT **GRASSHOPPER OBJECTS**

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

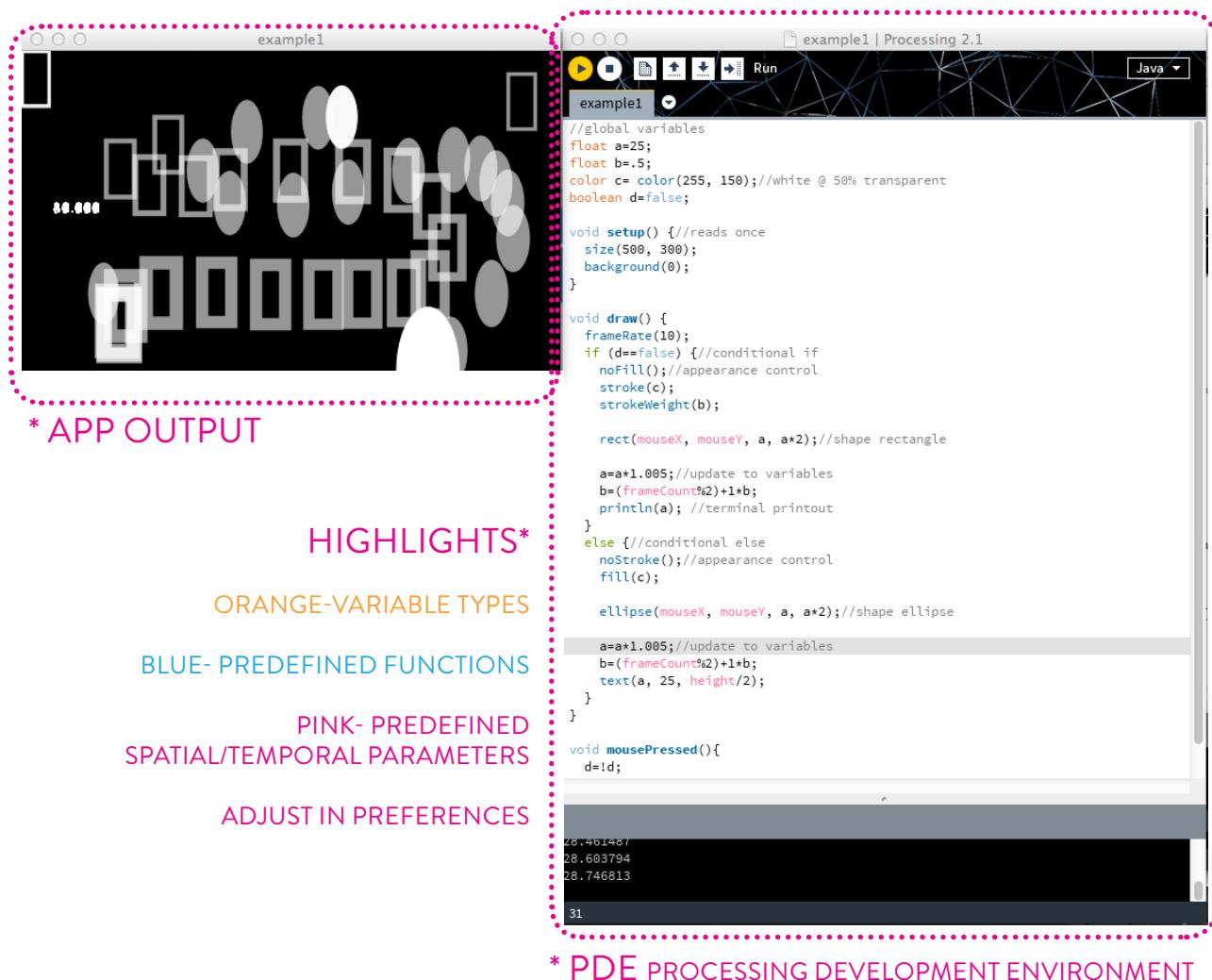
EDIT/FORMAT

EDIT/COMMENT

SKETCH/IMPORT LIBRARY

TOOLS/CREATE FONT

TOOLS/COLOR SELECTOR



## 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:

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

```

//global variables
float a=25;
float b=.5;
color c= color(255, 150); //white @ 50% transparent
boolean d=false;

void setup() { //reads once
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(10);
  if (d==false) { //conditional if
    noFill(); //appearance control
    stroke(c);
    strokeWeight(b);

    rect(mouseX, mouseY, a, a*2); //shape rectangle
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    println(a); //terminal printout
  } else { //conditional else
    noStroke(); //appearance control
    fill(c);

    ellipse(mouseX, mouseY, a, a*2); //shape ellipse
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    text(a, 25, height/2);
  }
}

void mousePressed(){
  d=!d;
}

```

\* GLOBAL VARIABLES

\* SETUP (STATIC)

\* DRAW (DYNAMIC)

\* MOUSE PRESSED (DYNAMIC)

64.81229  
65.136345  
65.46203

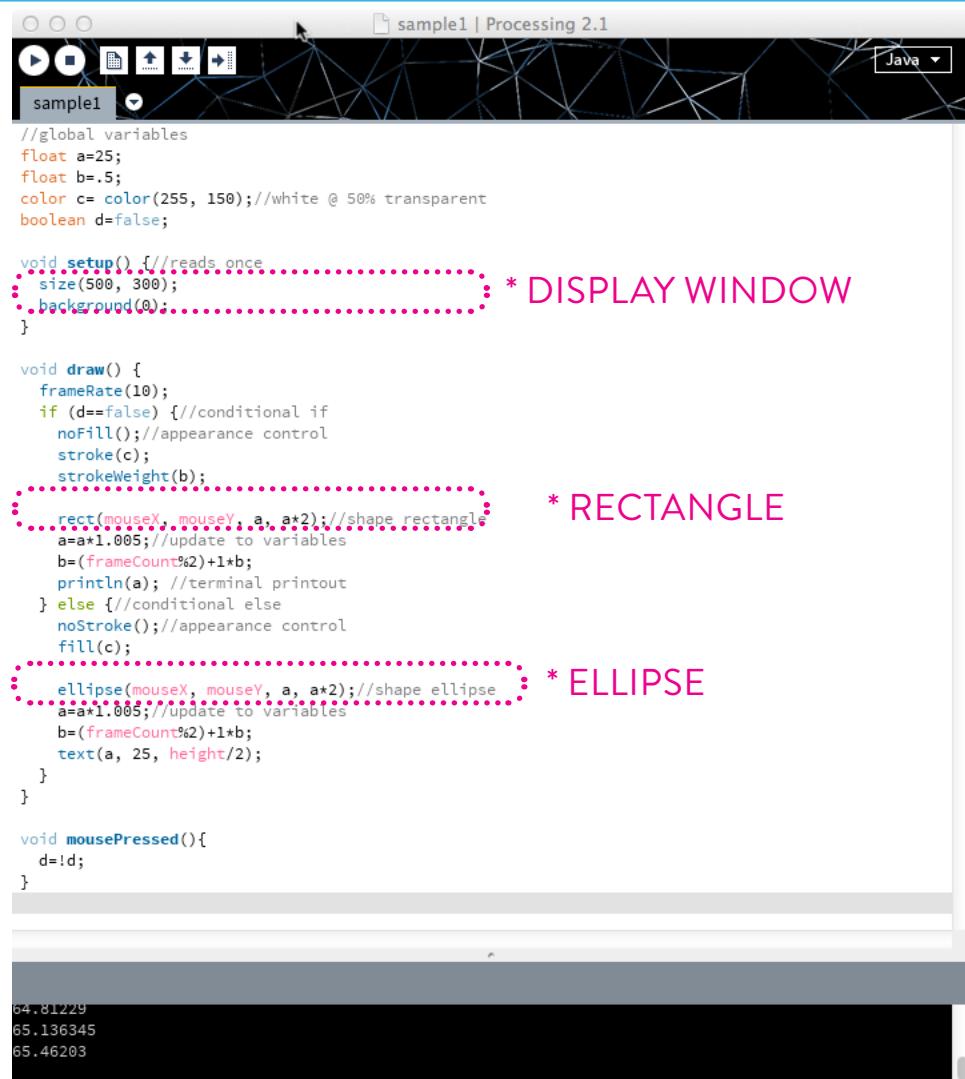
## SKETCH: EXAMPLE 1 PARTS/ORDER

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

Global variables are initialized (announced) 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).



```

//global variables
float a=25;
float b=.5;
color c= color(255, 150); //white @ 50% transparent
boolean d=false;

void setup() { //reads once
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(10);
  if (d==false) { //conditional if
    noFill(); //appearance control
    stroke(c);
    strokeWeight(b);
    rect(mouseX, mouseY, a, a*2); //shape rectangle
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    println(a); //terminal printout
  } else { //conditional else
    noStroke(); //appearance control
    fill(c);
    ellipse(mouseX, mouseY, a, a*2); //shape ellipse
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    text(a, 25, height/2);
  }
}

void mousePressed(){
  d=!d;
}

```

64.81229  
65.136345  
65.46203

## SKETCH: EXAMPLE 1 SHAPES

This simple sketch has two shapes drawn against a background.

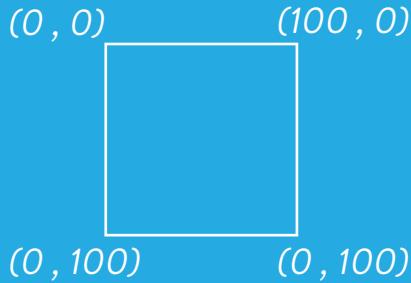
`size (width, height);` defines the window for the drawing. `Background (0)` defines its' color (black). These are static and do not refresh.

`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 similarly 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*

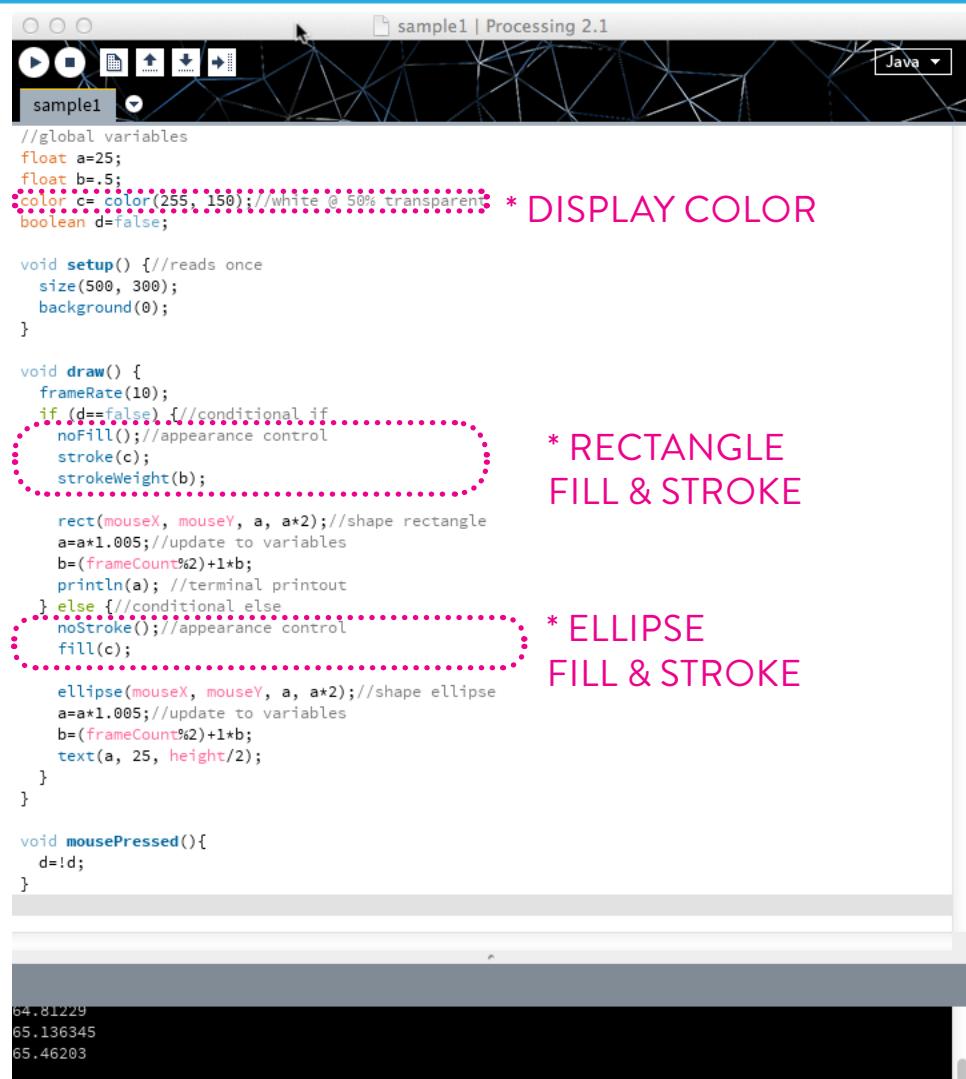
*ellipse ( x, y, diameter, diameter); ellipse, default centered*

*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/>*



```
//global variables
float a=25;
float b=.5;
color c= color(255, 150); //white @ 50% transparent * DISPLAY COLOR
boolean d=false;

void setup() {//reads once
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(10);
  if (d==false) { //conditional if
    noFill(); //appearance control
    stroke(c);
    strokeWeight(b);
    rect(mouseX, mouseY, a, a*2); //shape rectangle
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    println(a); //terminal printout
  } else { //conditional else
    noStroke(); //appearance control
    fill(c);
    ellipse(mouseX, mouseY, a, a*2); //shape ellipse
    a=a*1.005; //update to variables
    b=(frameCount%2)+1*b;
    text(a, 25, height/2);
  }
}

void mousePressed(){
  d=!d;
}

64.81229
65.136345
65.46203
```

## 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.

```

//global variables
float a=25;
float b=5;
color c=color(255, 150); //white @ 50% transparent
boolean d=false;

void setup() { //reads once
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(10);
  if (d==false) { //conditional if
    noFill(); //appearance control
    stroke(c);
    strokeWeight(b);
    rect(mouseX, mouseY, a, a*2); //shape rectangle
    a=a*1.005; //update to variables
    b=(frameCount*2)+1*b;
    println(a); //terminal printout
  } else { //conditional else
    noStroke(); //appearance control
    fill(c);
    ellipse(mouseX, mouseY, a, a*2); //shape ellipse
    a=a*1.005; //update to variables
    b=(frameCount*2)+1*b;
    text(a, 25, height/2);
  }
}

void mousePressed(){
  d=!d;
}

```

\* A, B DECLARED

\* D DECLARED

\* FRAMERATE SLOWS DRAW TO 10/SECOND

\* USED IN STROKE, RECTANGLE UPDATED FOR NEXT TIME THRU

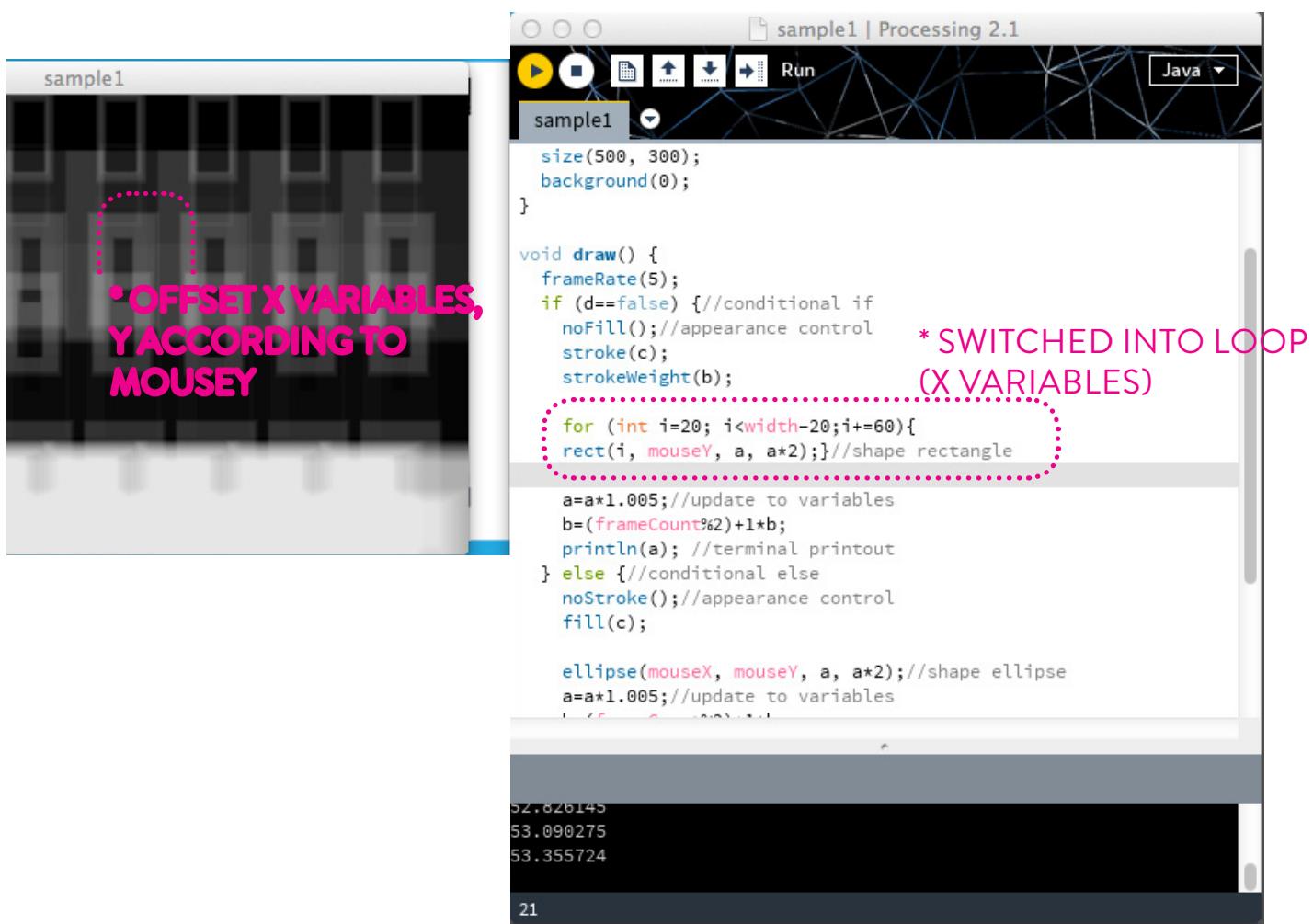
\* USED IN ELLIPSE UPDATED FOR NEXT TIME THRU

\* UPDATED BY MOUSEPRESS SHIFTS CONDITIONAL OUTCOME

## 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 in 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

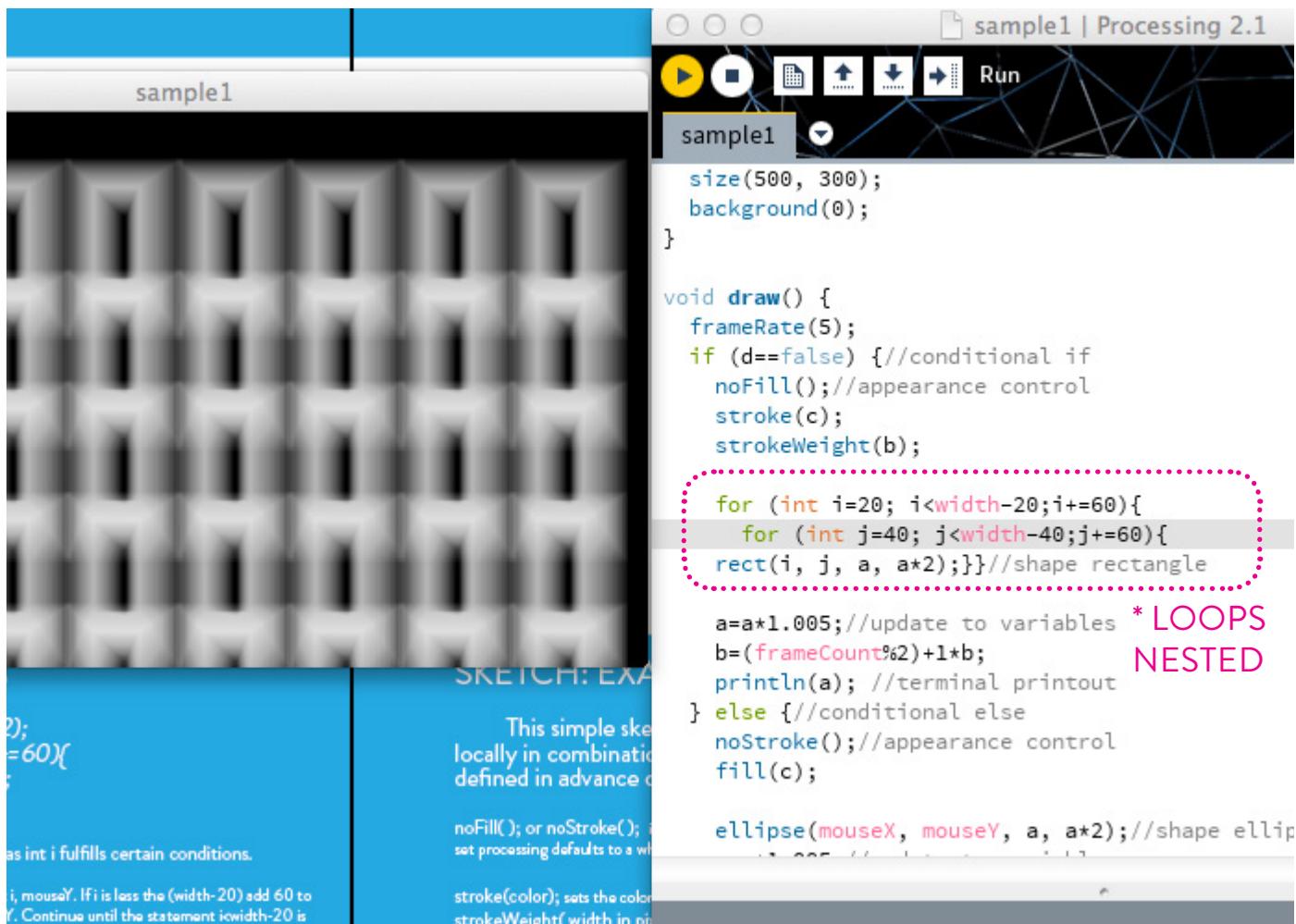
```

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 than (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.

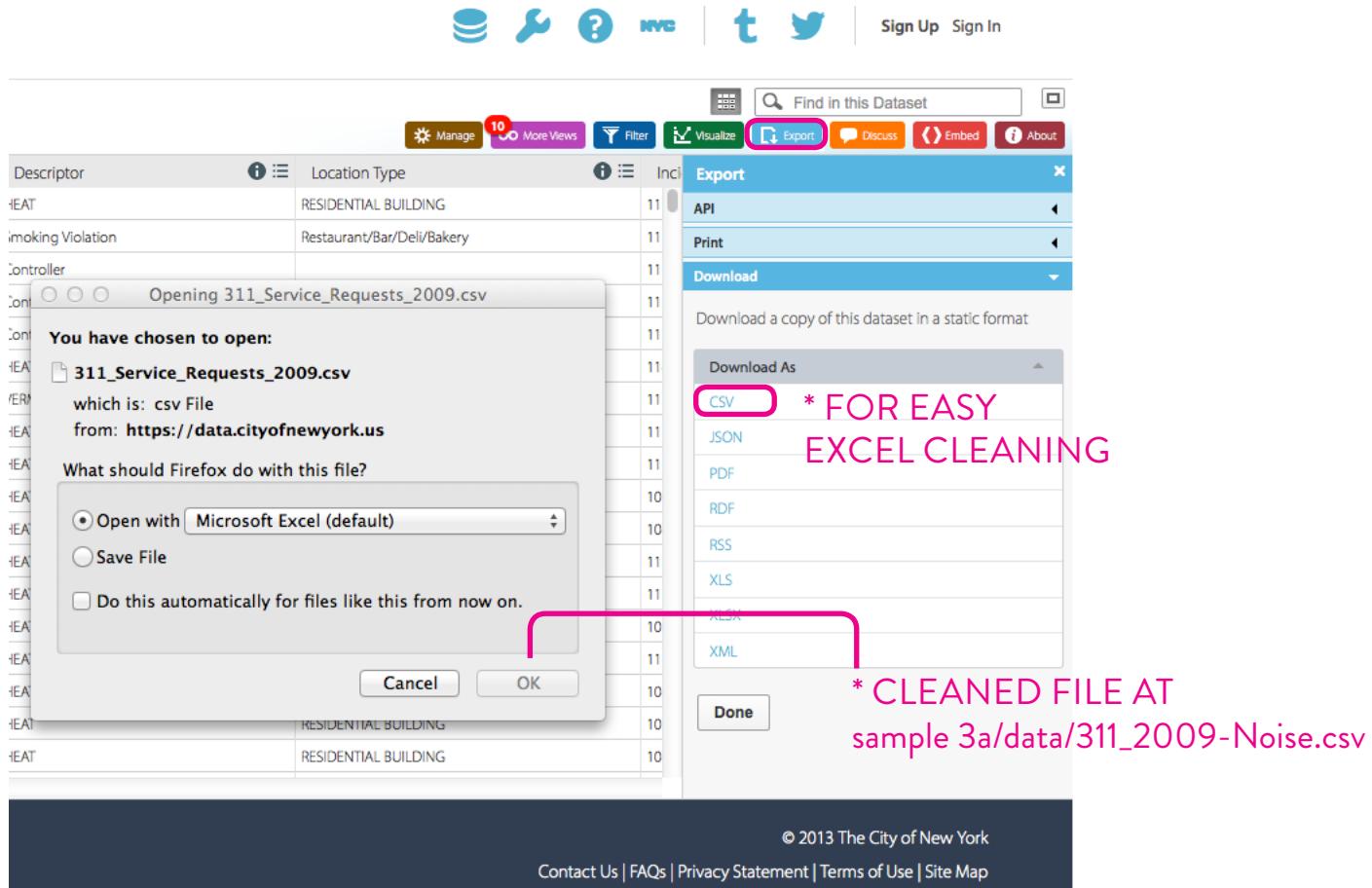
The grid of rectangles that results draws a series of columns of rectangle across the page. For  $i=20$ , it executes all at  $j$  positions and then moves to the next  $i$  variable,  $i=80\dots$  and so on.

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



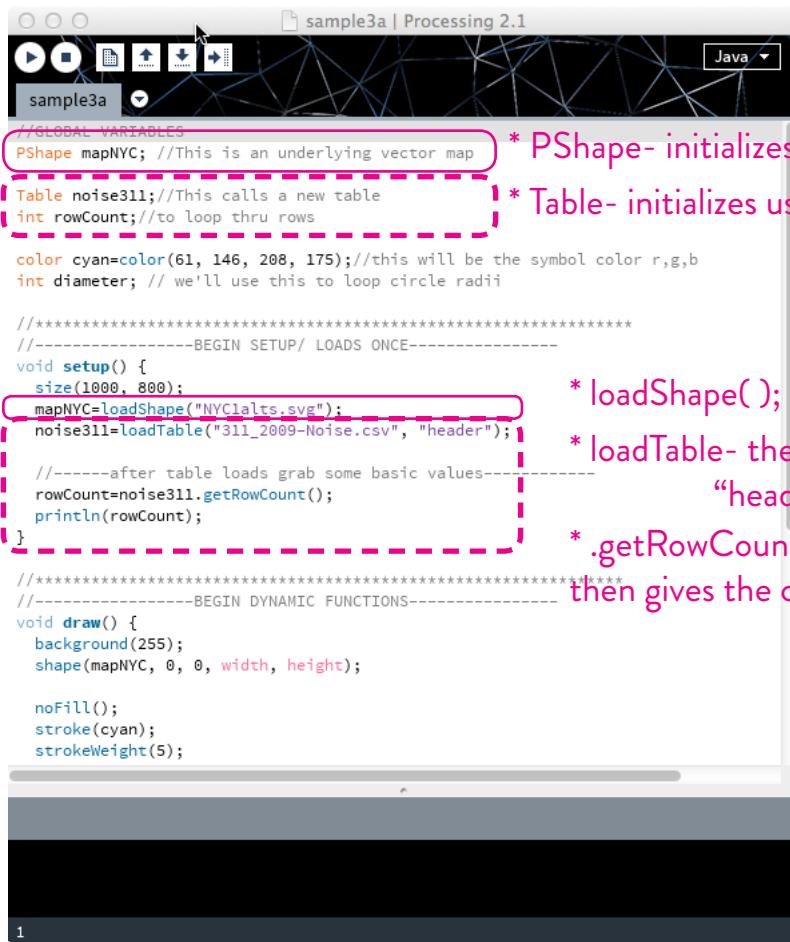
## SKETCH: EXAMPLE 3A ORIGINAL & CLEANED SOURCES

<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"



```

//GLOBAL VARIABLES
PShape mapNYC; //This is an underlying vector map
Table noise311;//This calls a new table
int rowCount;//to loop thru rows

color cyan=color(61, 146, 208, 175);//this will be the symbol color r,g,b
int diameter; // we'll use this to loop circle radii

//*****-----BEGIN SETUP/ LOADS ONCE-----*****
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYCnights.svg");
  noise311=loadTable("311_2009-Noise.csv", "header");
}

//-----after table loads grab some basic values-----
rowCount=noise311.getRowCount();
println(rowCount);
}

//*****-----BEGIN DYNAMIC FUNCTIONS-----*****
void draw() {
  background(255);
  shape(mapNYC, 0, 0, width, height);

  noFill();
  stroke(cyan);
  strokeWeight(5);
}

```

\* 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

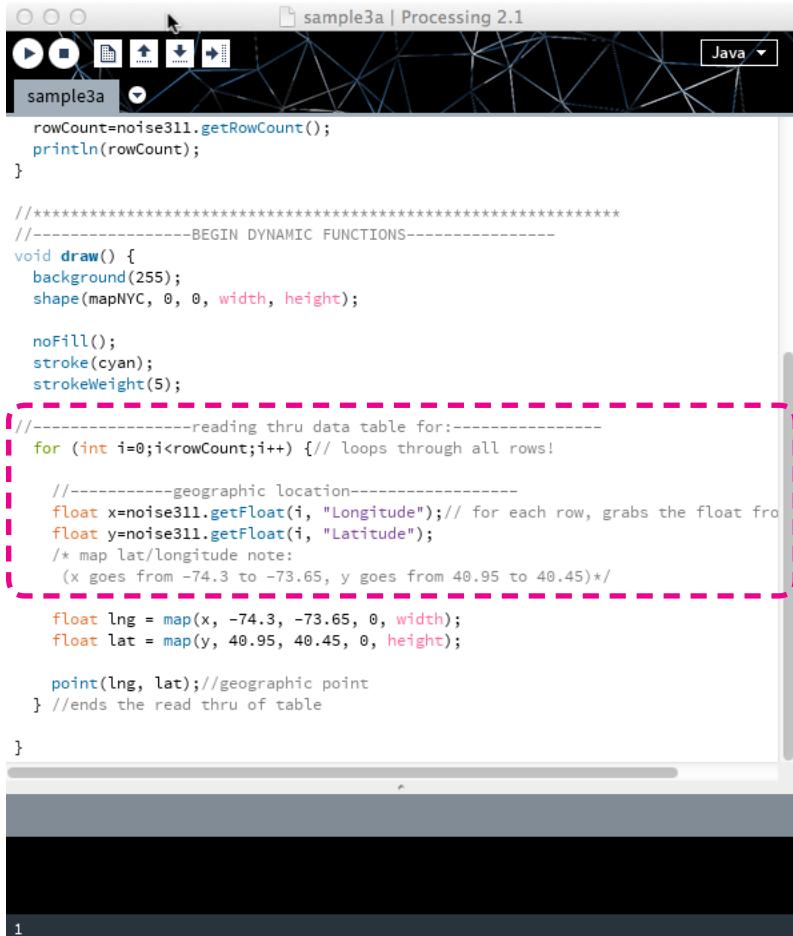
\* .getRowCount( );  
then gives the count of those rows for later use

## 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 sketch'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).



```
sample3a | Processing 2.1
Java

rowCount=noise311.getRowCount();
println(rowCount);
}

//*****-----BEGIN DYNAMIC FUNCTIONS-----//
void draw() {
background(255);
shape(mapNYC, 0, 0, width, height);

noFill();
stroke(cyan);
strokeWeight(5);

//-----reading thru data table for:-----
for (int i=0;i<rowCount;i++) { // loops through all rows!

//-----geographic location-----
float x=noise311.getFloat(i, "Longitude");// for each row, grabs the float fro
float y=noise311.getFloat(i, "Latitude");
/* map lat/longitude note:
(x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

float lng = map(x, -74.3, -73.65, 0, width);
float lat = map(y, 40.95, 40.45, 0, height);

point(lng, lat); //geographic point
} //ends the read thru of table

}

1
```

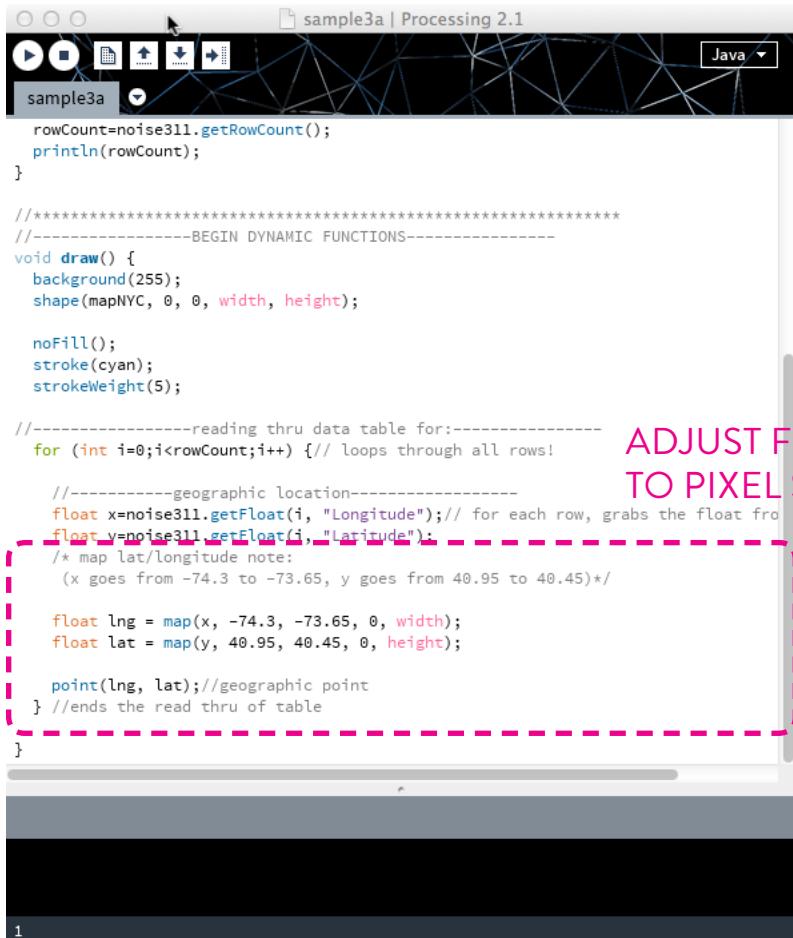
\*loop through table rows to get values

\* .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).



```
sample3a | Processing 2.1
Java

rowCount=noise311.getRowCount();
println(rowCount);
}

//*****-----BEGIN DYNAMIC FUNCTIONS-----//
void draw() {
background(255);
shape(mapNYC, 0, 0, width, height);

noFill();
stroke(cyan);
strokeWeight(5);

//-----reading thru data table for:-----
for (int i=0;i<rowCount;i++) { // loops through all rows!

//-----geographic location-----
float x=noise311.getFloat(i, "Longitude");// for each row, grabs the float fro
float y=noise311.getFloat(i, "Latitude");
/* map lat/longitude note:
(x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

float lng = map(x, -74.3, -73.65, 0, width);
float lat = map(y, 40.95, 40.45, 0, height);

point(lng, lat); //geographic point
} //ends the read thru of table
}

1
```

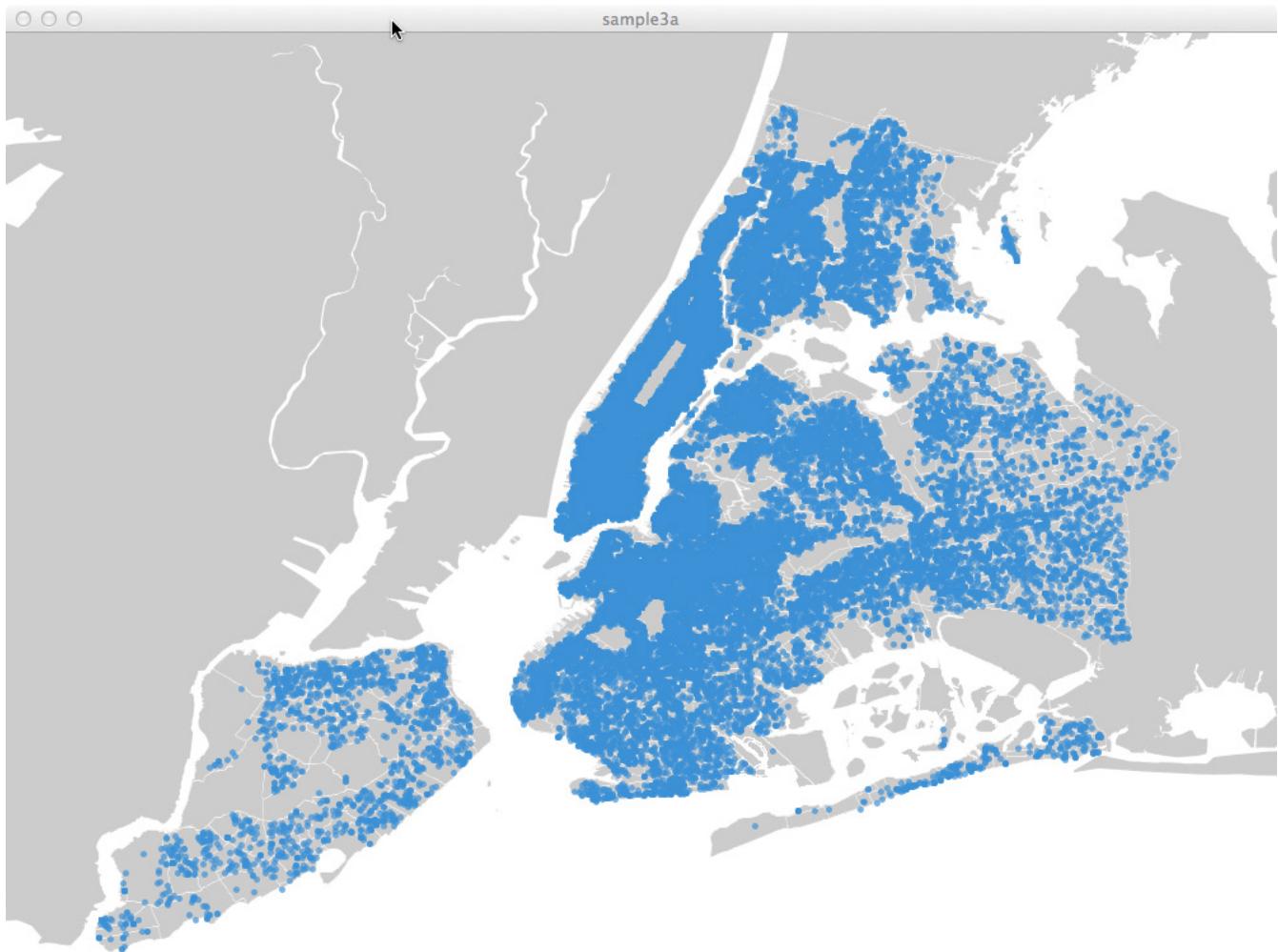
### ADJUST FROM GEOGRAPHIC TO PIXEL SPACE OF THE SKETCH

- \* 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 remap 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()*

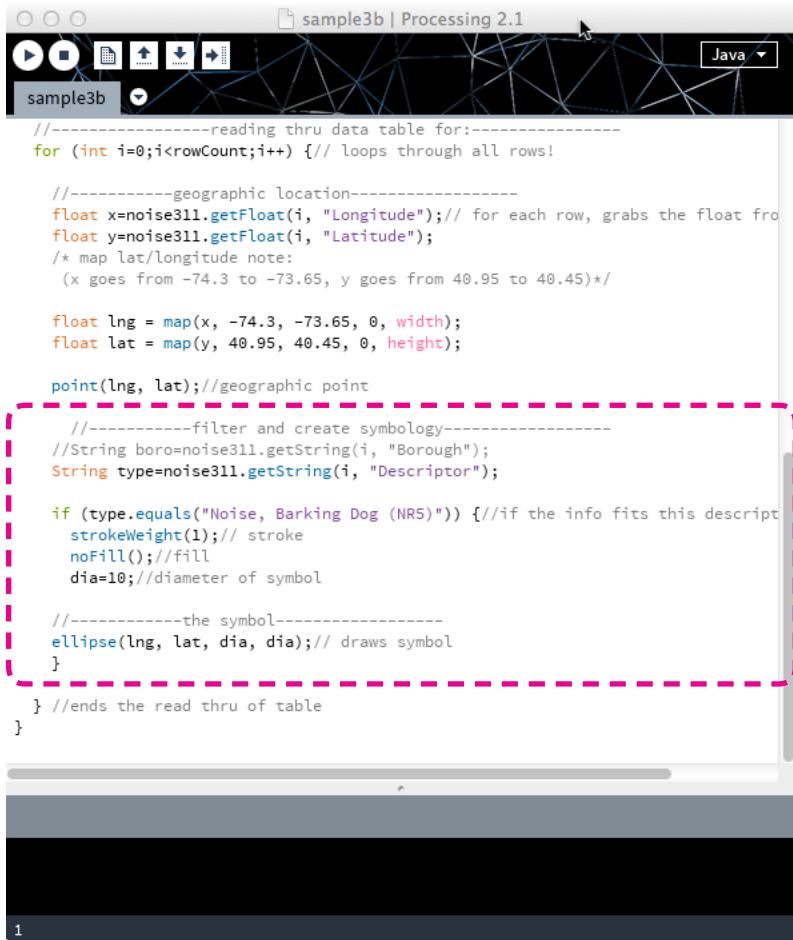
*addColumn()* Adds a new column to a table  
*removeColumn()* Removes a column from a table  
*getColumnCount()* Gets the number of columns in a table  
*getRowCount()* Gets the number of rows in a table  
*clearRows()* Removes all rows from a table  
*addRow()* Adds a row to a table  
*removeRow()*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



```
//-----reading thru data table for:-----
for (int i=0;i<rowCount;i++) { // loops through all rows!

    //-----geographic location-----
    float x=noise311.getFloat(i, "Longitude");// for each row, grabs the float fro
    float y=noise311.getFloat(i, "Latitude");
    /* map lat/longitude note:
     (x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

    float lng = map(x, -74.3, -73.65, 0, width);
    float lat = map(y, 40.95, 40.45, 0, height);

    point(lng, lat); //geographic point

    //-----filter and create symbology-----
    //String boro=noise311.getString(i, "Borough");
    String type=noise311.getString(i, "Descriptor");

    if (type.equals("Noise, Barking Dog (NR5)")) { //if the info fits this descrip
        strokeWeight(1); // stroke
        noFill(); //fill
        dia=10; //diameter of symbol

        //-----the symbol-----
        ellipse(lng, lat, dia, dia); // draws symbol
    }

} //ends the read thru of table
}
```

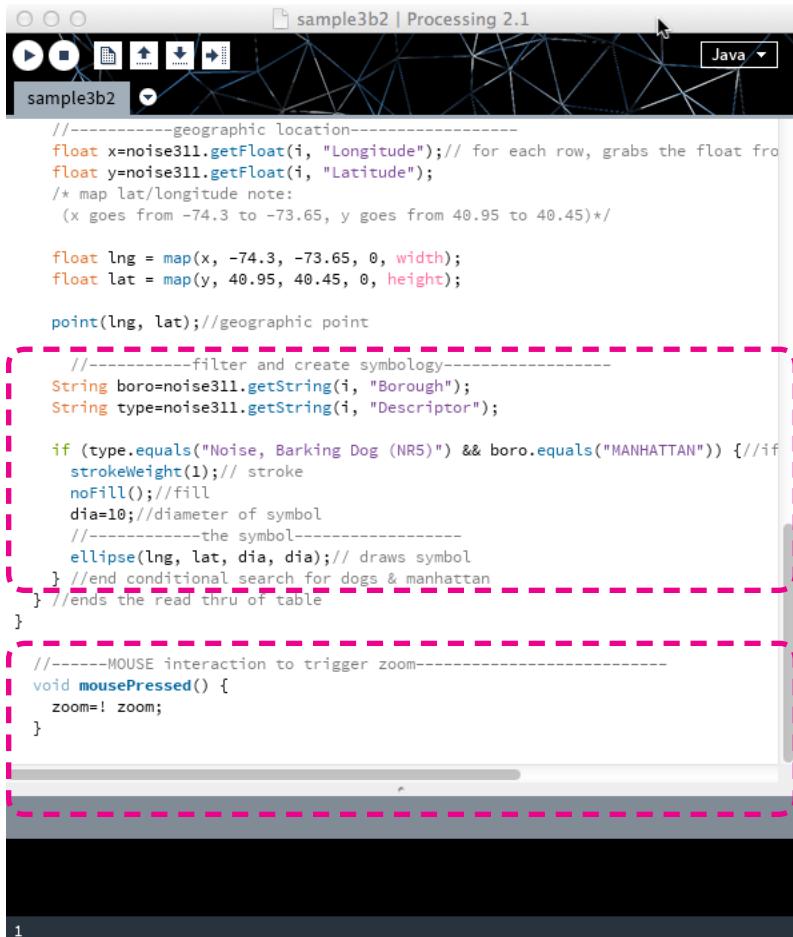
\* `.getString( row, column by header)`  
works like `.getFloat()`

\* `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.



The screenshot shows the Processing 2.1 interface with the sketch titled "sample3b2". The code is as follows:

```
-----geographic location-----
float x=noise311.getFloat(i, "Longitude");// for each row, grabs the float fro
float y=noise311.getFloat(i, "Latitude");
/* map lat/longitude note:
(x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

float lng = map(x, -74.3, -73.65, 0, width);
float lat = map(y, 40.95, 40.45, 0, height);

point(lng, lat); //geographic point

-----filter and create symbology-----
String boro=noise311.getString(i, "Borough");
String type=noise311.getString(i, "Descriptor");

if (type.equals("Noise, Barking Dog (NR5)") && boro.equals("MANHATTAN")) { //if
  strokeWeight(1); // stroke
  noFill(); //fill
  dia=10; //diameter of symbol
  //-----the symbol-----
  ellipse(lng, lat, dia, dia); // draws symbol
} //end conditional search for dogs & manhattan
} //ends the read thru of table

//-----MOUSE interaction to trigger zoom-----
void mousePressed() {
  zoom+=1;
}
```

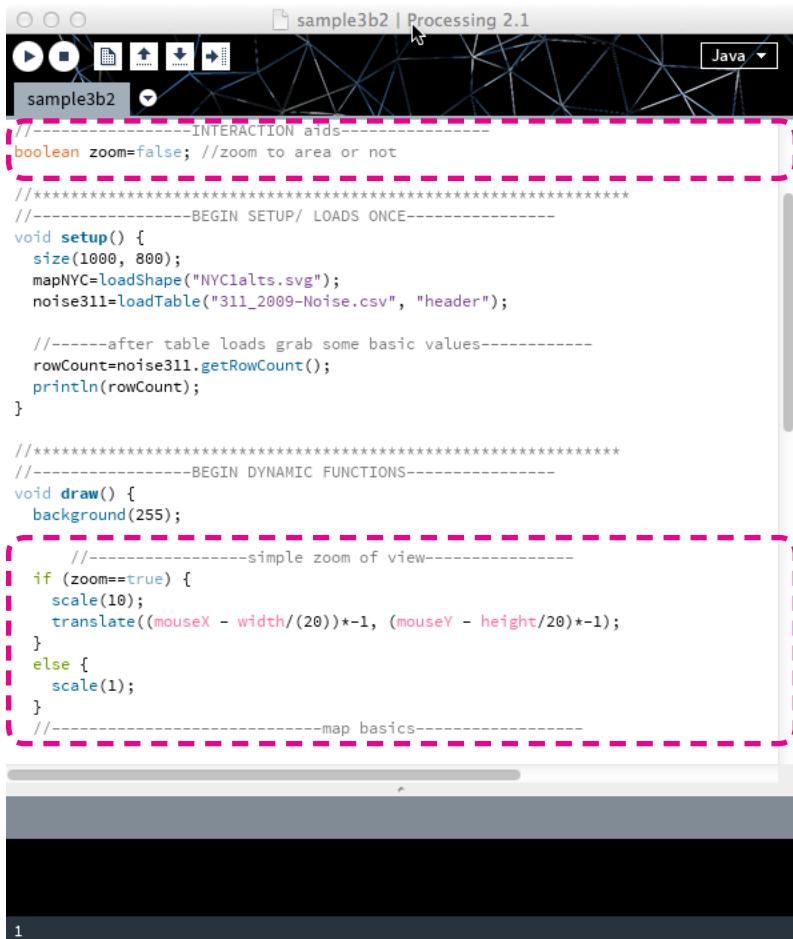
Create another variable boro and add to type.equals with a logical && to filter for "MANHATTAN" as well as Dog Barking

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.



```
//-----INTERACTION aids-----
boolean zoom=false; //zoom to area or not

//-----BEGIN SETUP/ LOADS ONCE-----
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYCaltz.svg");
  noise311=loadTable("311_2009-Noise.csv", "header");

  //-----after table loads grab some basic values-----
  rowCount=noise311.getRowCount();
  println(rowCount);
}

//-----BEGIN DYNAMIC FUNCTIONS-----
void draw() {
  background(255);

  //-----simple zoom of view-----
  if (zoom==true) {
    scale(10);
    translate((mouseX - width/(20))*-1, (mouseY - height/20)*-1);
  }
  else {
    scale(1);
  }
  //-----map basics-----
}
```

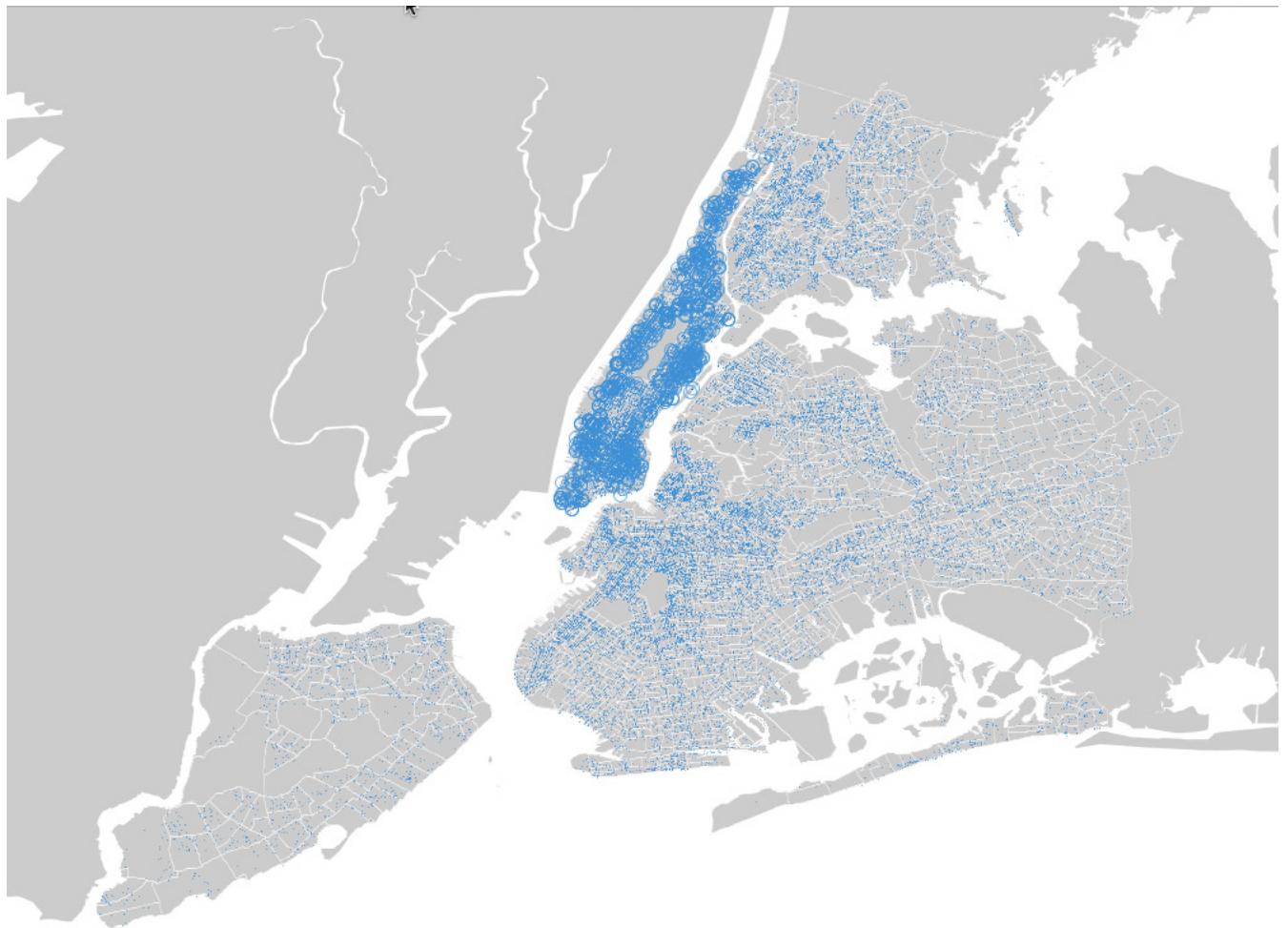
Add “Boolean zoom=false;” up at the top

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.



```

dogTable=new Table();
manhattanTable=new Table();

//-----after table loads grab some basic values-----
rowCount=noise311.getRowCount();
println(rowCount);

//-----do permanent filtering/write additional tables-----
for (TableRow barking : noise311.findRows("Noise, Barking Dog (NRS)", "Descriptor")) {
  TableRow newRow = dogTable.addRow(barking);
}
saveTable(dogTable, "data/dog311.csv");// all barking dogs into a table

for (TableRow barkingM : dogTable.findRows("MANHATTAN", 8)) {
  TableRow newRow = manhattanTable.addRow(barkingM);
}
saveTable(manhattanTable, "data/dogM311.csv");// all barking manhattan as table

rowCountDM=manhattanTable.getRowCount();
println(rowCountDM);

}

//*****
//-----BEGIN DYNAMIC FUNCTIONS-----
void draw() {

}

Done Saving.

50

```

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

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

\*TableRow newRow= table.addRow( row);

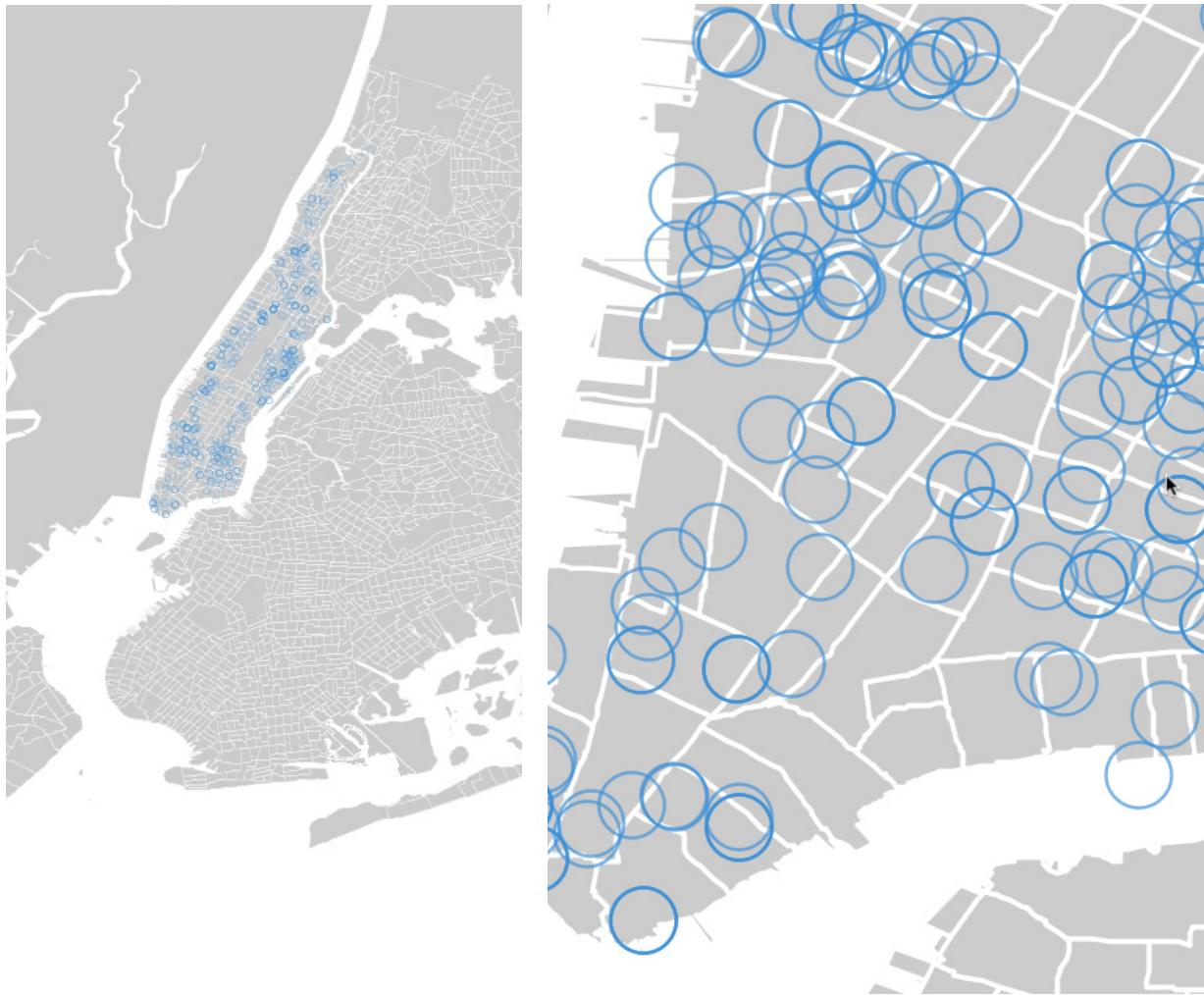
\*saveTable(table, “filename”);

## 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 function instead of iterating through all rows. 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. . .

9 WK 1-BASIC PLOTTING

**59 WK 2-MATH, MULTIPLE FORMS, INTERACTION**

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

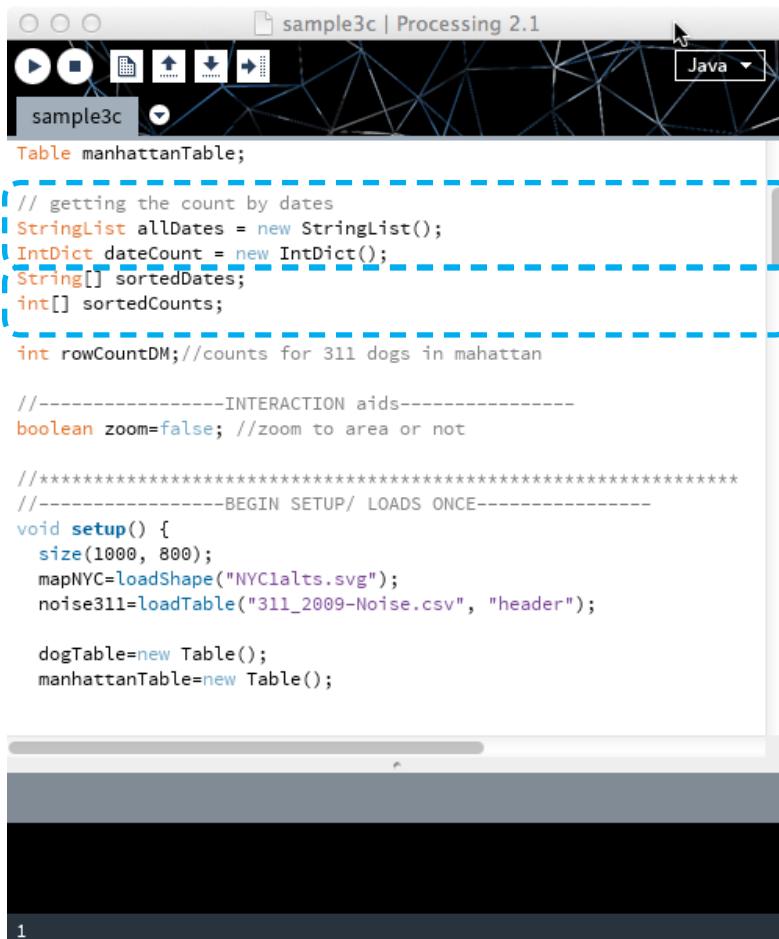
## WK 2-MATH, INTERACTIONS, & MULTIPLE DISPLAY FORMS

- 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 2

## SIMPLE MATH FROM TABLES

- `StringLists`
- `InterDicts`
- `Arrays`



The screenshot shows the Processing 2.1 software interface with the sketch titled 'sample3c'. The code is as follows:

```
Table manhattanTable;  
  
// getting the count by dates  
StringList allDates = new StringList();  
IntDict dateCount = new IntDict();  
String[] sortedDates;  
int[] sortedCounts;  
  
int rowCountDM;//counts for 311 dogs in manhattan  
  
//-----INTERACTION aids-----  
boolean zoom=false; //zoom to area or not  
  
//*****-----BEGIN SETUP/ LOADS ONCE-----  
void setup() {  
  size(1000, 800);  
  mapNYC=loadShape("NYCialts.svg");  
  noise311=loadTable("311_2009-Noise.csv", "header");  
  
  dogTable=new Table();  
  manhattanTable=new Table();  
}
```

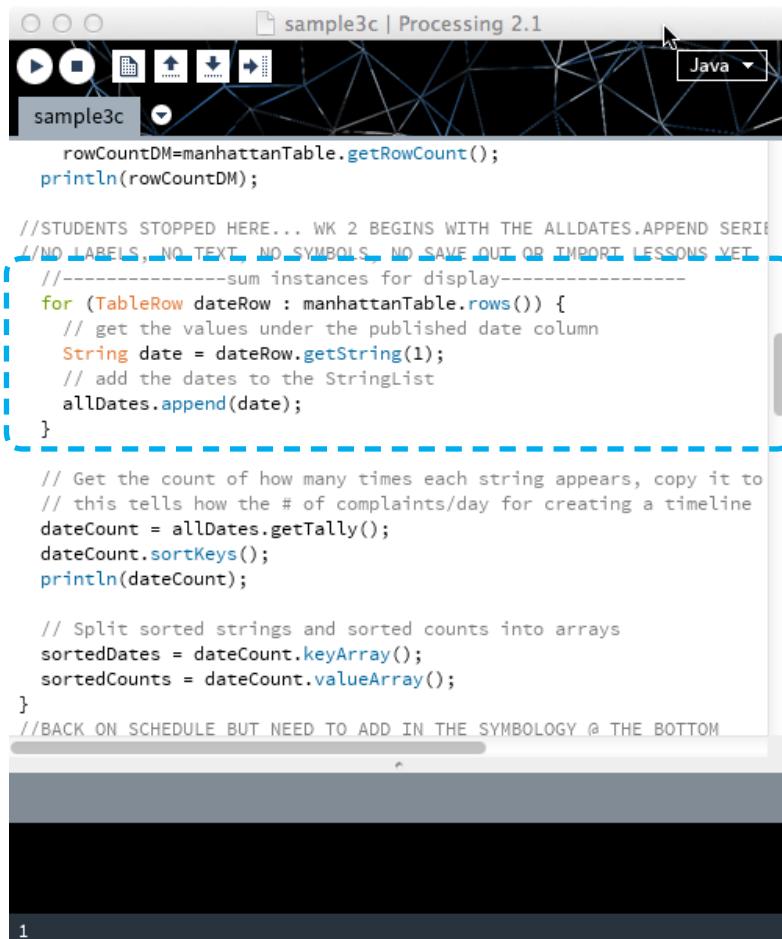
Add “StringList x=new StringList();”

Add “IntDict y=new IntDict();”

String[ ] or int [ ] are both basic arrays.  
The first holds several strings and the  
second holds several integer values.

## SKETCH: EXAMPLE 3C SIMPLE COUNTS

Starting from the same 311 dog example, we'll want to get the count of barking complaints by date to determine whether the season has a relationship to 311 reports. To do this we'll use three different types of 'arrays.' Arrays hold a list of data. It is possible to have an array of any type of data. Each piece of data in an array is identified by an index number representing its position in the array. StringLists and InterDicts hold not just an indexed number but additional information. To start with, declare these types at the top as global variables.



```

rowCountDM=manhattanTable.getRowCount();
println(rowCountDM);

//STUDENTS STOPPED HERE... WK 2 BEGINS WITH THE ALLDATES.APPEND SERIE
//NO LABELS, NO TEXT, NO SYMBOLS, NO SAVE OUT OR IMPORT LESSONS YET
//-----sum instances for display-----
for (TableRow dateRow : manhattanTable.rows()) {
  // get the values under the published date column
  String date = dateRow.getString(1);
  // add the dates to the StringList
  allDates.append(date);
}

// Get the count of how many times each string appears, copy it to
// this tells how the # of complaints/day for creating a timeline
dateCount = allDates.getTally();
dateCount.sortKeys();
println(dateCount);

// Split sorted strings and sorted counts into arrays
sortedDates = dateCount.keySet();
sortedCounts = dateCount.values();
}
//BACK ON SCHEDULE BUT NEED TO ADD IN THE SYMBOLIC @ THE BOTTOM

```

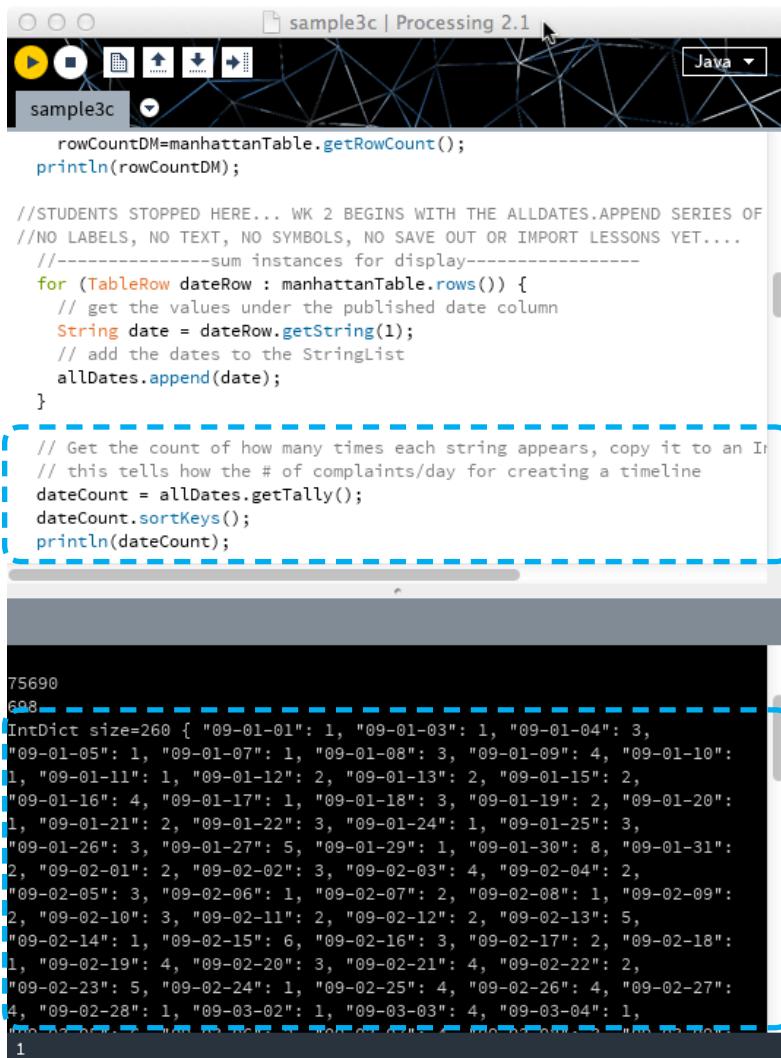
Using the “`for ( TableRow x: table.rows() )`” we can then read through all rows in the manhattan table in order to then pull the date value of each complaint by using `.getString(1)`. (There 1 is the column of complaint dates.)

`.append()` then adds each date value to the StringList `allDates`.

## SKETCH: EXAMPLE 3C **STRINGLIST OF EACH VALUE**

In order to count the number of complaints per date we will a) need to create a list of all unique dates, and b) then compare those values to tell which dates have multiple complaints.

For the first we will iterate through the table, using the ‘`TableRow`’ function, which is related to the ‘`Table`’ functions introduced last week. Once we’ve specified which table to iterate through in `TableRow`, we can then specify what values to retrieve according to column number using `.getString()` (or `.getInt` or `.getFloat`). Here we are then adding each value to the empty `StringList` declared in the last step.



```

sample3c | Processing 2.1
Java

rowCountDM=manhattanTable.getRowCount();
println(rowCountDM);

//STUDENTS STOPPED HERE... WK 2 BEGINS WITH THE ALLDATES.APPEND SERIES OF
//NO LABELS, NO TEXT, NO SYMBOLS, NO SAVE OUT OR IMPORT LESSONS YET....
//-----sum instances for display-----
for (TableRow dateRow : manhattanTable.rows()) {
    // get the values under the published date column
    String date = dateRow.getString(1);
    // add the dates to the StringList
    allDates.append(date);
}

// Get the count of how many times each string appears, copy it to an IntDict
// this tells how the # of complaints/day for creating a timeline
dateCount = allDates.getTally();
dateCount.sortKeys();
println(dateCount);

```

75690  
698  
IntDict size=260 { "09-01-01": 1, "09-01-03": 1, "09-01-04": 3,  
"09-01-05": 1, "09-01-07": 1, "09-01-08": 3, "09-01-09": 4, "09-01-10":  
1, "09-01-11": 1, "09-01-12": 2, "09-01-13": 2, "09-01-15": 2,  
"09-01-16": 4, "09-01-17": 1, "09-01-18": 3, "09-01-19": 2, "09-01-20":  
1, "09-01-21": 2, "09-01-22": 3, "09-01-24": 1, "09-01-25": 3,  
"09-01-26": 3, "09-01-27": 5, "09-01-29": 1, "09-01-30": 8, "09-01-31":  
2, "09-02-01": 2, "09-02-02": 3, "09-02-03": 4, "09-02-04": 2,  
"09-02-05": 3, "09-02-06": 1, "09-02-07": 2, "09-02-08": 1, "09-02-09":  
2, "09-02-10": 3, "09-02-11": 2, "09-02-12": 2, "09-02-13": 5,  
"09-02-14": 1, "09-02-15": 6, "09-02-16": 3, "09-02-17": 2, "09-02-18":  
1, "09-02-19": 4, "09-02-20": 3, "09-02-21": 4, "09-02-22": 2,  
"09-02-23": 5, "09-02-24": 1, "09-02-25": 4, "09-02-26": 4, "09-02-27":  
4, "09-02-28": 1, "09-03-02": 1, "09-03-03": 4, "09-03-04": 1,  
1

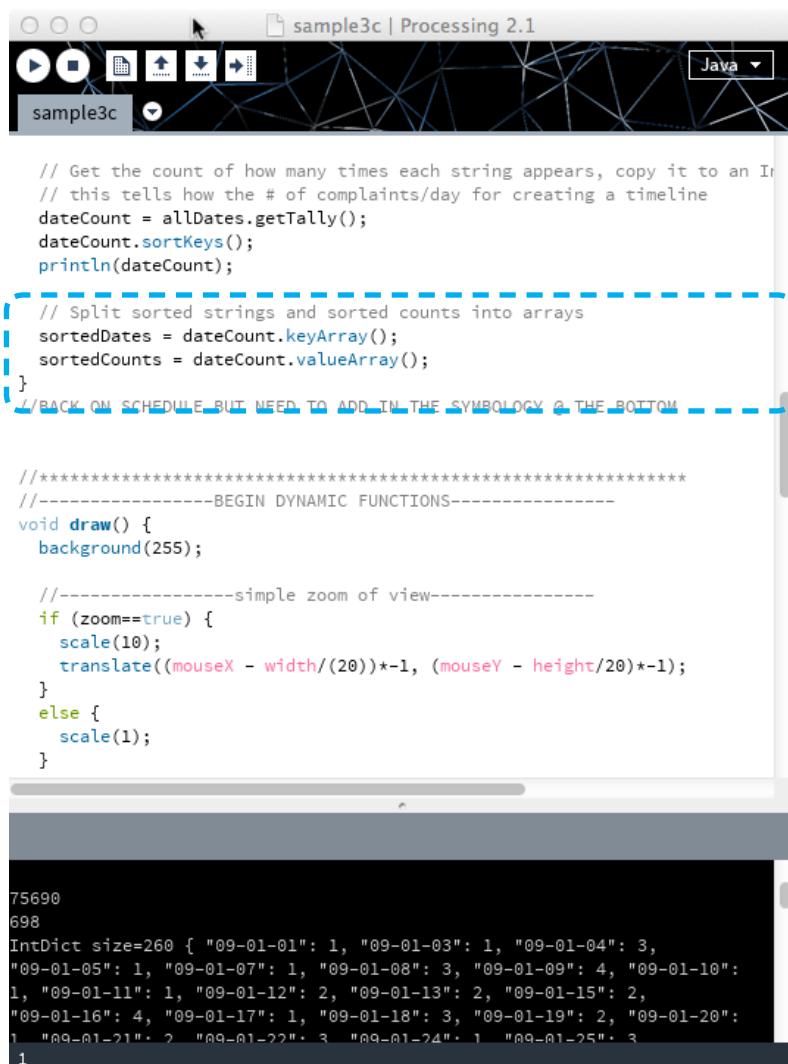
The function `.getTally()` compares the dates in the `StringList` and returns a condensed list of dates and the number of calls per date. To store these linked values date/#of calls we will use the `IntDict` `dateCount`.

To see the results look in the monitor below the code. (Println is used to produce that readout.)

## SKETCH: EXAMPLE 3C CONDENSED COUNTS

To compare the `StringList` internally, we'll use the `getTally()` function. See above for details of deployment and output as an `IntDict`.

In addition to the basic transfer of the condensed count of incidents/date to the `IntDict`, we are also using the function `.sortKeys()` in order to make sure those counts are in order. `Sort-Keys` shuffles the list to ascend from earliest/lowest date to highest/latest date recorded in the cvs table. Here, the sorted `IntDict` should start from Jan 1st, 2009 (with one complaint) and ascend toward December 31st, 2009.



```

// Get the count of how many times each string appears, copy it to an IntDict
// this tells how the # of complaints/day for creating a timeline
dateCount = allDates.getTally();
dateCount.sortKeys();
println(dateCount);

// Split sorted strings and sorted counts into arrays
sortedDates = dateCount.keySet();
sortedCounts = dateCount.value();
}

//BACK ON SCHEDULE BUT NEED TO ADD IN THE SYMBOLIC @ THE BOTTOM

```

```

//*****-----BEGIN DYNAMIC FUNCTIONS-----*****
void draw() {
  background(255);

  //-----simple zoom of view-----
  if (zoom==true) {
    scale(10);
    translate((mouseX - width/(20))*-1, (mouseY - height/20)*-1);
  }
  else {
    scale(1);
  }
}


```

```

75690
698
IntDict size=260 { "09-01-01": 1, "09-01-03": 1, "09-01-04": 3,
"09-01-05": 1, "09-01-07": 1, "09-01-08": 3, "09-01-09": 4, "09-01-10": 1,
"09-01-11": 1, "09-01-12": 2, "09-01-13": 2, "09-01-15": 2,
"09-01-16": 4, "09-01-17": 1, "09-01-18": 3, "09-01-19": 2, "09-01-20": 1
"09-01-21": 2 "09-01-22": 3 "09-01-24": 1 "09-01-25": 3
1

```

The function `.keyArray()` makes an array of those keys, i.e. dates. The function `.valueArray()` then holds the values, i.e. count of incidents, in a second array.

## SKETCH: EXAMPLE 3C INTDICT TO ARRAYS

After getting the paired values in IntDict `dateCount`, we can then hold those two lists, dates and # of incidents, in parallel arrays for ease of use.

As shown above, the arrays `String [ ] sortedDates` and `Int [ ] sortedCounts` (declared before) are thus defined by using the functions `keyArray()` and `valueArray()`. See the IntDict reference in [processing.org](http://processing.org) for more information.

Double check that your void `setup()` curly brackets are matched and you should be ready to then visualize this timeline of 311 incidents.



```

//BAR GRAPH OF VALUES
//borders for graph - left, right, top
int lb = 100;
int rb = 25;
int tb = 600;

int barWidth = ((width - lb)/sortedDates.length);
int graphHeight = (height - tb);
int maxValue = max(sortedCounts);

for (int i = 0; i <sortedCounts.length; i++) { // map data to height of graph
  int barHeight = (int) map(sortedCounts[i], 0, maxValue, 0, graphHeight);
  fill(cyan);
  //if mouse is within the boundary of a bar, highlight it
  //and display its data over the bar
  if (mouseX >= lb + (i*barWidth) && mouseX <= lb + (i*barWidth) + barWidth && mouseY >= height-barHeight) {
    fill(0);
    textAlign(CENTER);
    text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-10);
    text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-30);
  }
  // draw the bar
  rect(lb+(i*barWidth), height, barWidth, -barHeight);
}

//-----MOUSE interaction to trigger zoom-----

```

75690  
698  
TotalDate: 8120-269 [ "00-01-01" ] 1 [ "00-01-02" ] 1 [ "00-01-04" ] 3 [ "00-01-05" ] 1 [ "00-01-07" ] 1 [ "00-01-08" ] 2  
1

Add borders for your bar graph. Here left (lb), right (rb), and top(tb) are base on page size.

Set up variables for bar width, height, and max value based on the results of the arrays for sortedDates and sortedCounts. See below.

## SKETCH: EXAMPLE 3C SPACING YOUR TIMELINE

Once you've got your arrays, go to the bottom of void draw () and copy the code above.

Margins are self-explanatory, but let's look at internal spacing/sizing of the bars: 'barWidth' is based on the sketch width minus margins and divided by the total # of dates (bars). The maximum 'graphHeight' is based on height minus top margin. The 'maxValue' or max 'barHeight' is then equal to the highest count and will be mapped to be equivalent to the 'graphHeight'. This mapping is done as processing loops through all the sortedCount values, i.e. in the for loop. Thus, in this loop, each bar size is defined as a rectangle. The left margin and its place within the iteration define x. The height is the initial y. The bar's width is already defined as barWidth, and height is a negative of barHeight (stretching from bottom to top).



```

//BAR GRAPH OF VALUES
//borders for graph - left, right, top
int lb = 100;
int rb = 25;
int tb = 600;

int barWidth = ((width - lb)/sortedDates.length);
int graphHeight = (height - tb);
int maxValue = max(sortedCounts);

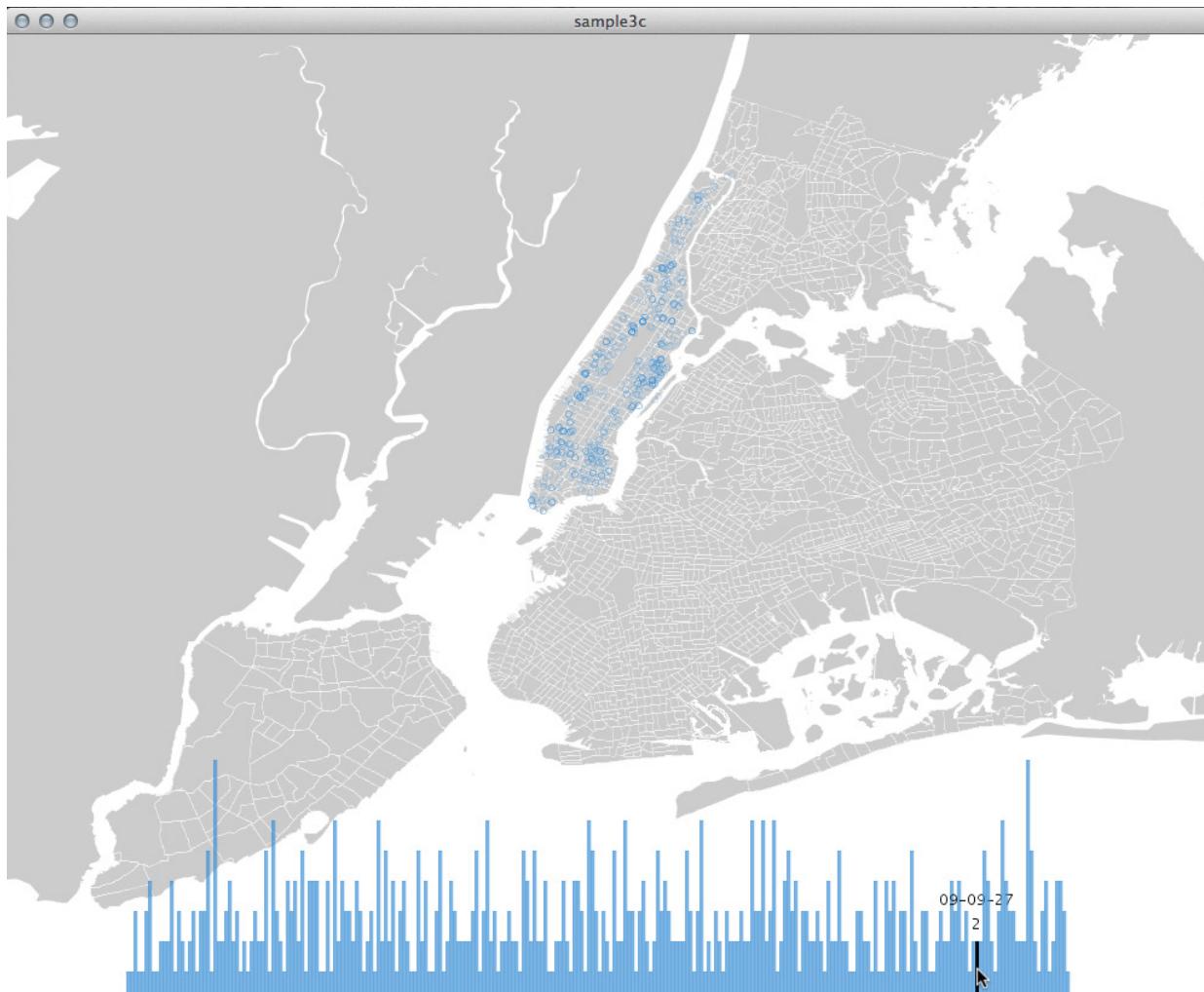
for (int i = 0; i < sortedCounts.length; i++) {      // map data to height of graph
    int barHeight = (int) map(sortedCounts[i], 0, maxValue, 0, graphHeight);
    fill(cyan);
    //if the mouse is within the x-boundary of a bar, highlight it
    //and display its data over the bar
    if (mouseX >= lb + (i*barWidth) && mouseX <= lb +(i*barWidth) + barWidth && mouseY >= height-barHeight) {
        fill(0);
        textAlign(CENTER);
        text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-10);
        text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-30);
    }
    // draw the bar
    rect(lb+(i*barWidth), height, barWidth, -barHeight);
// END ADDITION OF SYMBOLS
}
//-----MOUSE interaction to trigger zoom-----

```

\*interaction area:  
`mouseX >= value && mouseX <= value`  
`mouseY >= value && mouseY <= value`

\*`textAlign(CENTER, LEFT OR RIGHT);`  
\*`text (content, x position, y position);`

75690  
698  
Tn-Dict-07-08-09-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61-62-63-64-65-66-67-68-69-70-71-72-73-74-75-76-77-78-79-80-81-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99-100-101-102-103-104-105-106-107-108-109-110-111-112-113-114-115-116-117-118-119-120-121-122-123-124-125-126-127-128-129-130-131-132-133-134-135-136-137-138-139-140-141-142-143-144-145-146-147-148-149-150-151-152-153-154-155-156-157-158-159-160-161-162-163-164-165-166-167-168-169-170-171-172-173-174-175-176-177-178-179-180-181-182-183-184-185-186-187-188-189-190-191-192-193-194-195-196-197-198-199-200-201-202-203-204-205-206-207-208-209-210-211-212-213-214-215-216-217-218-219-220-221-222-223-224-225-226-227-228-229-230-231-232-233-234-235-236-237-238-239-240-241-242-243-244-245-246-247-248-249-250-251-252-253-254-255-256-257-258-259-260-261-262-263-264-265-266-267-268-269-270-271-272-273-274-275-276-277-278-279-280-281-282-283-284-285-286-287-288-289-290-291-292-293-294-295-296-297-298-299-299-300-301-302-303-304-305-306-307-308-309-3010-3011-3012-3013-3014-3015-3016-3017-3018-3019-3020-3021-3022-3023-3024-3025-3026-3027-3028-3029-3030-3031-3032-3033-3034-3035-3036-3037-3038-3039-3040-3041-3042-3043-3044-3045-3046-3047-3048-3049-3050-3051-3052-3053-3054-3055-3056-3057-3058-3059-3060-3061-3062-3063-3064-3065-3066-3067-3068-3069-3070-3071-3072-3073-3074-3075-3076-3077-3078-3079-3080-3081-3082-3083-3084-3085-3086-3087-3088-3089-3090-3091-3092-3093-3094-3095-3096-3097-3098-3099-30100-30101-30102-30103-30104-30105-30106-30107-30108-30109-30110-30111-30112-30113-30114-30115-30116-30117-30118-30119-30120-30121-30122-30123-30124-30125-30126-30127-30128-30129-30130-30131-30132-30133-30134-30135-30136-30137-30138-30139-30140-30141-30142-30143-30144-30145-30146-30147-30148-30149-30150-30151-30152-30153-30154-30155-30156-30157-30158-30159-30160-30161-30162-30163-30164-30165-30166-30167-30168-30169-30170-30171-30172-30173-30174-30175-30176-30177-30178-30179-30180-30181-30182-30183-30184-30185-30186-30187-30188-30189-30190-30191-30192-30193-30194-30195-30196-30197-30198-30199-30199-30200-30201-30202-30203-30204-30205-30206-30207-30208-30209-302010-302011-302012-302013-302014-302015-302016-302017-302018-302019-302020-302021-302022-302023-302024-302025-302026-302027-302028-302029-302030-302031-302032-302033-302034-302035-302036-302037-302038-302039-302040-302041-302042-302043-302044-302045-302046-302047-302048-302049-302050-302051-302052-302053-302054-302055-302056-302057-302058-302059-302060-302061-302062-302063-302064-302065-302066-302067-302068-302069-302070-302071-302072-302073-302074-302075-302076-302077-302078-302079-302080-302081-302082-302083-302084-302085-302086-302087-302088-302089-302090-302091-302092-302093-302094-302095-302096-302097-302098-302099-302099-302100-302101-302102-302103-302104-302105-302106-302107-302108-302109-3021010-3021011-3021012-3021013-3021014-3021015-3021016-3021017-3021018-3021019-3021020-3021021-3021022-3021023-3021024-3021025-3021026-3021027-3021028-3021029-3021030-3021031-3021032-3021033-3021034-3021035-3021036-3021037-3021038-3021039-3021040-3021041-3021042-3021043-3021044-3021045-3021046-3021047-3021048-3021049-3021050-3021051-3021052-3021053-3021054-3021055-3021056-3021057-3021058-3021059-3021060-3021061-3021062-3021063-3021064-3021065-3021066-3021067-3021068-3021069-3021070-3021071-3021072-3021073-3021074-3021075-3021076-3021077-3021078-3021079-3021080-3021081-3021082-3021083-3021084-3021085-3021086-3021087-3021088-3021089-3021090-3021091-3021092-3021093-3021094-3021095-3021096-3021097-3021098-3021099-3021099-3021100-3021101-3021102-3021103-3021104-3021105-3021106-3021107-3021108-3021109-30211010-30211011-30211012-30211013-30211014-30211015-30211016-30211017-30211018-30211019-30211020-30211021-30211022-30211023-30211024-30211025-30211026-30211027-30211028-30211029-30211030-30211031-30211032-30211033-30211034-30211035-30211036-30211037-30211038-30211039-30211040-30211041-30211042-30211043-30211044-30211045-30211046-30211047-30211048-30211049-30211050-30211051-30211052-30211053-30211054-30211055-30211056-30211057-30211058-30211059-30211060-30211061-30211062-30211063-30211064-30211065-30211066-30211067-30211068-30211069-30211070-30211071-30211072-30211073-30211074-30211075-30211076-30211077-30211078-30211079-30211080-30211081-30211082-30211083-30211084-30211085-30211086-30211087-30211088-30211089-30211090-30211091-30211092-30211093-30211094-30211095-30211096-30211097-30211098-30211099-30211099-30211100-30211101-30211102-30211103-30211104-30211105-30211106-30211107-30211108-30211109-302111010-302111011-302111012-302111013-302111014-302111015-302111016-302111017-302111018-302111019-302111020-302111021-302111022-302111023-302111024-302111025-302111026-302111027-302111028-302111029-302111030-302111031-302111032-302111033-302111034-302111035-302111036-302111037-302111038-302111039-302111040-302111041-302111042-302111043-302111044-302111045-302111046-302111047-302111048-302111049-302111050-302111051-302111052-302111053-302111054-302111055-302111056-302111057-302111058-302111059-302111060-302111061-302111062-302111063-302111064-302111065-302111066-302111067-302111068-302111069-302111070-302111071-302111072-302111073-302111074-302111075-302111076-302111077-302111078-302111079-302111080-302111081-302111082-302111083-302111084-302111085-302111086-302111087-302111088-302111089-302111090-302111091-302111092-302111093-302111094-302111095-302111096-302111097-302111098-302111099-302111099-302111100-302111101-302111102-302111103-302111104-302111105-302111106-302111107-302111108-302111109-302111110-302111111-302111112-302111113-302111114-302111115-302111116-302111117-302111118-302111119-302111120-302111121-302111122-302111123-302111124-302111125-302111126-302111127-302111128-302111129-302111130-302111131-302111132-302111133-302111134-302111135-302111136-302111137-302111138-302111139-302111140-302111141-302111142-302111143-302111144-302111145-302111146-302111147-302111148-302111149-302111150-302111151-302111152-302111153-302111154-302111155-302111156-302111157-302111158-302111159-302111160-302111161-302111162-302111163-302111164-302111165-302111166-302111167-302111168-302111169-302111170-302111171-302111172-302111173-302111174-302111175-302111176-302111177-302111178-302111179-302111180-302111181-302111182-302111183-302111184-302111185-302111186-302111187-302111188-302111189-302111190-302111191-302111192-302111193-302111194-302111195-302111196-302111197-302111198-302111199-302111199-302111200-302111201-302111202-302111203-302111204-302111205-302111206-302111207-302111208-302111209-302111210-302111211-302111212-302111213-302111214-302111215-302111216-302111217-302111218-302111219-302111220-302111221-302111222-302111223-302111224-302111225-302111226-302111227-302111228-302111229-302111230-302111231-302111232-302111233-302111234-302111235-302111236-302111237-302111238-302111239-302111240-302111241-302111242-302111243-302111244-302111245-302111246-302111247-302111248-302111249-302111250-302111251-302111252-302111253-302111254-302111255-302111256-302111257-302111258-302111259-302111260-302111261-302111262-302111263-302111264-302111265-302111266-302111267-302111268-302111269-302111270-302111271-302111272-302111273-302111274-302111275-302111276-302111277-302111278-302111279-302111280-302111281-302111282-302111283-302111284-302111285-302111286-302111287-302111288-302111289-302111290-302111291-302111292-302111293-302111294-302111295-302111296-302111297-302111298-302111299-302111299-302111300-302111301-302111302-302111303-302111304-302111305-302111306-302111307-302111308-302111309-302111310-302111311-302111312-302111313-302111314-302111315-302111316-302111317-302111318-302111319-302111320-302111321-302111322-302111323-302111324-302111325-302111326-302111327-302111328-302111329-302111330-302111331-302111332-302111333-302111334-302111335-302111336-302111337-302111338-302111339-302111340-302111341-302111342-302111343-302111344-302111345-302111346-302111347-302111348-302111349-302111350-302111351-302111352-302111353-302111354-302111355-302111356-302111357-302111358-302111359-302111360-302111361-302111362-302111363-302111364-302111365-302111366-302111367-302111368-302111369-302111370-302111371-302111372-302111373-302111374-302111375-302111376-302111377-302111378-302111379-302111380-302111381-302111382-302111383-302111384-302111385-302111386-302111387-302111388-302111389-302111390-302111391-302111392-302111393-302111394-302111395-302111396-302111397-302111398-302111399-302111399-302111400-302111401-302111402-302111403-302111404-302111405-302111406-302111407-302111408-302111409-302111410-302111411-302111412-302111413-302111414-302111415-302111416-302111417-302111418-302111419-302111420-302111421-302111422-302111423-302111424-302111425-302111426-302111427-302111428-302111429-302111430-302111431-302111432-302111433-302111434-302111435-302111436-302111437-302111438-302111439-302111440-302111441-302111442-302111443-302111444-302111445-302111446-302111447-302111448-302111449-302111450-302111451-302111452-302111453-302111454-302111455-302111456-302111457-302111458-302111459-302111460-302111461-302111462-302111463-302111464-302111465-302111466-302111467-302111468-302111469-302111470-302111471-302111472-302111473-302111474-302111475-302111476-302111477-302111478-302111479-302111480-302111481-302111482-302111483-302111484-302111485-302111486-302111487-302111488-302111489-302111490-302111491-302111492-302111493-302111494-302111495-302111496-302111497-302111498-302111499-302111499-302111500-302111501-302111502-302111503-302111504-302111505-302111506-302111507-302111508-302111509-302111510-302111511-302111512-302111513-302111514-302111515-302111516-302111517-302111518-302111519-302111520-302111521-302111522-302111523-302111524-302111525-302111526-302111527-302111528-302111529-302111530-302111531-302111532-302111533-302111534-302111535-302111536-302111537-302111538-302111539-302111540-302111541-302111542-302111543-302111544-302111545-302111546-302111547-302111548-302111549-302111550-302111551-302111552-302111553-302111554-302111555-302111556-302111557-302111558-302111559-302111560-302111561-302111562-302111563-302111564-302111565-302111566-302111567-302111568-302111569-302111570-302111571-302111572-302111573-302111574-302111575-302111576-302111577-302111578-302111579-302111580-302111581-302111582-302111583-302111584-302111585-302111586-302111587-302111588-302111589-302111590-302111591-302111592-302111593-302111594-302111595-302111596-302111597-302111598-302111599-302111599-302111600-302111601-302111602-302111603-302111604-302111605-302111606-302111607-302111608-302111609-302111610-302111611-302111612-302111613-302111614-302111615



## SKETCH: EXAMPLE 3C **INTERACTIVE LABELS**

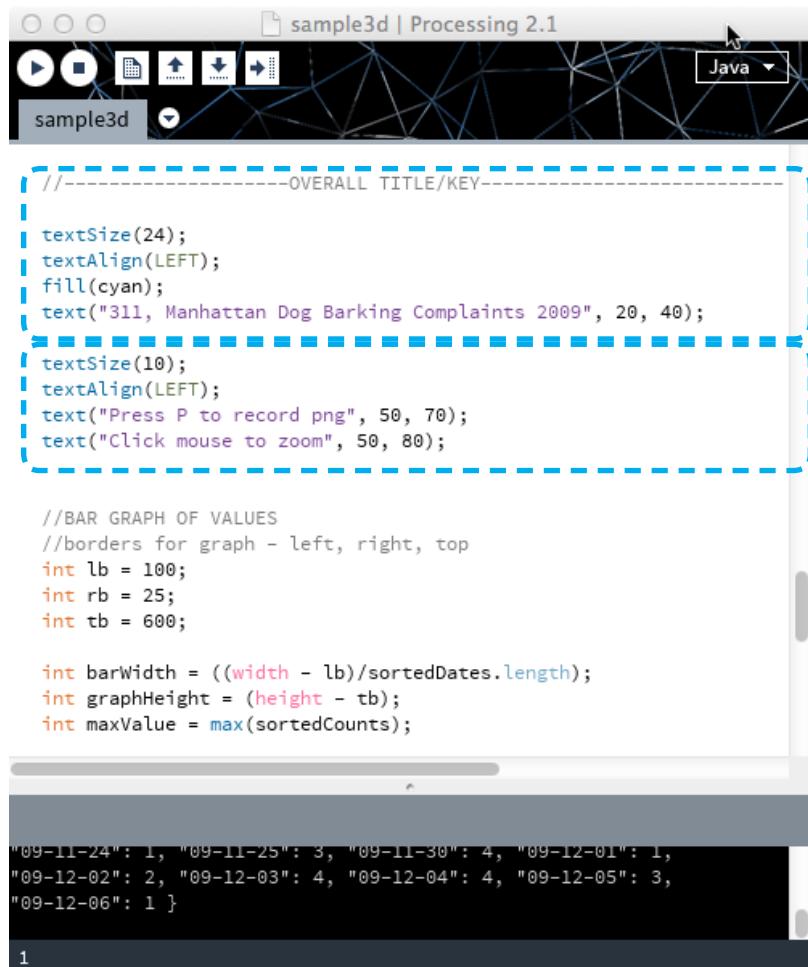
As seen at the bottom right, a hoovering mouse triggers a black fill and the display of the date and # of complaints.

Oddly enough, there does not seem to be much correspondence between the season and number of dog complaints. (Perhaps enough of NYC is air-conditioned and thus windows are always closed to street noise?) The fairly regular peaks do however suggest that some sort of weekly cycle might be at work.

# WK 2

## LEGIBLE & INTERACTIVE INFO

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



The screenshot shows the Processing 2.1 interface with a 3D wireframe visualization. The code editor contains the following Java code:

```
//-----OVERALL TITLE/KEY-----
textSize(24);
textAlign(LEFT);
fill(cyan);
text("311, Manhattan Dog Barking Complaints 2009", 20, 40);

textSize(10);
textAlign(LEFT);
text("Press P to record png", 50, 70);
text("Click mouse to zoom", 50, 80);

//BAR GRAPH OF VALUES
//borders for graph - left, right, top
int lb = 100;
int rb = 25;
int tb = 600;

int barWidth = ((width - lb)/sortedDates.length);
int graphHeight = (height - tb);
int maxValue = max(sortedCounts);

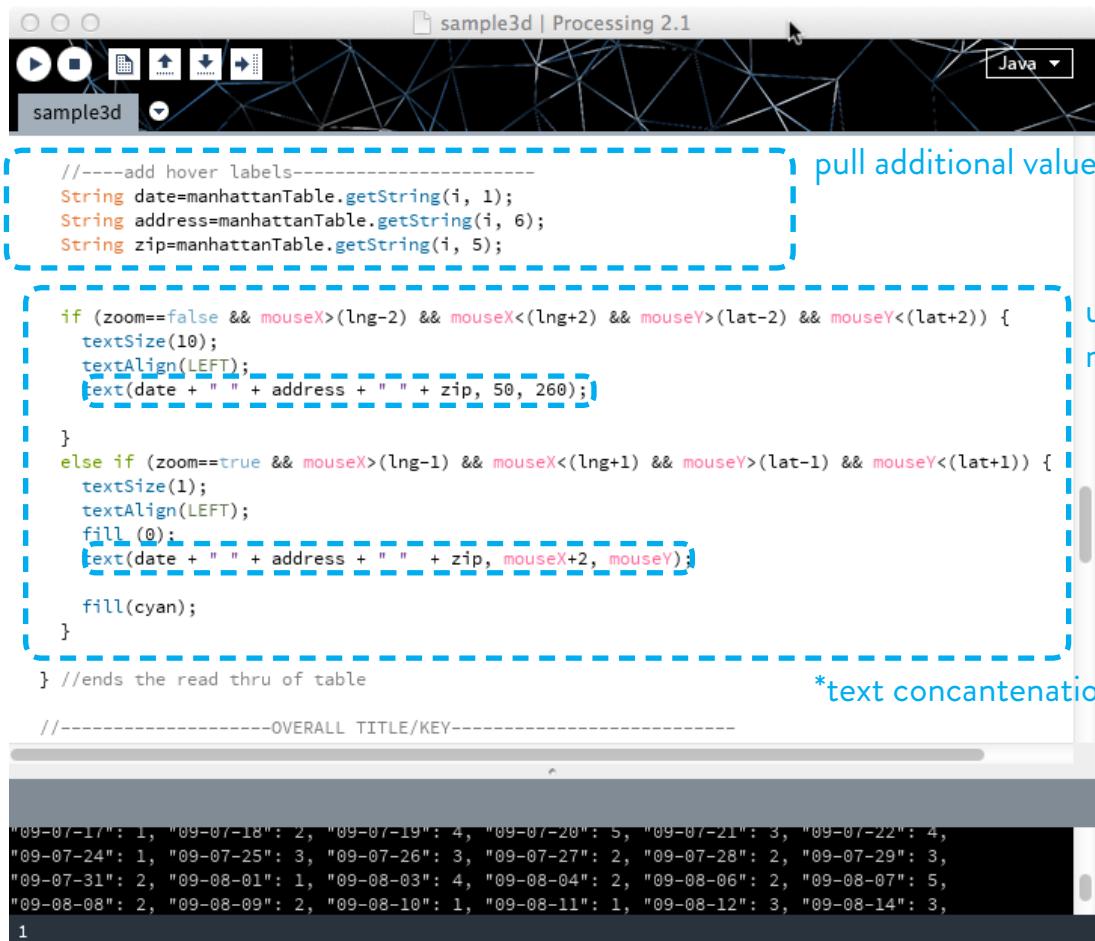
"09-11-24": 1, "09-11-25": 3, "09-11-30": 4, "09-12-01": 1,
"09-12-02": 2, "09-12-03": 4, "09-12-04": 4, "09-12-05": 3,
"09-12-06": 1 }
```

Annotations on the right side of the code editor:

- overall title
  - \* `textSize(font size in pts);`
  - \* `textAlign(); & text (content, x, y);`
- subtitles
  - redefine size and add infor for future interactions

## SKETCH: EXAMPLE 3D **LABELS**

As seen in the last step, labels are fairly simple. To add a title to our drawing, find a space above the bar graph, but outside the main loop that is doing geolocation. Copy the text above.



```

//----add hover labels-----
String date=manhattanTable.getString(i, 1);
String address=manhattanTable.getString(i, 6);
String zip=manhattanTable.getString(i, 5);

if (zoom==false && mouseX>(lng-2) && mouseX<(lng+2) && mouseY>(lat-2) && mouseY<(lat+2)) {
    textSize(10);
    textAlign(LEFT);
    text(date + " " + address + " " + zip, 50, 260);
}

else if (zoom==true && mouseX>(lng-1) && mouseX<(lng+1) && mouseY>(lat-1) && mouseY<(lat+1)) {
    textSize(1);
    textAlign(LEFT);
    fill (0);
    text(date + " " + address + " " + zip, mouseX+2, mouseY);
    fill(cyan);
}

} //ends the read thru of table

//-----OVERALL TITLE/KEY-----

```

pull additional value from the original table

using conditionals for mouse position & zoom

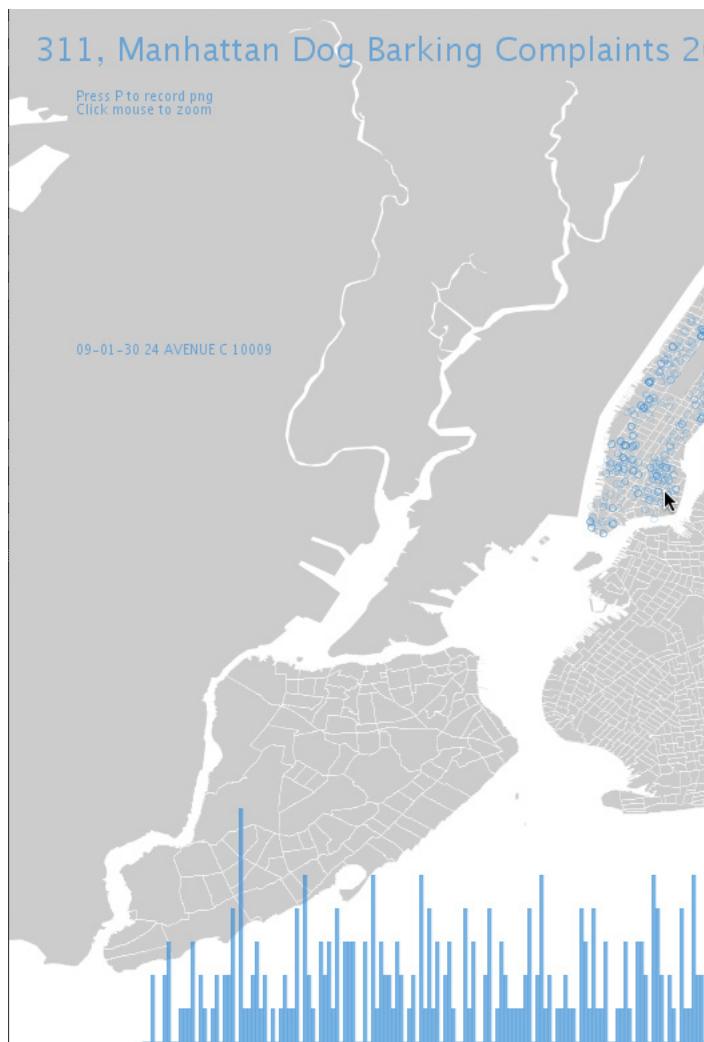
\*text concatenation with +

1

## SKETCH: EXAMPLE 3D **MORE INTERACTIONS**

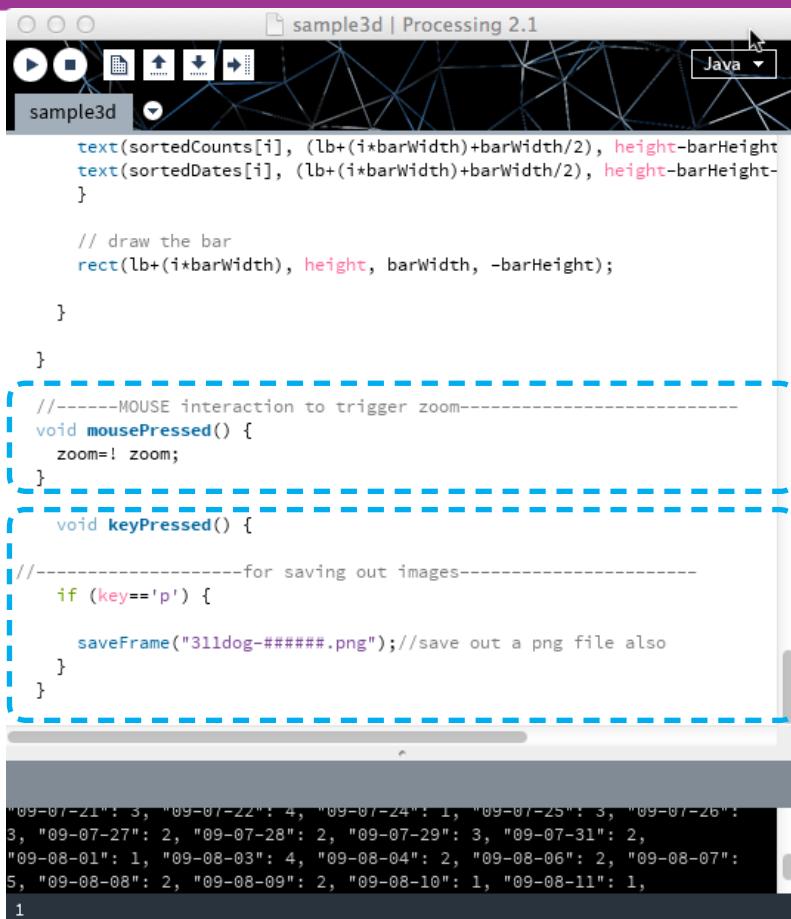
For hoovering labels on the main map, we'll do something similar to the bar-graph area.

First, within the loop that is reading the original table for geolocation, add variables to grab additional values from each row of the table: date, address, and zipcode of the complaint. Then, we'll use an 'if/else if' statement to check the position of the mouse and the zoom boolean value. That general process should be familiar by now. The thing to note here is that the text for output is concatenated, i.e. composed of several different variables strung together. In processing, this is done by using the +. Hence when we specify "text(date + " " + address + " " + zip, and so on) this is read as, 'write the date, leave a space (" "), write the address, leave another space' and so on.



## SKETCH: EXAMPLE 3D **INTERACTION**

The results should look like the above screen shots.



```

sample3d | Processing 2.1
Java

text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), height-barHeight
text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-
}

// draw the bar
rect(lb+(i*barWidth), height, barWidth, -barHeight);

}

}

//-----MOUSE interaction to trigger zoom-----
void mousePressed() {
  zoom+=1;
}

void keyPressed() {

//-----for saving out images-----
if (key=='p') {

  saveFrame("311dog-#####.png");//save out a png file also
}
}

"09-07-21": 3, "09-07-22": 4, "09-07-24": 1, "09-07-25": 3, "09-07-26": 3, "09-07-27": 2, "09-07-28": 2, "09-07-29": 3, "09-07-31": 2, "09-08-01": 1, "09-08-03": 4, "09-08-04": 2, "09-08-06": 2, "09-08-07": 5, "09-08-08": 2, "09-08-09": 2, "09-08-10": 1, "09-08-11": 1,
1

```

older void mousePressed (){ }

\*void keyPressed (){ }

\*conditional: if (key=='x'){ }

\*saveFrame("folder/name-#####.png");  
works for png, jpg, tif, tga files

## SKETCH: EXAMPLE 3D **KEY OPTIONS-SAVEFRAME**

In addition to mouse-hoovering interactions and the function `mousePressed()`, we can specify finer interactions options through keyboard use.

In the sketch above, add another function at the very end of the code. Built into processing, `keyPressed()` senses any keyboard interaction, from letters, numbers, punctuation, etc. Here, it is listening for a keyboard 'event' and then the 'if' conditional compares whether that keyboard interaction is the specified trigger 'p'. If so, the sketch runs the `saveFrame()` function to save out an image of the current frame. In this case, it will save "311dog-whatever number tags that frame.png".

# WK 2-OTHER INTERACTIONS-SAVING OPTIONS

## EXAMPLE 3D **SAVING OPTIONS**

### *MULTIPLE FRAMES:*

```
save("imagename.jpg");
```

*As the drawing loops through it will auto-save an image. Because there is no framecount to this procedure, processing is likely to save over and over on top of an original file... not recommended for general use.*

```
saveFrame("imagename-#####.jpg");
```

*Will save images out with their frame number. For discrete images embedded within either key or mouse interaction functions. If used alone, within a dynamic (draw) function, it will save a new numbered frame each time through the program. This can be handy for generative graphics and animations.*

### *making mov. animation files*

*In addition to the use of save frame above, you can easily gather files for a movie by a) saving them into a specific folder:*

```
saveFrame ("img/image-#####.png");
```

*and then selecting TOOLS/MOVIE MAKER from processing's menu bar.*

*From there you can specify that folder of images, the frame rate of your animation, an alteration of its' ratio and accompanying sound files.*

### *pdf library use*

*Under the sketch menu of processing you can import the pdf library to save out pdf vector files. There are several different options of how to save single and multiple images at <http://processing.org/reference/libraries/pdf/index.html>*

*I prefer to save on interactions, as in the keypressed example. To do this, distribute this code accordingly:*

```
global area      import processing.pdf.*; boolean record=false;
in void draw (top)  if (record) { beginRecord(PDF, "frame-####.pdf");}
draw some stuff in between
in void draw (bottom)  if (record) { endRecord(); record = false; }
as interactive function  void mousePressed() {record = true;}
condense for display, type as normal with line-breaks
```

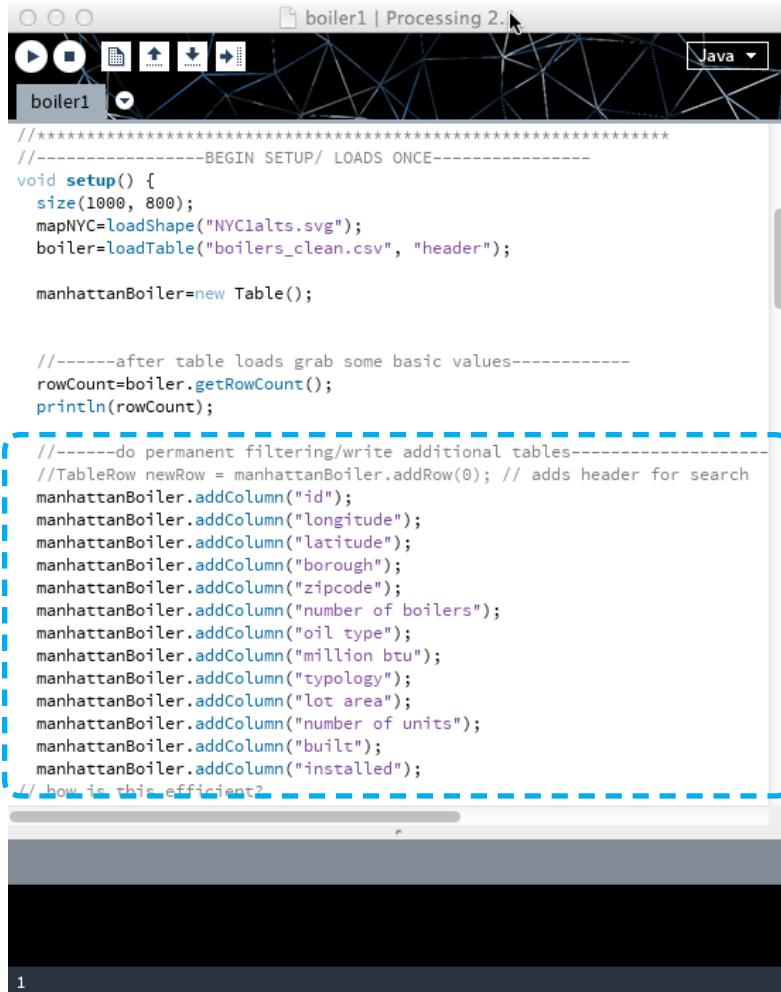
# WK 2

## MORE ADVANCED INFO & SYMBOLS

- NYC Open data for Boilers and Oil Use
- quantitative thresholds and symbols
- scaled keys
- more time-based variables (addition/interaction)
- secondary data- typologies and oil use (zipcode)
- representational forms- data roses
- combining modulo and raw BTU quantities
- visually cleaning your geographic ‘dashboard’

This example builds on familiar steps and familiar functions from the 311 example. We will take advantage of more initial data for display options and secondary data display types. When we use old function, I will refer back to their original definition by page number but will not repeat their introduction.

The code in example boiler.pde thus starts approximately where we've left off with the 311 example. The original, pre-cleaned boiler data is available at <https://nycopendata.socrata.com/Housing-Development/Oil-Boilers-Detailed-Fuel-Consumption-and-Building/jfzu-yy6n>



```
//*****BEGIN SETUP/ LOADS ONCE*****
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYCaltz.svg");
  boiler=loadTable("boilers_clean.csv", "header");

  manhattanBoiler=new Table();

  //-----after table loads grab some basic values-----
  rowCount=boiler.getRowCount();
  println(rowCount);

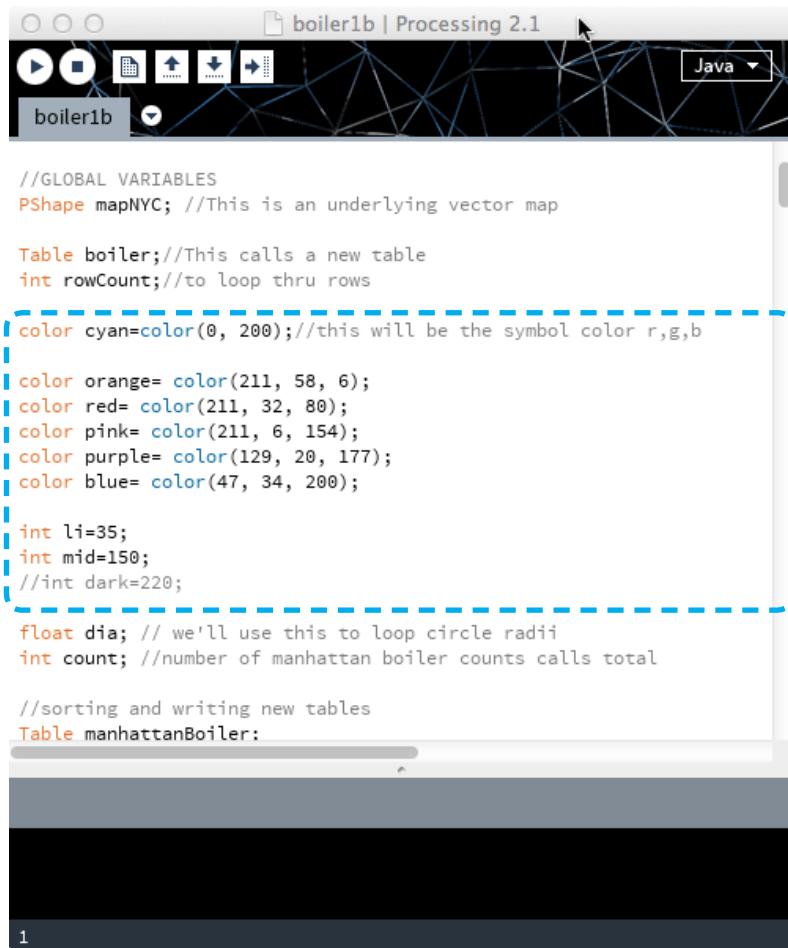
  //----do permanent filtering/write additional tables-----
  //TableRow newRow = manhattanBoiler.addRow(0); // adds header for search
  manhattanBoiler.addColumn("id");
  manhattanBoiler.addColumn("longitude");
  manhattanBoiler.addColumn("latitude");
  manhattanBoiler.addColumn("borough");
  manhattanBoiler.addColumn("zipcode");
  manhattanBoiler.addColumn("number of boilers");
  manhattanBoiler.addColumn("oil type");
  manhattanBoiler.addColumn("million btu");
  manhattanBoiler.addColumn("typology");
  manhattanBoiler.addColumn("lot area");
  manhattanBoiler.addColumn("number of units");
  manhattanBoiler.addColumn("built");
  manhattanBoiler.addColumn("installed");
  // how is this efficient?
```

Within setup, we've done a bit more prep:

Adds headers to the new table, before adding the actual information that we'll filter to write the manhattanBoiler table itself.

## SKETCH: BOILER 1 **ANOTHER NYC EXAMPLE**

If you open the boiler1.pde it should seem familiar in structure after the 311 manhattan dog barking example. A few additions include adding a row for headers into a saved table. This will enable easier reader of data.



```

//GLOBAL VARIABLES
PShape mapNYC; //This is an underlying vector map

Table boiler; //This calls a new table
int rowCount; //to loop thru rows

color cyan=color(0, 200); //this will be the symbol color r,g,b

color orange= color(211, 58, 6);
color red= color(211, 32, 80);
color pink= color(211, 6, 154);
color purple= color(129, 20, 177);
color blue= color(47, 34, 200);

int li=35;
int mid=150;
//int dark=220;

float dia; // we'll use this to loop circle radii
int count; //number of manhattan boiler counts calls total

//sorting and writing new tables
Table manhattanBoiler:

```

Addition of several standard colors for use across the map symbols and graphic key

## SKETCH: BOILER 1B **COLORS FOR KEY**

This boiler data also contains several other types of info:  
 the amount of fuel consumed by each building annually (in million BTU),  
 the age of building that the oil burners are installed in,  
 the age of furance burning #4 or #6 oil,  
 which type of oil is being burned,  
 and, finally,  
 the general type of building, its lot area and number of units heated.

Let's visualize those, to start with, using color...



```

boiler1b | Processing 2.1
boiler1b | Processing 2.1

//-----reading thru data table for-----
for (int i=0;i<rowCountMB;i++) {// loops through all rows!

//-----geographic location-----
float x=manhattanBoiler.getFloat(i, "longitude");// for each row, gra
float y=manhattanBoiler.getFloat(i, "latitude");

float blBTU=manhattanBoiler.getFloat(i, "million btu");//for diameter

/* map lat/longitude note:
(x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

float lng = map(x, -74.3, -73.65, 0, width);
float lat = map(y, 40.95, 40.45, 0, height);

noFill();//fill
dia=blBTU*.5;//diameter of symbol
}

//-----the symbolization-----
if (blBTU<=5) {
  fill(blue, li);
  stroke(blue, mid);
}
if (blBTU>5 && blBTU<=25) {
  fill(purple, li);
  stroke(purple, mid);
}
if (blBTU>25 && blBTU<=50) {
  fill(pink, li);
  stroke(pink, mid);
}
if (blBTU>50 && blBTU<=100) {
  fill(red, li);
  stroke(red, mid);
}
if (blBTU>100 && blBTU<=1000) {
  fill(orange, li);
  stroke(orange, mid);
}

//noStroke();
//-----the symbol-----
ellipse(lng, lat, dia, dia);// draws symbol
}

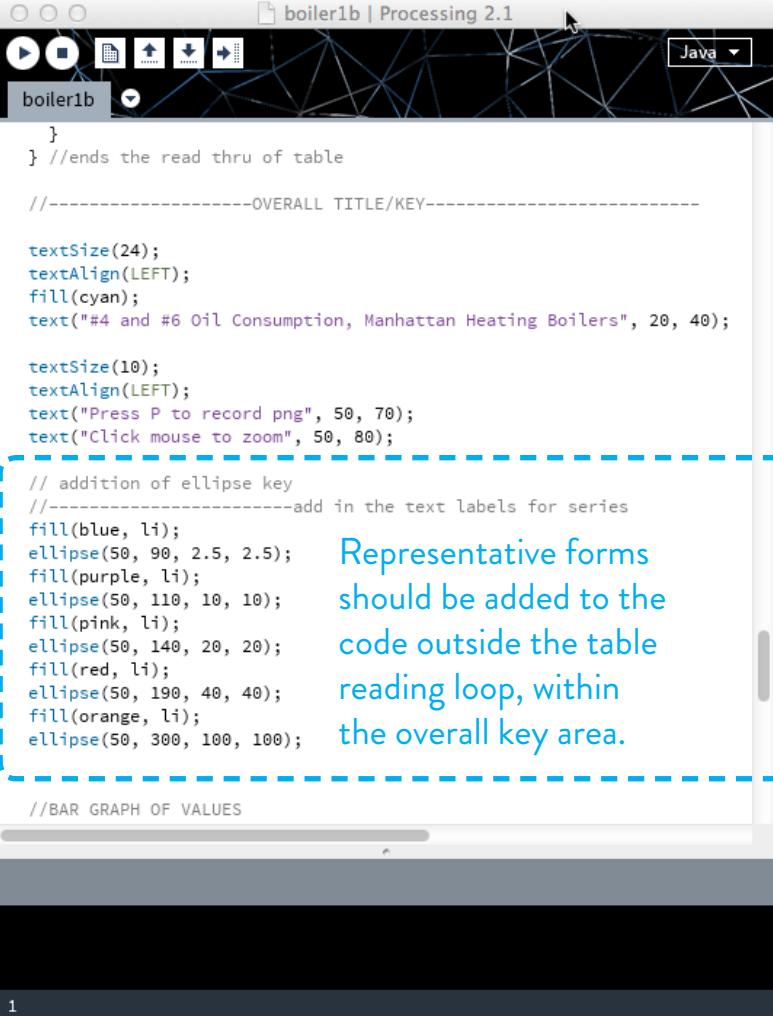
```

Reading table energy use to 'bIBTU' and then setting 'if' thresholds to color fill and stroke based on 'bIBTU' for ellipse symbol

## SKETCH: BOILER 1B VAR SIZE, COLORS BY THRESHOLDS

Based on the existing table-reading loop in void draw( ), adding size variation for symbols and color is easy to do.

Create a variable to hold the info/row on energy used annually as 'float blBTU' and extract that info with a .getFloat( ) function. Then set your ellipse diameters, fill, and stroke according to a relationship with blBTU. For diameter, we can use a variable of one-half times blBTU. For stroke and fill, we'll set up thresholds based on blBTU values by using the above 'if' statements.



```
boiler1b | Processing 2.1
Java

boiler1b

}

} //ends the read thru of table

//-----OVERALL TITLE/KEY-----

textSize(24);
textAlign(LEFT);
fill(cyan);
text("#4 and #6 Oil Consumption, Manhattan Heating Boilers", 20, 40);

textSize(10);
textAlign(LEFT);
text("Press P to record png", 50, 70);
text("Click mouse to zoom", 50, 80);

// addition of ellipse key
//-----add in the text labels for series
fill(blue, li);
ellipse(50, 90, 2.5, 2.5);
fill(purple, li);
ellipse(50, 110, 10, 10);
fill(pink, li);
ellipse(50, 140, 20, 20);
fill(red, li);
ellipse(50, 190, 40, 40);
fill(orange, li);
ellipse(50, 300, 100, 100);

//BAR GRAPH OF VALUES
1
```

(We'll add the text in a few steps.)

Representative forms should be added to the code outside the table reading loop, within the overall key area.

## SKETCH: BOILER 1B **SYMBOL KEY FORMS**

After adjusting the actual symbols, we should then add ellipses that approximate those fill values and sizes to the overall key.

```

textAlign(CENTER);
text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), hei
text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), hei
}

// draw the bar
rect(lb+(i*barWidth), height-of, barWidth, -barHeight-of
}

//-----MOUSE interaction to trigger zoom-----
void mousePressed() {
  zoom=! zoom;
}

void keyPressed() {
  //-----the on/off of the map
  if (key=='m') {
    m = !m;//inverts the boolean
  }
}

add a 'boolean m=false;' as a global variable
activate under keyPressed-----ng out images-----
if (key=='p') {

  saveFrame("boiler-#####.png");//save out a png file als
}

}

void draw() {
  background(255);

  //crosshairs
  line(0, height/2, width, height/2);
  line(width/2, 0, width/2, height);

  //-----simple zoom of view-----
  if (zoom==true) {
    scale(10);
    translate((mouseX - width/(20))*-1, (mouseY - height/20)*-1);
  }
  else {
    scale(1);
  }

  if (m==true) {
    shape(mapNYC, 0, 0, width, height);
  }

  noFill();
  stroke(cyan);
  //strokeWeight(5); comment out for smaller stroke
  strokeWeight(.25);

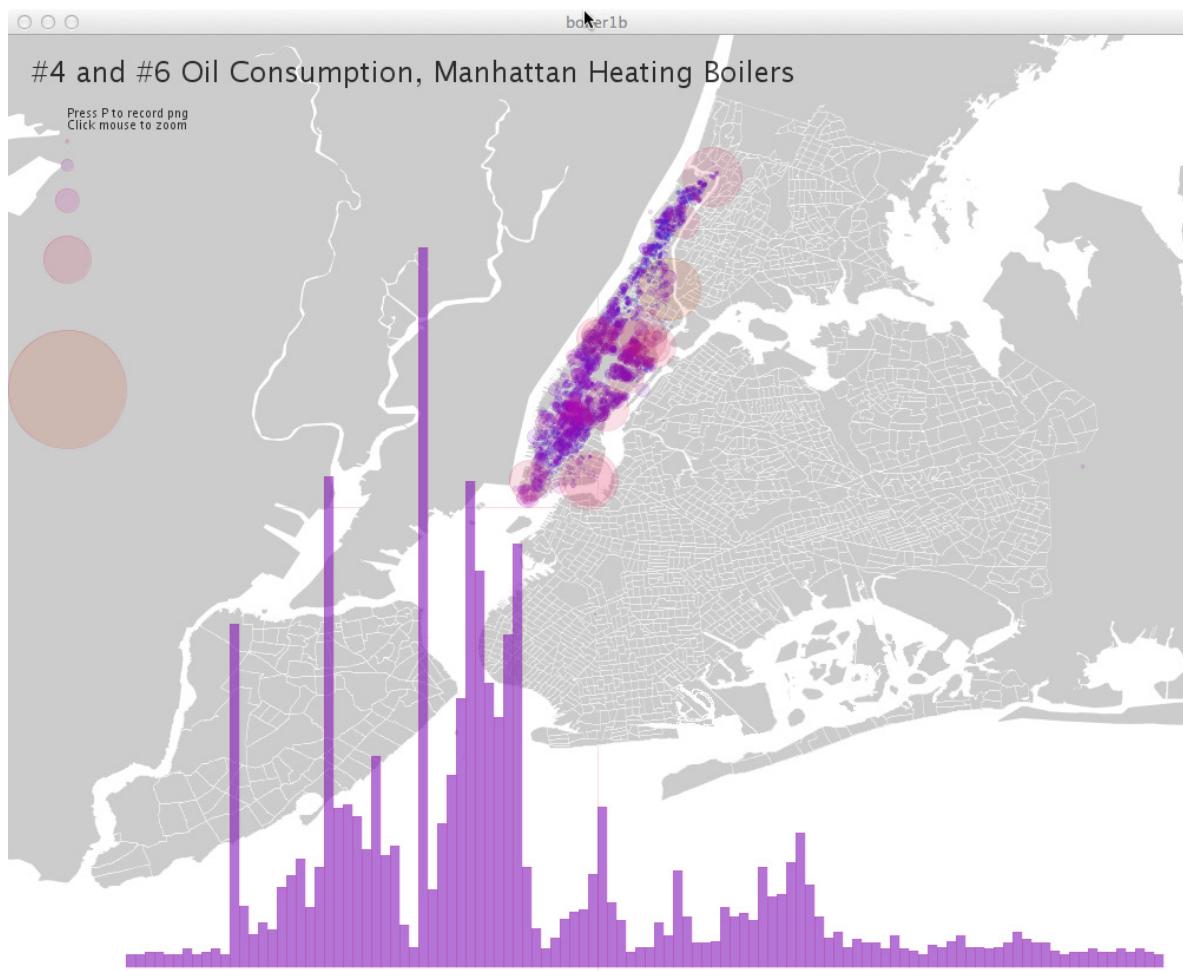
  //-----reading thru data table for-----
  for (int i=0;i<rowCountMB;i++) {// loops through all rows!
    //-----geographic location-----
    float x=manhattanBoiler.getFloat(i, "longitude");// for each row, gra
}

```

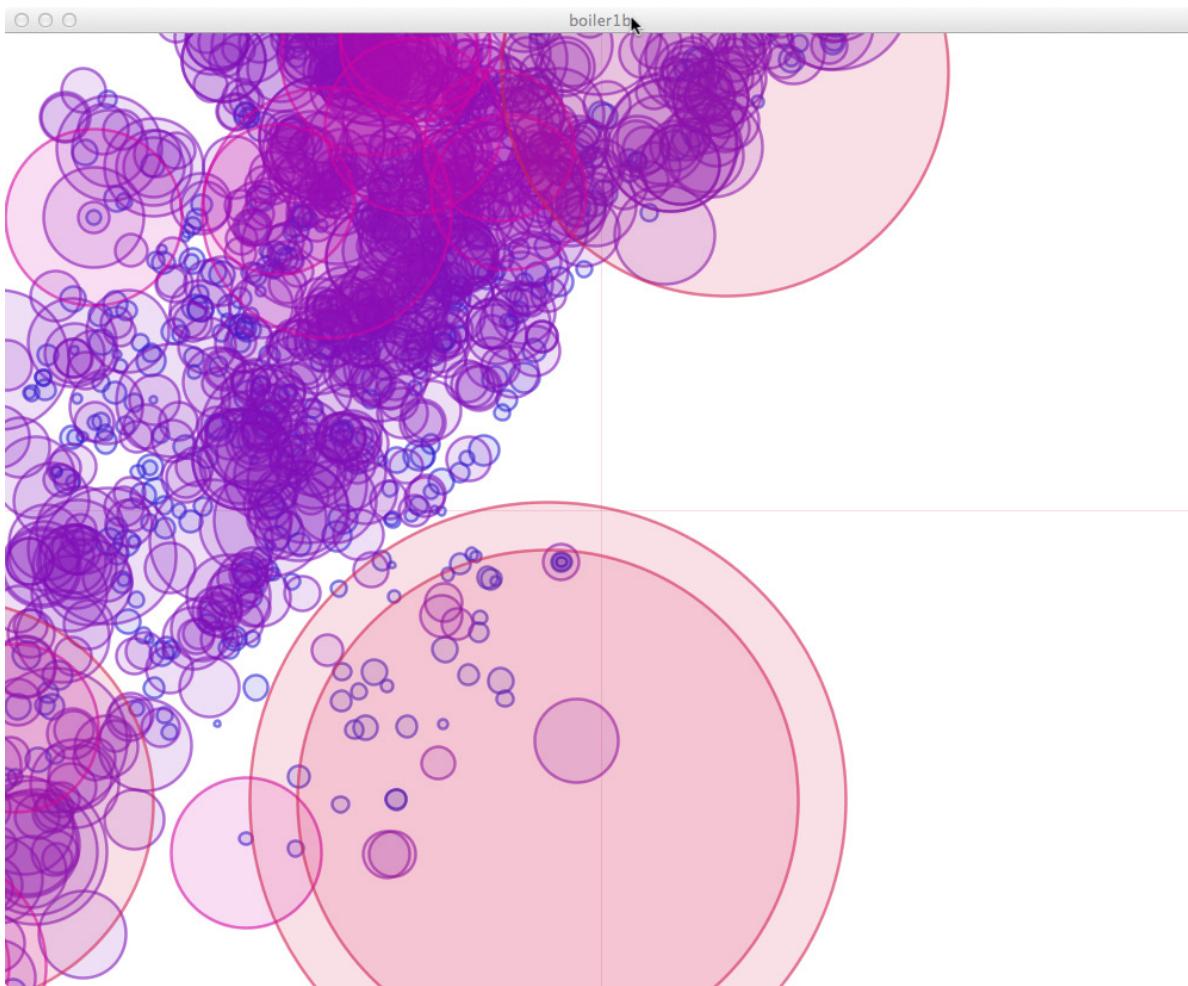
under the zoom area  
add an 'if' statement  
around the shape() function.

## SKETCH: BOILER 1B BACKGROUND OPTIONS

In addition, we'll add another boolean `m` to allow the background map to turn on and off. Use the same type of conditional 'if' statement with `keyPressed` to turn this boolean on and off (see p 50, wk1). Once the interaction is set up, go up to the top of `void draw()` and find the '`shape(mapNYC...);`' line. Add the above 'if' statement.



SKETCH: BOILER 1B **CURRENT-MAP ON**



SKETCH: BOILER 1B **CURRENT-MAP OFF / ZOOM**



```

boiler1c | Processing 2.1
boiler1c

text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-30-of);
text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-20-of);

// add the if/then that will reinitate the series of row reads...
for (TableRow builtRow : manhattanBoiler.findRows(sortedDates[i], "built")) {

    //-----geographic location-----
    float xB=builtRow.getFloat("longitude");// for each row, grabs the float from longitude
    float yB=builtRow.getFloat("latitude");

    float blBTUbar=builtRow.getFloat("million btu");//for diameter and for color keys

    /* map lat/longitude note:
     * (x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

    float lngB = map(xB, -74.3, -73.65, 0, width);
    float latB = map(yB, 40.95, 40.45, 0, height);

    fill(0); //fill
    float diaB=blBTUbar;//diameter of symbol

    ellipse(lngB, latB, diaB, diaB);
}

// draw the bar
rect(lb+(i*barWidth), height-of, barWidth, -barHeight-of);

}
}

8, "1980": 11, "1981": 5, "1982": 5, "1983": 4, "1984": 5, "1985": 7, "1986": 13, "1987": 9,
"1988": 7, "1989": 7, "1991": 3, "1998": 1, "2000": 2, "2001": 2, "2002": 4, "2003": 2, "2004": 2,
"2005": 4, "2006": 2, "2007": 4, "2008": 2, "2009": 1 }

```

insert a 'for' statement using the TableRow function to match the year of each bar and the year buildings were built as listed in the original manhattanBoiler table

## SKETCH: BOILER 1C **INTERACTIVE LINKING**

Since this sketch has a timeline feature, akin to the 311 bar-graph timeline, we can build more intelligent interaction onto that framework.

Find the code at the end of void draw( ) that forms the bar graph. Copy the above into the major loop defining the table read-through that starts “for (int i = 0; i < sortedCounts.length; i++) {”. To define a reaction between hoovering-over-the-bar-graph and the reading-of-map-quantities, we start with a TableRow function. Here, the TableRow is searching through each row in the manhattanBoiler table to see if it matches ‘sortedDates[i]’, which is to say the current date which is defining the bar.

```

text(sortedCounts[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-30-of);
text(sortedDates[i], (lb+(i*barWidth)+barWidth/2), height-barHeight-20-of);

// add the if/then that will reinitiate the series of row reads...
for (TableRow builtRow : manhattanBoiler.findRows(sortedDates[i], "built")) {

    //-----geographic location-----
    float xB=builtRow.getFloat("longitude");// for each row, grabs the float from longitude
    float yB=builtRow.getFloat("latitude");

    float blBTUbar=builtRow.getFloat("million btu");//for diameter and for color keys

    /* map lat/longitude note:
     * (x goes from -74.3 to -73.65, y goes from 40.95 to 40.45)*/

    float lngB = map(xB, -74.3, -73.65, 0, width);
    float latB = map(yB, 40.95, 40.45, 0, height);

    fill(0); //fill
    float diaB=blBTUbar;//diameter of symbol

    ellipse(lngB, latB, diaB, diaB);
}

// draw the bar
rect(lb+(i*barWidth), height-of, barWidth, -barHeight-of);
}

```

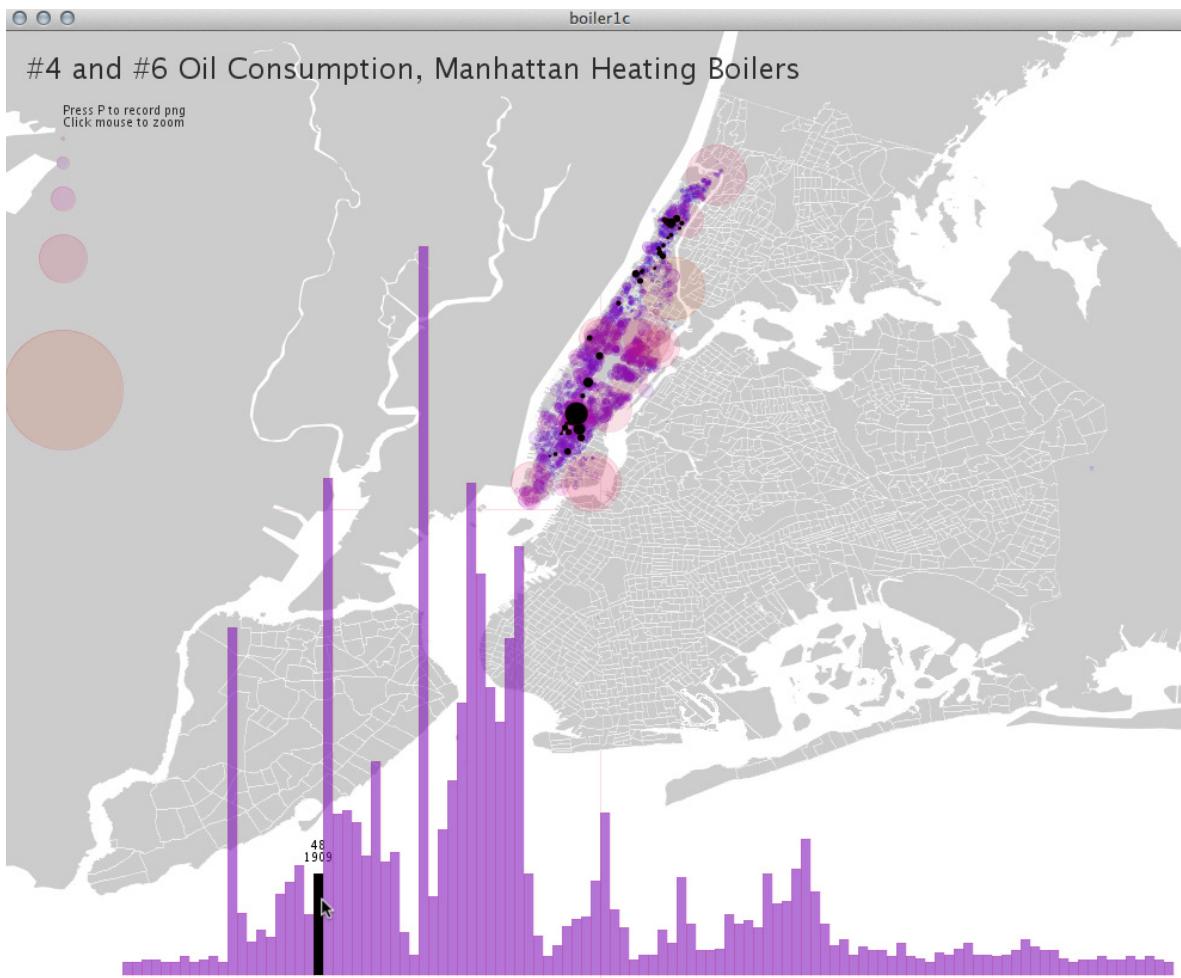
8, "1980": 11, "1981": 5, "1982": 5, "1983": 4, "1984": 5, "1985": 7, "1986": 13, "1987": 9, "1988": 7, "1989": 7, "1991": 3, "1998": 1, "2000": 2, "2001": 2, "2002": 4, "2003": 2, "2004": 2, "2005": 4, "2006": 2, "2007": 4, "2008": 2, "2009": 1 }

Then we just copy the code from the initial geolocation sequence.

Rename the variables to avoid conflict.

## SKETCH: BOILER 1C **INTERACTIVE LINKING**

After we establish that initial matching, we'll merely repeat the code needed to draw the original dots. With new variable names and a new fill color, i.e. fill(0), this will thus enable hoovers on the bar graph to highlight map features. As before, the date and number of matches will also show above the graph when the mouse is in range.



## SKETCH: BOILER 1C **TIMELINE-GEO INTERACTIONS**



```
void zipCodes (String zipcoding) {  
    if (keyPressed) {  
        if (key == 'c' || key == 'C') {  
  
zipBoiler= new Table(); //created a new each time....so as to not sum  
  
for (TableRow zipSum : manhattanBoiler.findRows(zipcoding, "zipcode")) {  
    TableRow newRow = zipBoiler.addRow(zipSum);  
}  
saveTable(zipBoiler, "data/boilers_Z" + zipcoding + ".csv"); // all boilers manhattan as table  
  
rowCountZip=zipBoiler.getRowCount(); // all in that zipcode  
println(rowCountZip);  
  
//so now can we get keys for all those buildingtypes... same as setup...  
for (TableRow type : zipBoiler.rows()) {  
    // get the values under the building built date  
    String types = type.getString(8);  
    zipTypes.append(types);  
}  
1
```

## SKETCH: BOILER 1D **CODE STRUCTURES**

What we're going to do now is to add a new function to processing.

You should be familiar with 'void setup(){}' or 'void mousePressed(){}'. Functions that begin with 'void' are functions that do some work, like drawing forms, but don't return a data result (such as calculating a specific sum or generating a new string phrase). Look up functions on [processing.org](http://processing.org) for how to create new float, String, or int functions. For now, we're going to create a new void function at the very end of the program in order to draw data-roses based on summed zipcode info. Write 'void zipCodes (String zipcoding){...}'. Here we have inserted a variable into the initial parentheses (). This variable, String zipcoding, will hold the zipcode that we use to match/filter table values. Functions can accept multiple input variables.

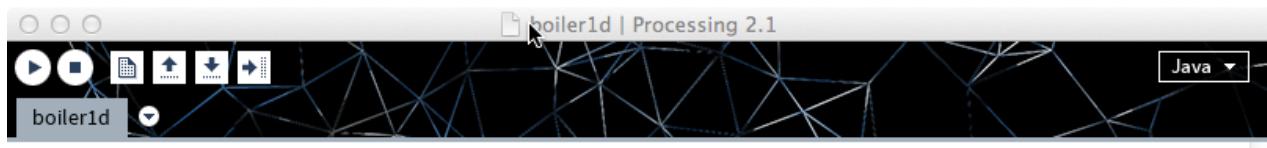


```
if (zoom==false && mouseX>(lng-2) && mouseX<(lng+2) && mouseY>(lat-2) && mouseY<(lat+2)) {  
    textSize(10);  
    textAlign(LEFT);  
    fill(0);  
    text(oil + " oil, " + date + " built, " + type + " , " + zip, 750, 260);  
  
    //-----other info graphics? on interaction  
    // back within the main void draw() function,  
    // we'll place this function within the mouse interaction/label trigger area  
    /* pie chart for types in zip code-  
    getRow= based on zipcode matching hoover row  
  
    BUT LET'S ADD AS A SEPERATE FUNCTION IN ORDER TO KEEP CODE CLEAN/DISCUSS STRUCTURE*/  
    zipCodes (zip);  
}  
else if (zoom==true && mouseX>(lng-1) && mouseX<(lng+1) && mouseY>(lat-1) && mouseY<(lat+1)) {  
    textSize(1);  
    textAlign(LEFT);  
    fill (0);  
    text(oil + " " + date + " " + type + " " + zip, mouseX+2, mouseY);  
  
    fill(cyan);  
}
```

## SKETCH: BOILER 1D **CODE STRUCTURES**

We'll return to defining the function `zipCodes` in a few moments. Above is how we'll call it within the main `void draw()` function.

Note that here the variable used with `zipCodes(zip)` is `zip`. If you scroll up, the variable `zip` is defined by “`String zip=manhattanBoiler.getString(i, “zipcode”);`” when iterating through the rows in `manhattanBoiler` table. Thus we are passing the `zipcode`, as pulled up per row, of each dot we hoover over to the new function.



```

void zipCodes (String zipcoding) {
    if (keyPressed) {
        if (key == 'c' || key == 'C') {
            zipBoiler= new Table(); //created a new each time....so as to not sum

            for (TableRow zipSum : manhattanBoiler.findRows(zipcoding, "zipcode")) {
                TableRow newRow = zipBoiler.addRow(zipSum);
            }
            saveTable(zipBoiler, "data/boilers_Z" + zipcoding + ".csv"); // all boilers manhattan as table
            rowCountZip=zipBoiler.getRowCount(); // all in that zipcode
            println(rowCountZip);
        }
    }
    //so now can we get keys for all those buildingtypes... same as setup...
    for (TableRow type : zipBoiler.rows()) {
        // get the values under the building built date
        String types = type.getString(8);
        zipTypes.append(types);
    }
}

```

limit the function with two keyPressed conditionals

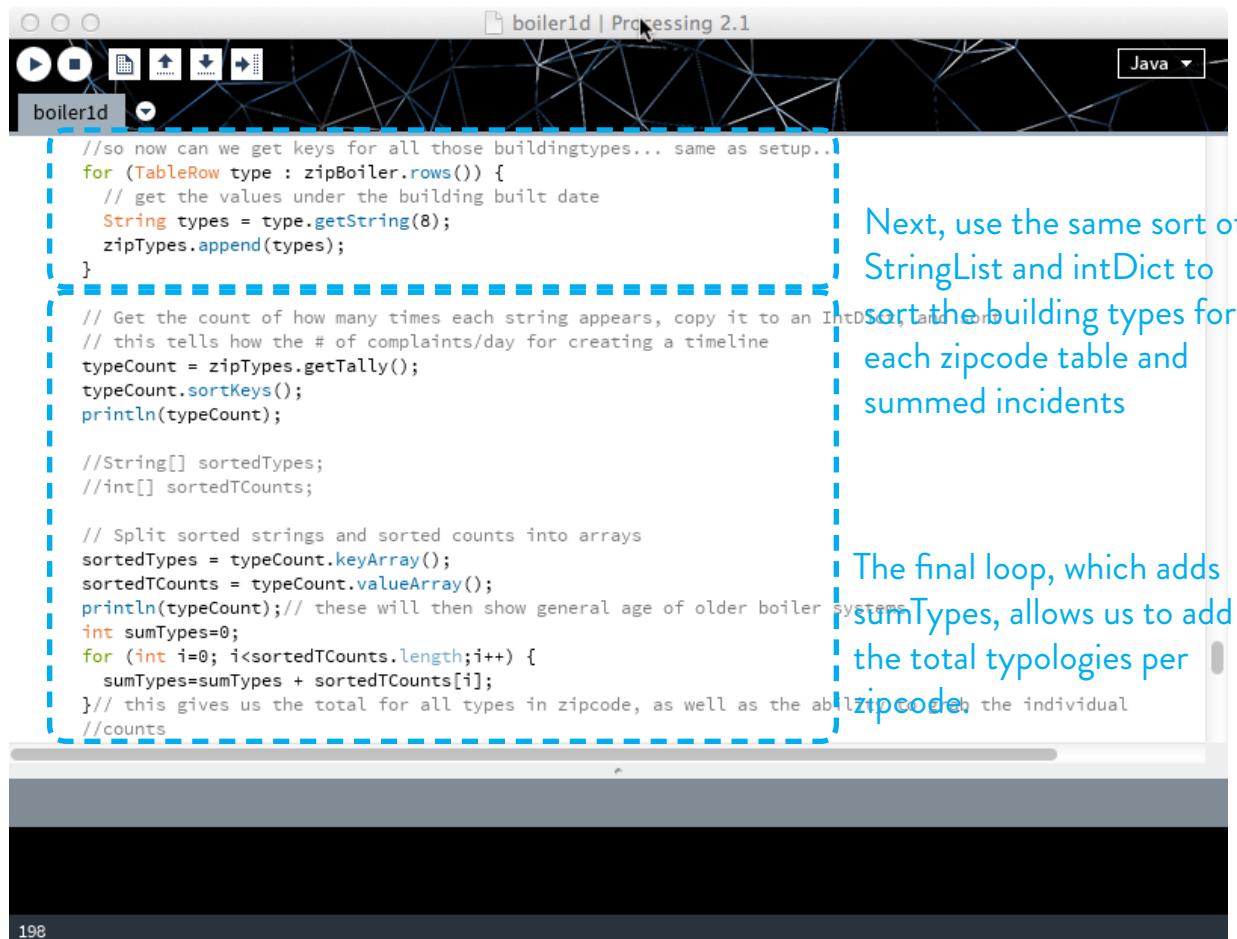
begin by creating a 'temp' table 'zipBoiler' and adding the info from all rows that matches the inherited zipcode from 'String zipcoding'

remember to add 'Table zipBoiler;' at the top as a global variable

## SKETCH: BOILER 1D **NEW TABLES (AGAIN)**

In order to hold data, we'll create new tables organized around unique zipcodes.

Return to the introduction of new tables on tutorial wk1, p 54 to refresh your memory on these structures. What is different here is that instead of being written once, as in void setup(), this function will create a new table anytime the mouse is over a dot and the key 'c' or 'C' is pressed. It will also, in effect, write-over a file if we hover over a zipcoded area more than once, but the data in that file will be the same.



```
// so now can we get keys for all those buildingtypes... same as setup...
for (TableRow type : zipBoiler.rows()) {
    // get the values under the building built date
    String types = type.getString(8);
    zipTypes.append(types);
}

// Get the count of how many times each string appears, copy it to an IntDict
// this tells how the # of complaints/day for creating a timeline
typeCount = zipTypes.getTally();
typeCount.sortKeys();
println(typeCount);

//String[] sortedTypes;
//int[] sortedTCounts;

// Split sorted strings and sorted counts into arrays
sortedTypes = typeCount.keySet();
sortedTCounts = typeCount.values();
println(typeCount); // these will then show general age of older boiler systems
int sumTypes=0;
for (int i=0; i<sortedTCounts.length;i++) {
    sumTypes=sumTypes + sortedTCounts[i];
}
// this gives us the total for all types in zipcode, as well as the ability to code the individual
//counts
```

Next, use the same sort of `StringList` and `intDict` to **sort the building types for each zipcode table and summed incidents**

The final loop, which adds `sumTypes`, allows us to add **the total typologies per zipcode**.

## SKETCH: BOILER 1D **SIMPLE COUNTS AGAIN**

After creating the new table, use the simple summing procedures from before (wk 2, p 59-62). As in that case, remember to declare the `StringList`, `intDict`, and arrays up in the area for global variables.



```

boiler1d | Processing 2.1
Java

sumTypes=sumTypes + sortedTCounts[i];
}// this gives us the total for all types in zipcode, as well as the ability to grab the individual
//counts
println(sumTypes);
float[] pie=new float [sortedTCounts.length];
for (int i=0; i<sortedTCounts.length;i++) {
  pie[i]=float(100*sortedTCounts[i]/sumTypes);
}

float lastAngle=0;
float sumBtu=0;
for (int i=0; i<sortedTCounts.length;i++) {
  for (TableRow zipBtu : zipBoiler.findRows(sortedTypes[i], 8)) {
    sumBtu=sumBtu+(zipBtu.getFloat(7));
  }
  fill(purple, (i*10)+100);
  arc(750, 450, sumBtu, sumBtu, lastAngle, lastAngle+(pie[i]/100*TWO_PI));
  lastAngle += (pie[i]/100*TWO_PI);
  fill(0);
  text(sortedTypes[i]+ " , " + sortedTCounts[i] + " , total Btu: "+ sumBtu, 750, 650+10*i);
}
// this gives us the total for all types in zipcode, as well as the ability to grab the individual
//counts
println(pie);

```

Declare a new array 'float [ ] pie' that is the lenght of sortedCounts.length. Then define each pie position[i] as a percentage of number per type/all types.

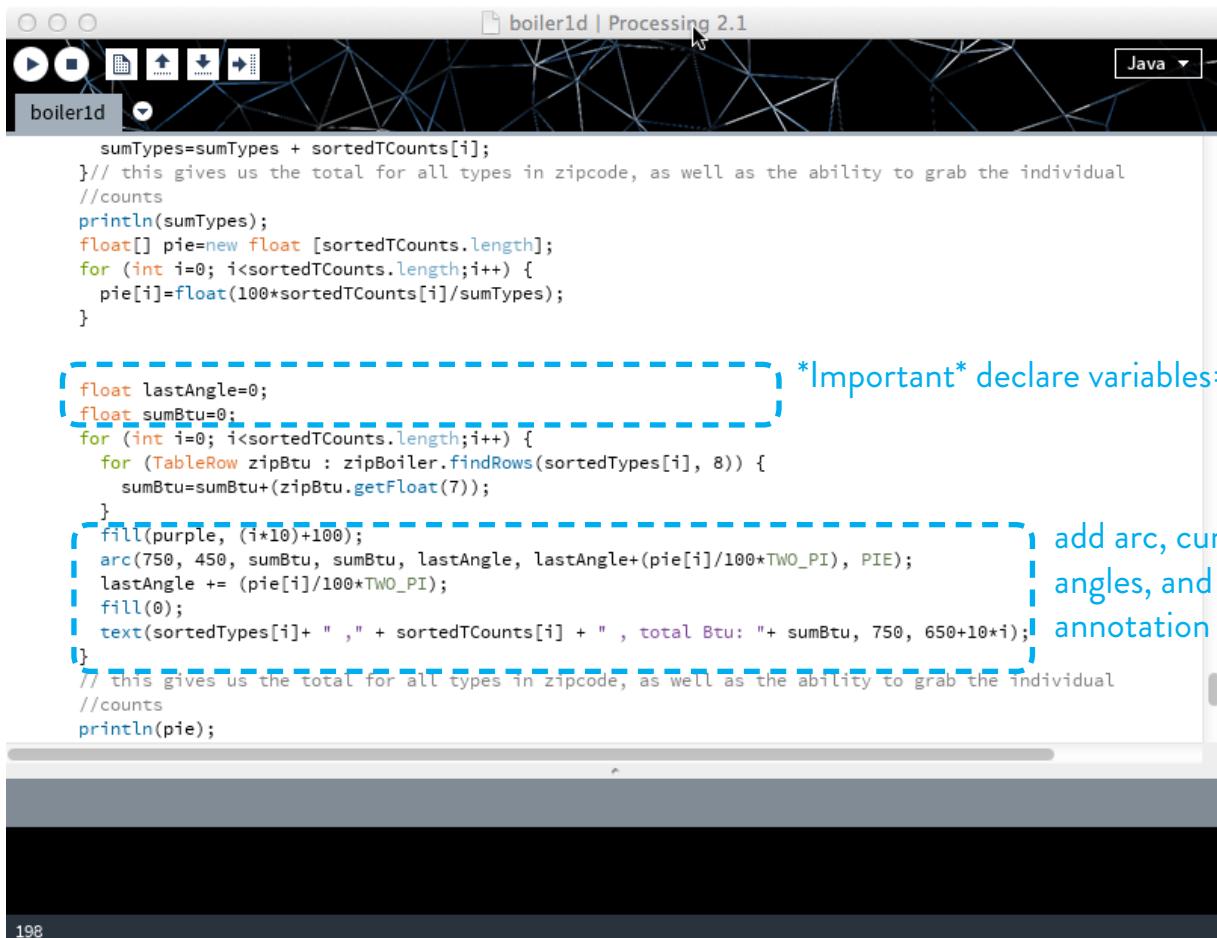
Declare a new float 'sumBtu' that, for each sorted typology count, grabs the Btu for those entries in zipBoiler. It then adds those to get a variable for the arc radii.

198

## SKETCH: BOILER 1D VARIABLES FOR ARCS

Here, in order to build a data-rose that displays the percentage contribution of each building typology to that zipcodes's #4 and #6 oil use, we'll derive two sets of values.

First, we set up the array pie [i] to hold the count of properties per type as divided by the total count per zipcode. This series of percentages will be used to get the angle of each typology pie slice (percentage x 360 degrees). Second, we'll want to get the total amount of energy used in each typology as well as used overall, here as 'sumBtu'. These numbers will be used to dictated the length of the arc, i.e. the varing radii of each pie slice in our data-rose.



```

boiler1d | Processing 2.1
boiler1d

sumTypes=sumTypes + sortedTCounts[i];
}// this gives us the total for all types in zipcode, as well as the ability to grab the individual
//counts
println(sumTypes);
float[] pie=new float [sortedTCounts.length];
for (int i=0; i<sortedTCounts.length;i++) {
  pie[i]=float(100*sortedTCounts[i]/sumTypes);
}

float lastAngle=0;
float sumBtu=0;
for (int i=0; i<sortedTCounts.length;i++) {
  for (TableRow zipBtu : zipBoiler.findRows(sortedTypes[i], 8)) {
    sumBtu=sumBtu+(zipBtu.getFloat(7));
  }
  fill(purple, (i*10)+100);
  arc(750, 450, sumBtu, sumBtu, lastAngle, lastAngle+(pie[i]/100*TWO_PI), PIE);
  lastAngle += (pie[i]/100*TWO_PI);
  fill(0);
  text(sortedTypes[i]+ " , " + sortedTCounts[i] + " , total Btu: " + sumBtu, 750, 650+10*i);
}
// this gives us the total for all types in zipcode, as well as the ability to grab the individual
//counts
println(pie);

```

\*Important\* declare variables=0; | add arc, cumulative  
angles, and text  
annotation

## SKETCH: BOILER 1D **COMPLETE DATA-ROSE**

After setting up those variables, we can thus use them with the data-roses.

Here, there are two simple things that have to be done to get clean numbers. First, any of the variables (lastAngle and sumBtu) should be declared as equal to zero. We do this here, instead of at the top with global variables, so that each time, with each new zipcode, we're not adding to an infinite sum process, but always restarting at zero. Second, in using the percentages of pie[i], we need to cumulatively add those angles up to 360 degrees or TWO\_PI. The variable lastAngle is thus set after the arc and each time received the addition of (pie[i]/100 \* TWO\_PI). Notice that the definition of pie pieces happens within the loop iterating through sortedTCounts.



The screenshot shows the Processing 2.1 interface with a network visualization in the background. The sketch is titled 'boiler1d'. The code is as follows:

```
boiler1d | Processing 2.1
Java

}
fill(purple, (i*10)+100);
arc(750, 450, sumBtu, sumBtu, lastAngle, lastAngle+(pie[i]/100*TWO_PI), PIE);
lastAngle += (pie[i]/100*TWO_PI);
fill(0);
text(sortedTypes[i]+ " , " + sortedTCounts[i] + " , total Btu: " + sumBtu, 750, 650+10*i);
}
// this gives us the total for all types in zipcode, as well as the ability to grab the individual
// counts
println(pie);

sumBtu=0;
//ellipse (750, 450, 200, 200); //placeholder location
//-----clear the values to only call during keypress, reload-----
zipTypes.clear(); // hell yeh, works to restart count on each round
typeCount.clear();
//sumTypes=0;
println(zipTypes);
}
// need the keystroke or else iterates with each loop, not with singular sample
}

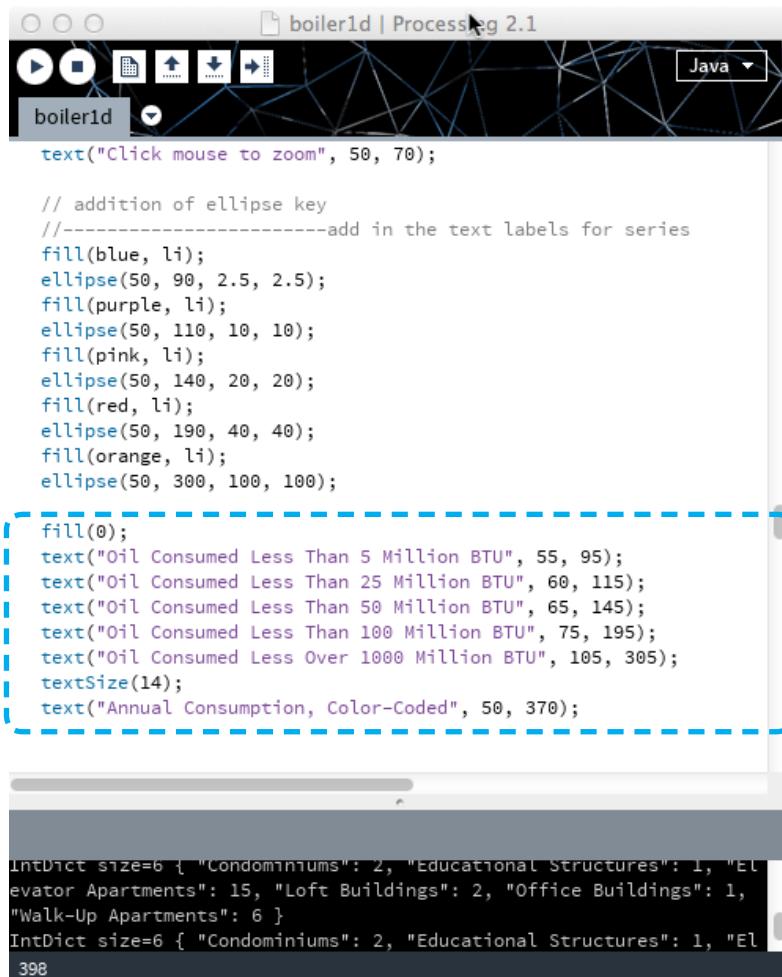
*Important* clear arrays
for re-use.
```

The code includes a note in the margin: '\*Important\* clear arrays for re-use.'

The status bar at the bottom says 'Done Saving.' and '398'.

## SKETCH: BOILER 1D **KEEPING STATIC COUNTS**

Because we loop through this function, it's important to make sure that everything is clearer, erased, or set to zero so as not to get double, triple, quad (etc.) counts of both sums and objects in arrays.



The screenshot shows the Processing IDE with the sketch titled "boiler1d | Processing 2.1". The code is as follows:

```
text("Click mouse to zoom", 50, 70);

// addition of ellipse key
//-----add in the text labels for series
fill(blue, li);
ellipse(50, 90, 2.5, 2.5);
fill(purple, li);
ellipse(50, 110, 10, 10);
fill(pink, li);
ellipse(50, 140, 20, 20);
fill(red, li);
ellipse(50, 190, 40, 40);
fill(orange, li);
ellipse(50, 300, 100, 100);

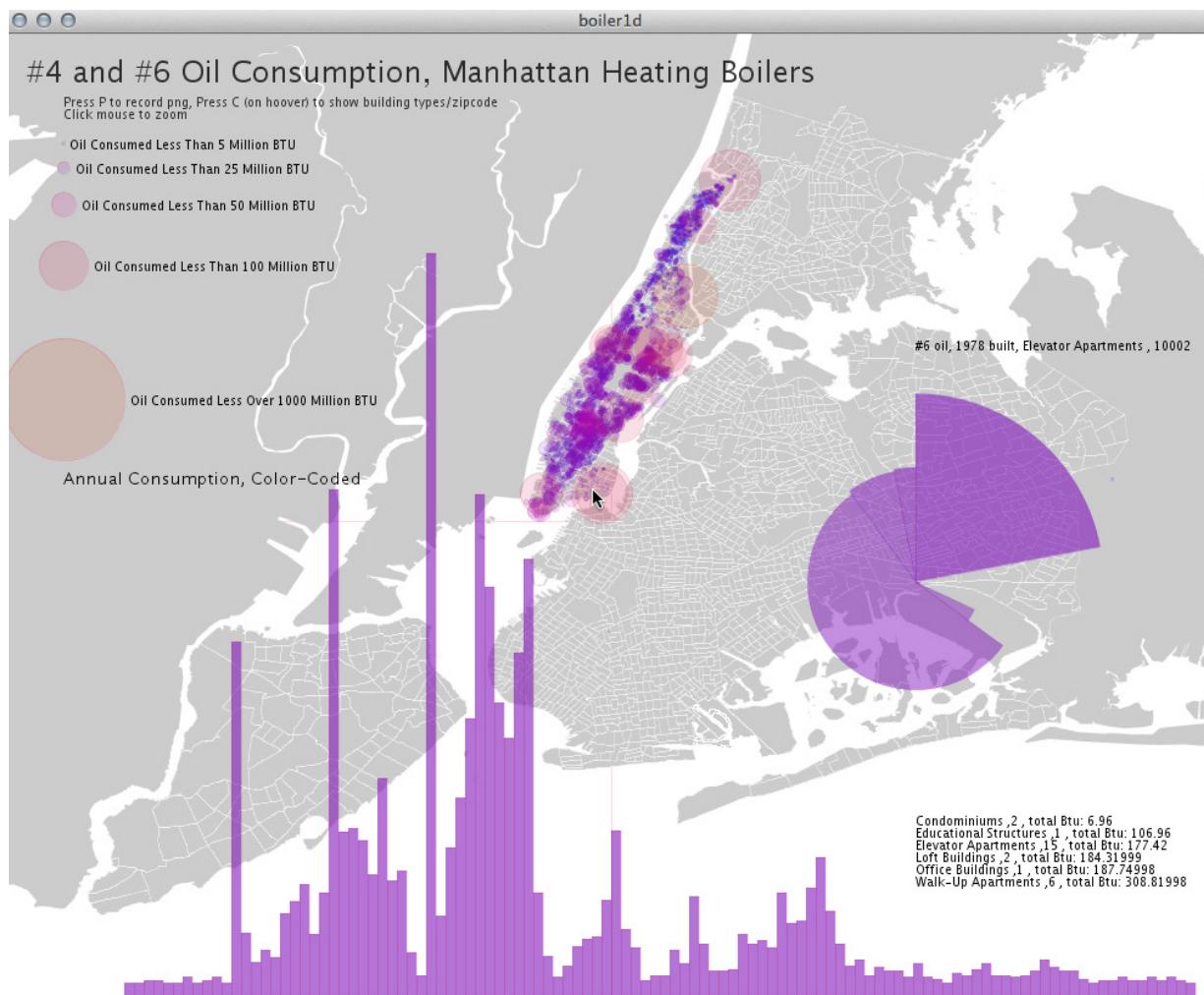
fill(0);
text("Oil Consumed Less Than 5 Million BTU", 55, 95);
text("Oil Consumed Less Than 25 Million BTU", 60, 115);
text("Oil Consumed Less Than 50 Million BTU", 65, 145);
text("Oil Consumed Less Than 100 Million BTU", 75, 195);
text("Oil Consumed Less Over 1000 Million BTU", 105, 305);
textSize(14);
text("Annual Consumption, Color-Coded", 50, 370);

IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
elevator Apartments": 15, "Loft Buildings": 2, "Office Buildings": 1,
"Walk-Up Apartments": 6 }
IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
398
```

A blue dashed box highlights the text labels for the keys. A callout bubble with the text "Add some text labels for the keys." is positioned to the right of the highlighted code.

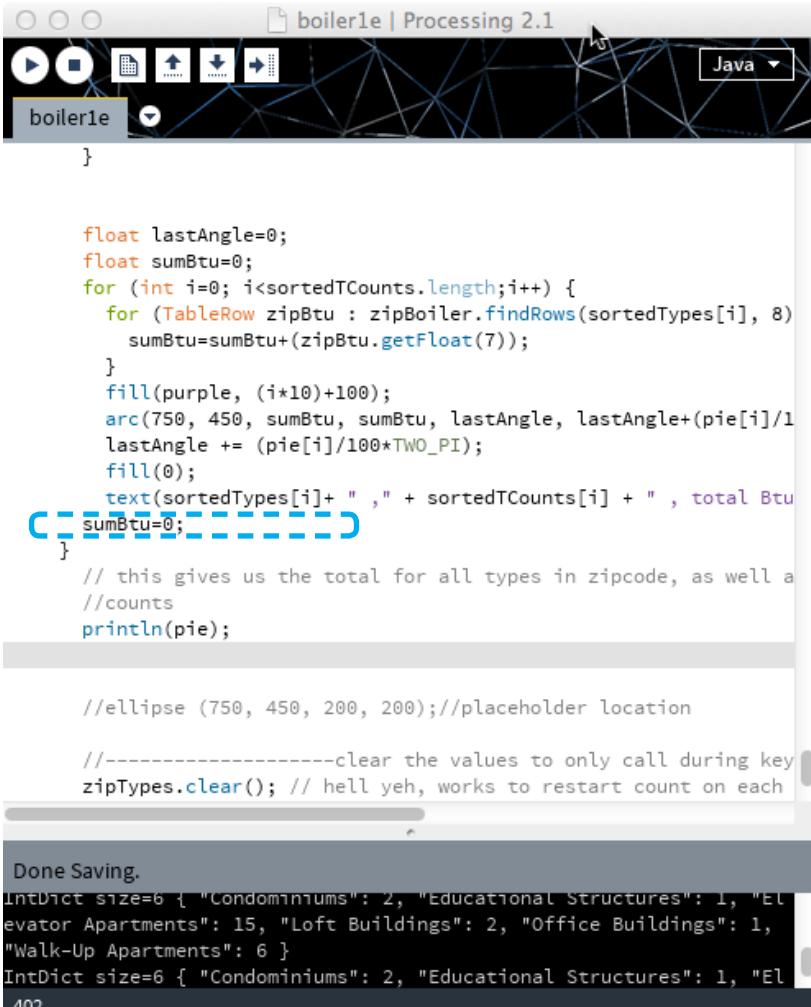
## SKETCH: BOILER 1D **A FEW LABELS**

As just a bit of housekeeping, go back up to the keys/label area of void draw() and add in the following code after the color-coded key.



## SKETCH: BOILER 1D **CURRENT OUTPUT**

As seen above, the graphics are beginning to come together. But we're just getting a simple summation of radii values (i.e. total energy consumption) for each building typology.



```

float lastAngle=0;
float sumBtu=0;
for (int i=0; i<sortedTCounts.length;i++) {
  for (TableRow zipBtu : zipBoiler.findRows(sortedTypes[i], 8)
    sumBtu=sumBtu+(zipBtu.getFloat(7));
  }
  fill(purple, (i*10)+100);
  arc(750, 450, sumBtu, sumBtu, lastAngle, lastAngle+(pie[i]/1
  lastAngle += (pie[i]/100*TWO_PI);
  fill(0);
  text(sortedTypes[i]+ " , " + sortedTCounts[i] + " , total Btu
sumBtu=0;
}
// this gives us the total for all types in zipcode, as well as
//counts
println(pie);

//ellipse (750, 450, 200, 200); //placeholder location

//-----clear the values to only call during key
zipTypes.clear(); // hell yeh, works to restart count on each

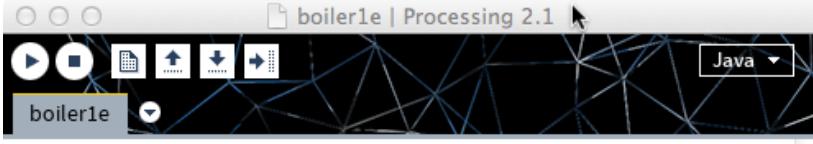
Done Saving.
IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
evator Apartments": 15, "Loft Buildings": 2, "Office Buildings": 1,
"Walk-Up Apartments": 6 }
IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
402

```

Move from below `println(pie);` and outside the iteration brackets to the interior.

## SKETCH: BOILER 1E SUM CLEARANCE PLACEMENT

In order to correct that data-rose summation of values, we need to adjust the placement of where we set `sumBtu=0`. If we leave it in the original place (grey above) it merely clear the value after every type in the zipcode is summed. To clear after each typology, we need to nest it within the initial iteration through that particular typology. Thus we move it up so that a) the inner loop finds and adds all type matches to `sumBtu`, b) that value is used to define the arc radii, c) we then set `sumBtu` back to zero, and d) the loop can then return and start the process for the next type.



```

int rowCountZip;//counts for 311 dogs in manhattan

//-----INTERACTION aids-----
boolean zoom=false; //zoom to area or not
boolean m=true; //map on or not

PFont avenir; Declare a font in the global area

//*****-----BEGIN SETUP/ LOADS ONCE-----
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYCAlts.svg");
  boiler=loadTable("boilers_clean.csv", "header");

  manhattanBoiler=new Table();
  avenir = loadFont("Avenir-Roman-24.vlw"); // so the above will disp
  textFont(avenir, 10); load and specify size within void setup()

  //-----after table loads grab some basic values-----
  rowCount=boiler.getRowCount();
  println(rowCount);
}

Done Saving.
IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
evator Apartments": 15, "Loft Buildings": 2, "Office Buildings": 1,
"Walk-Up Apartments": 6 }
IntDict size=6 { "Condominiums": 2, "Educational Structures": 1, "El
402

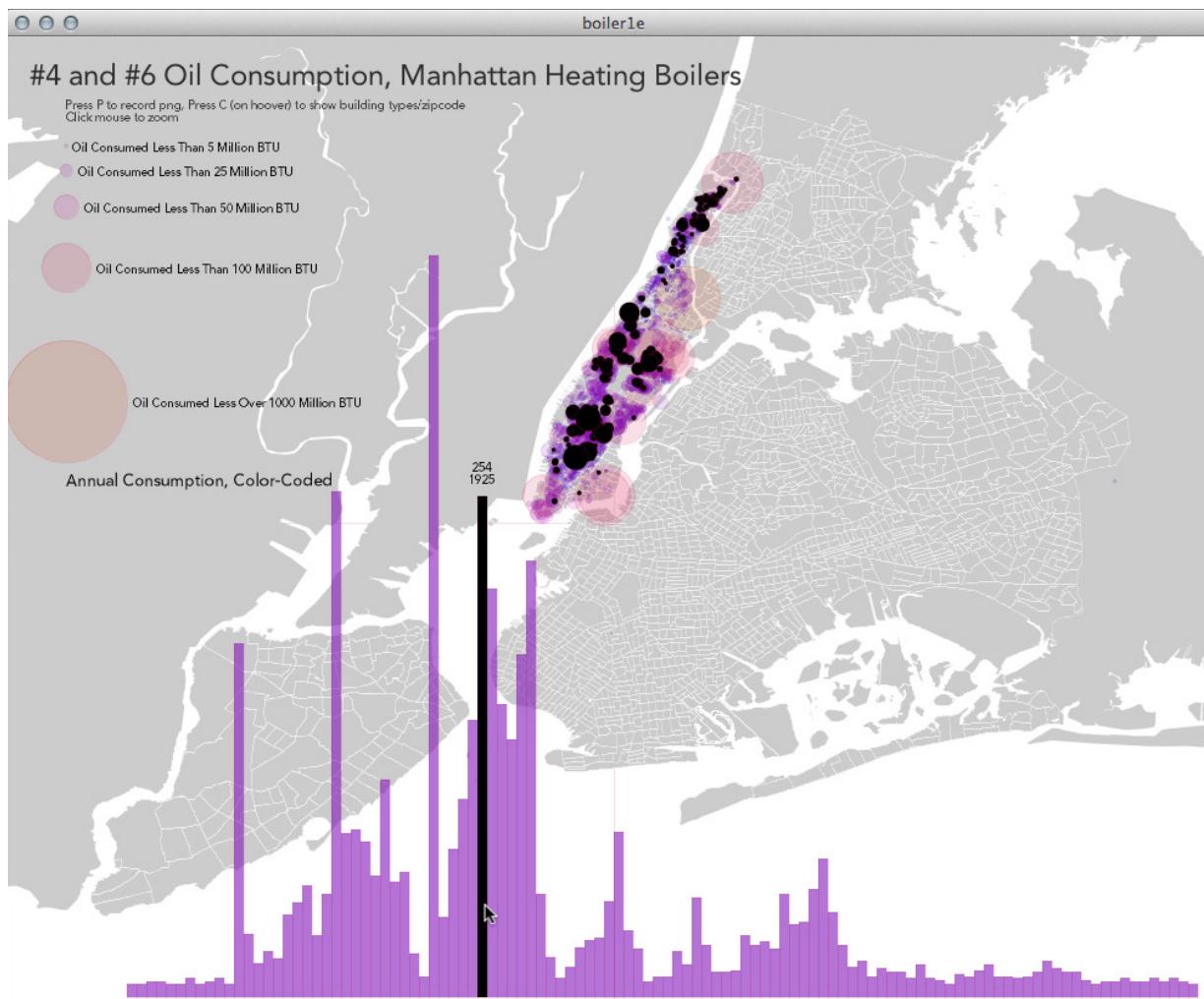
```

create fonts within the menu  
TOOLS/CREATE FONT

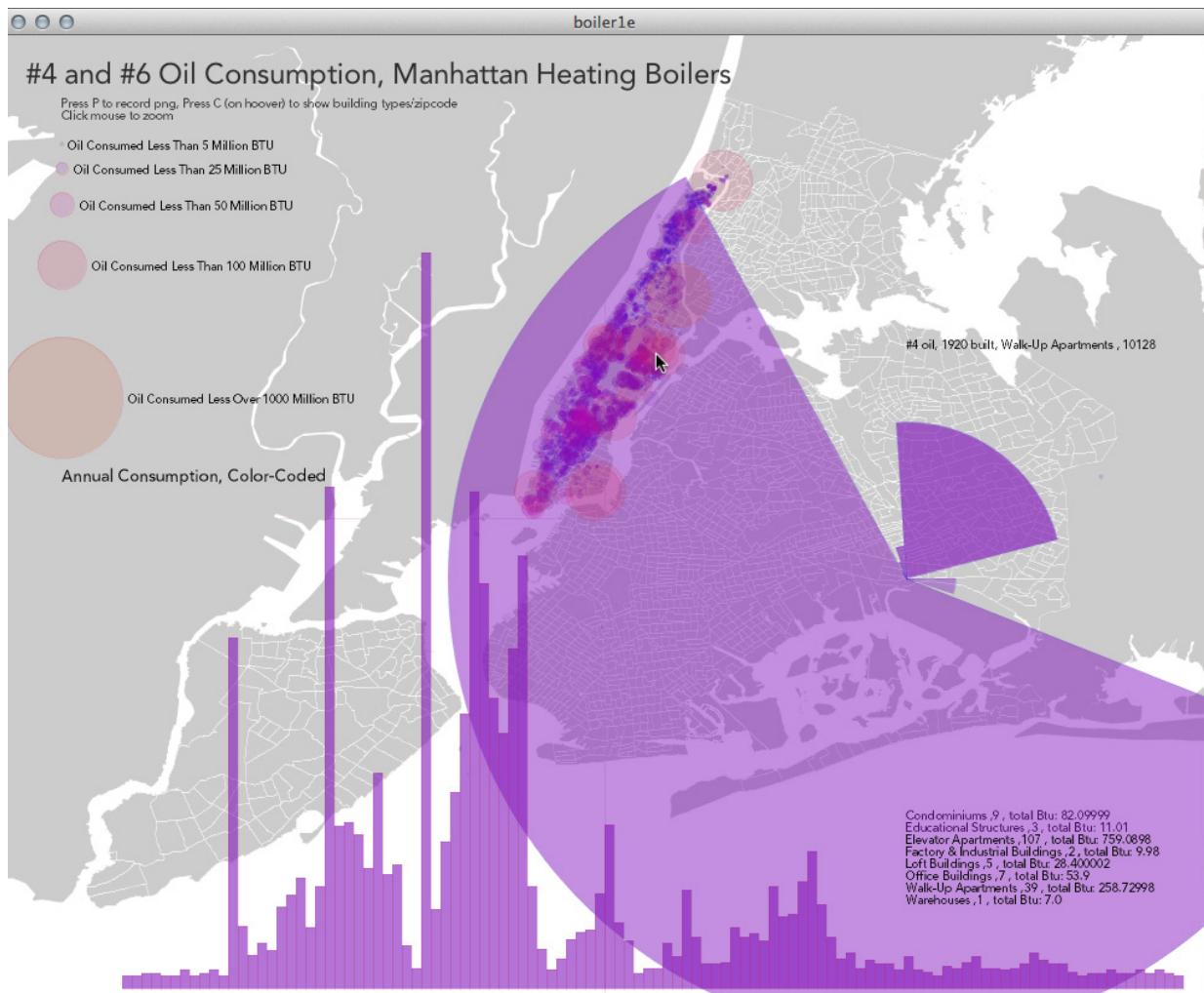
This font will then be found in the data folder of your sketch and can be used on any computer, even those without the original loaded into their systems.

## SKETCH: BOILER 1E FONT ADJUSTMENTS

Here, we're just adding a font for a slightly cleaner look. If you're worried about matching composition and layout on existing boards, several fonts/sizes can be used but you will need to dictate where font is being call each time.



SKETCH: BOILER 1E



SKETCH: BOILER 1E

9 WK 1-BASIC PLOTTING

59 WK 2-MATH, MULTIPLE FORMS, INTERACTION

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

