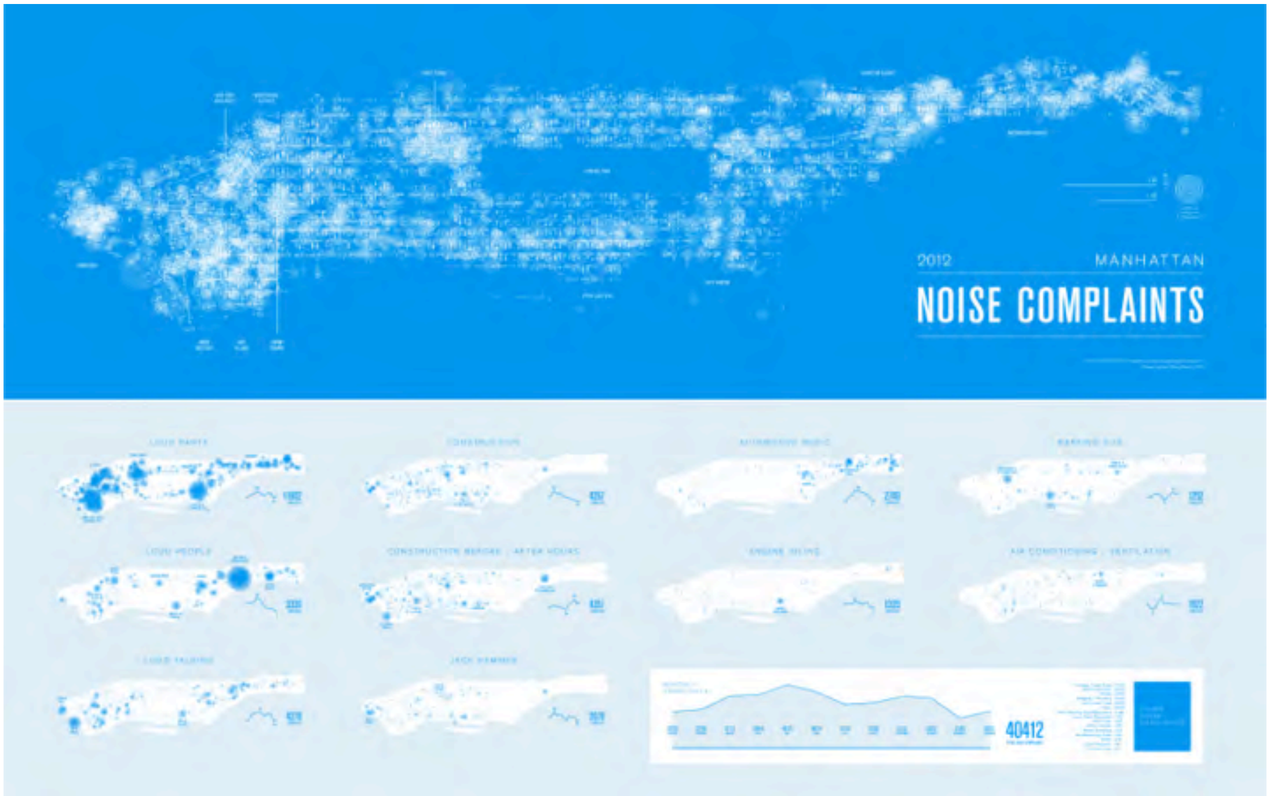
PROCESSING FOR LANDSCAPE ARCHITECTURE

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



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

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

NO CODING EXPERIENCE NECESSARY! THIS IS AN INTRODUCTION!

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

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

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

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

WORKSHOP BASICS:

Processing.org:

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

Background Readings:

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

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

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

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

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

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

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

Sourcing Code & Code Information:

GitHub:
https://github.com/

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

Stack Overflow:
http://stackoverflow.com/

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

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

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

Data Sources & Visualization Inspiration at end of syllabus

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

WK 1-BASIC PLOTTING

  - examples from the plotting environment

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

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

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

WK 2-MATH, INTERACTIONS, & MULTIPLE DISPLAY FORMS

  - shifting from 311 dog calls to other NYC Open Data

- simple math from tables
  - StringLists
  - InterDicts
  - Arrays

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

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

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

  - Moving toward correllation and arguments with data

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

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

Useful Data Sources:

Government/Municipal- varied types:

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

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

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

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

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

Non-Profit/Press- varied types:

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

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

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

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

and so on...

Commercial- varied types:

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

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

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

https://dev.twitter.com/ twitter

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

https://developer.foursquare.com/ foursquare

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

Visualization Inspiration:

Varied types:

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

https://vimeo.com search visualization animations

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

http://www.openprocessing.org/   openprocessing

Miscellaneous Data Representation Project samples:

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

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

PROCESSING WORKSHOP

11.09.2013 [WEEK 1]

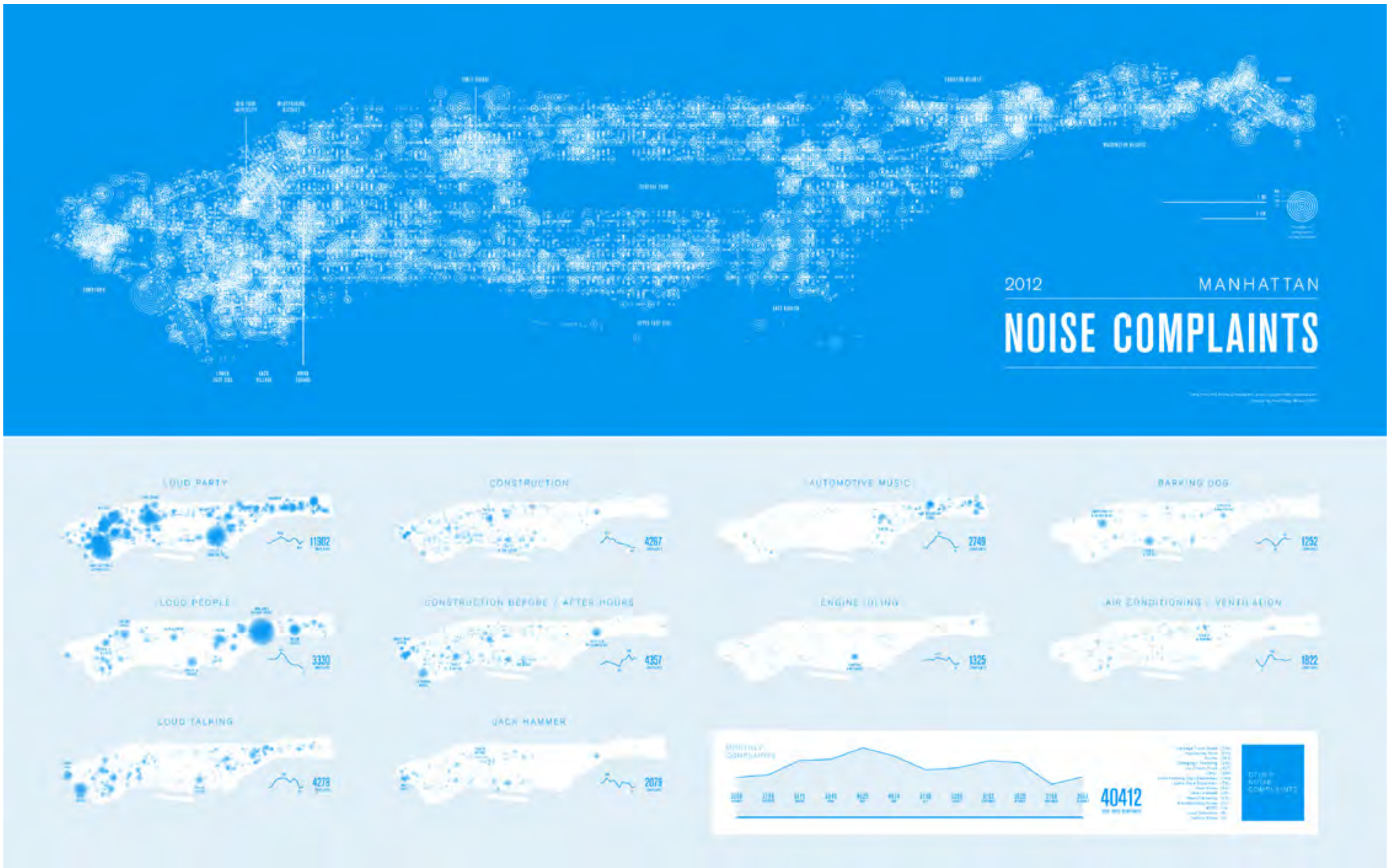
WK 1- BASIC PLOTTING

- examples from the plotting environment

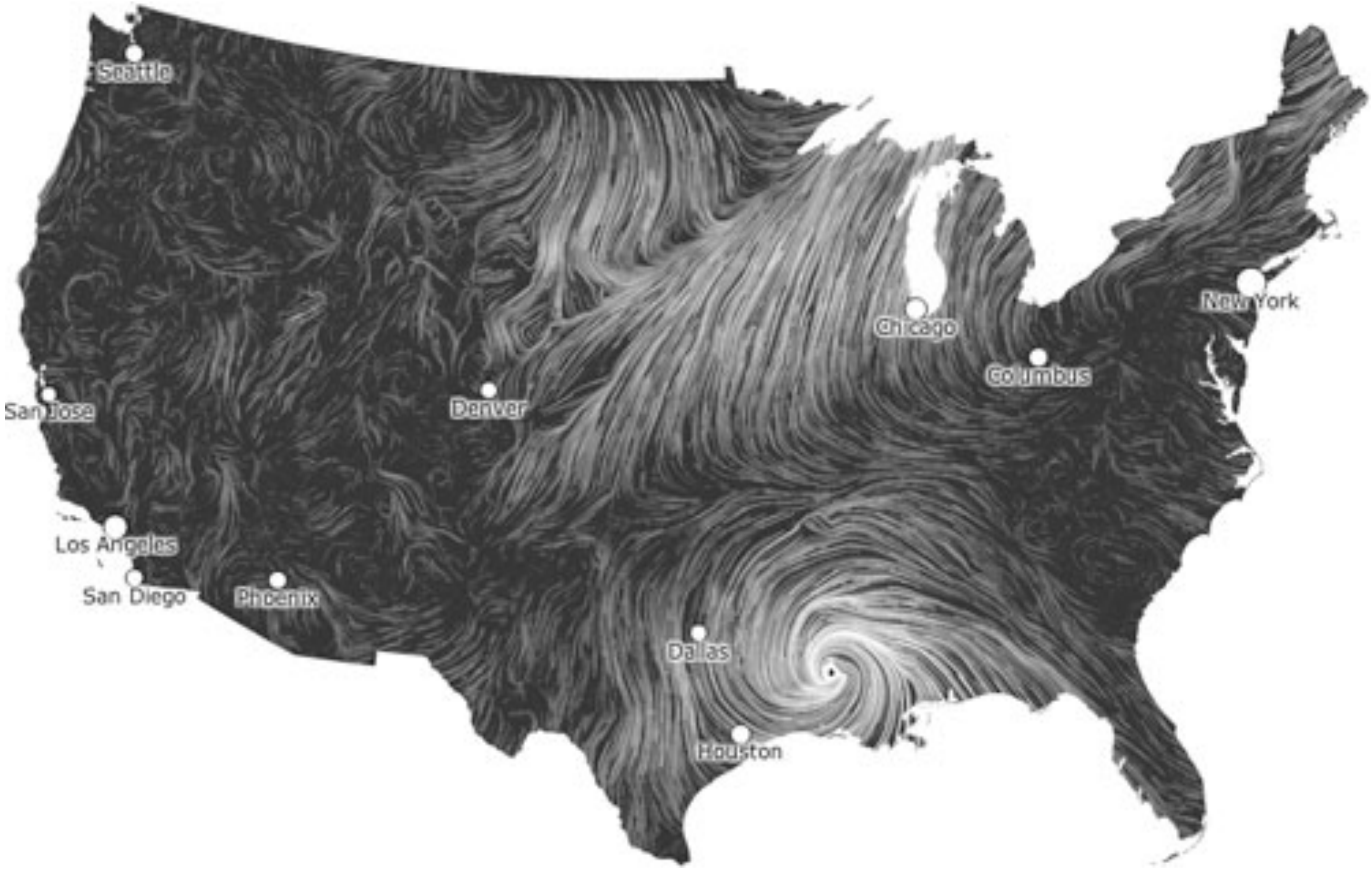
- computation structures
  - references
  - variables
  - conditionals
  - operators
  - syntax
- comparative structures

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

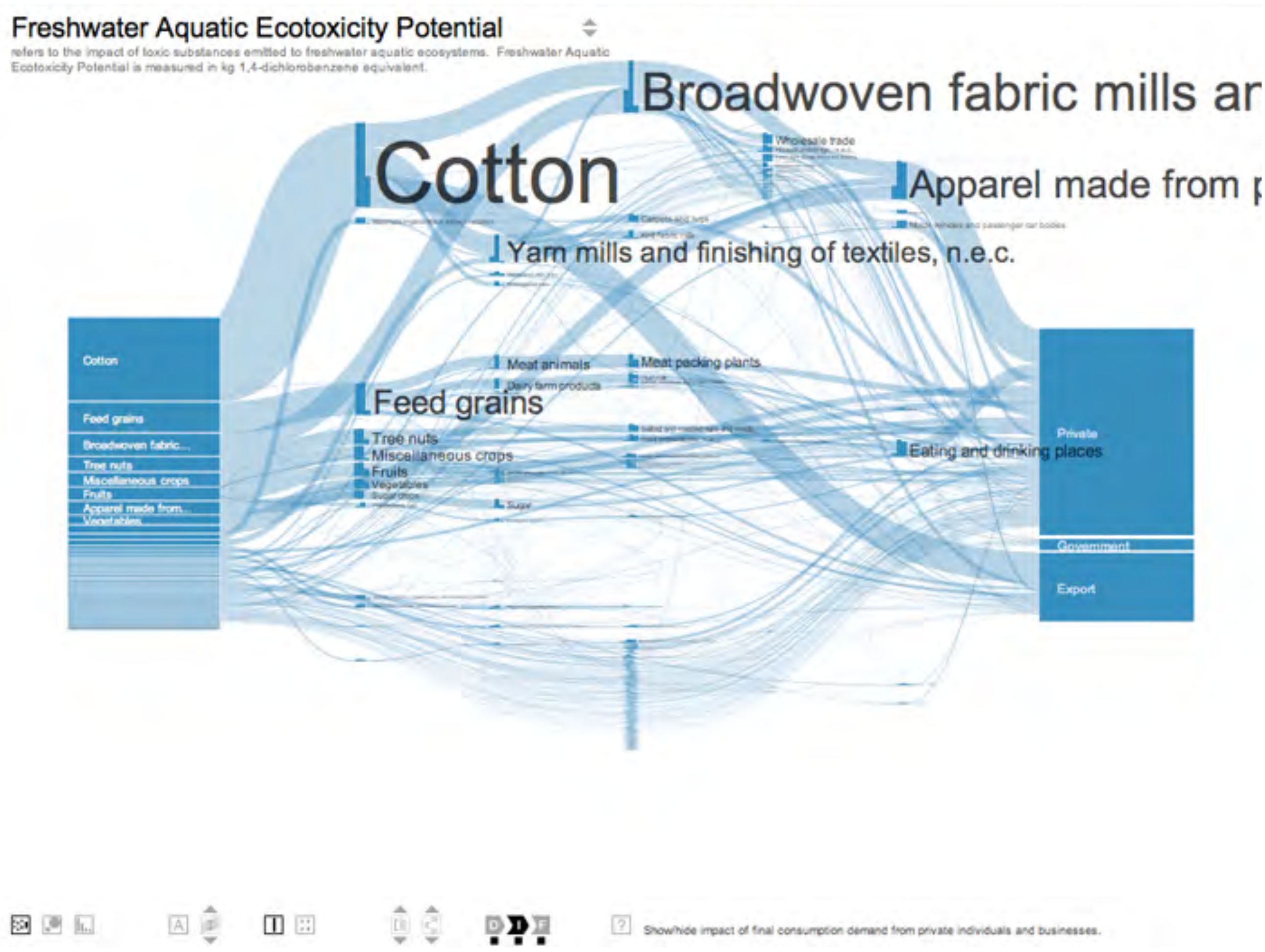
- re-constructing the 311 example
  - finding files: NYC Open data
  - reading tables- core functions
  - geoplotting data
  - symbology
  - internal & external filtering of data
  - parsing/saving new files



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/)

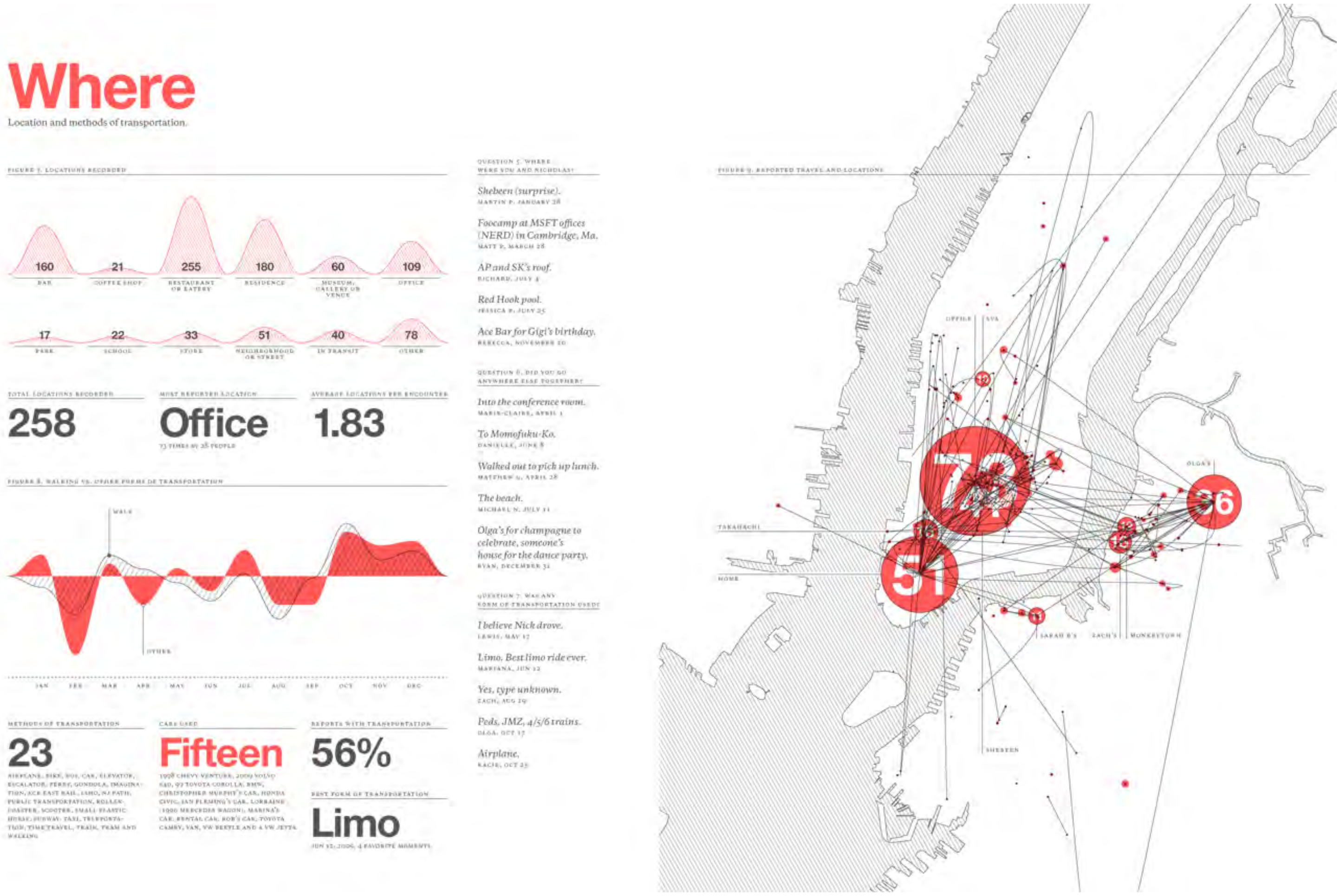


Freshwater Aquatic Ecotoxicity Potential

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

? Show/hide impact of final consumption demand from private individuals and businesses.

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



Where

Location and methods of transportation.

TOTAL LOCATIONS RECORDED

258

MOST REPORTED LOCATION

Office

73 TIMES BY 28 PEOPLE

AVERAGE LOCATIONS PER ENCOUNTER

1.83

METHODS OF TRANSPORTATION

23

AIRPLANE, BIKE, BUS, CAR, ELEVATOR, ESCALATOR, FERRY, GONDOLA, IMAGINATION, ACE EAST RAIL, LIMO, NJ PATH, PUBLIC TRANSPORTATION, ROLLER COASTER, SCOOTER, SMALL PLASTIC HORSE, SUBWAY, TAXI, TELEPORTATION, TIME TRAVEL, TRAIN, TRAM AND WALKING

CARS USED

Fifteen

1998 CHEVY VENTURE, 2009 VOLVO S40, 97 TOYOTA COROLLA, BMW, CHRISTOPHER MURPHY'S CAR, HONDA CIVIC, IAN FLEMING'S CAR, LORRAINE (1990 MERCEDES WAGON), MARINA'S CAR, RENTAL CAR, ROB'S CAR, TOYOTA CAMRY, VAN, VW BEETLE AND A VW JETTA

REPORTS WITH TRANSPORTATION

56%

BEST FORM OF TRANSPORTATION

Limo

JUN 12, 2006, 4 FAVORITE MOMENTS

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 P., 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.
DANIELLE, 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 31

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 29

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

Airplane.
KACIE, OCT 23

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

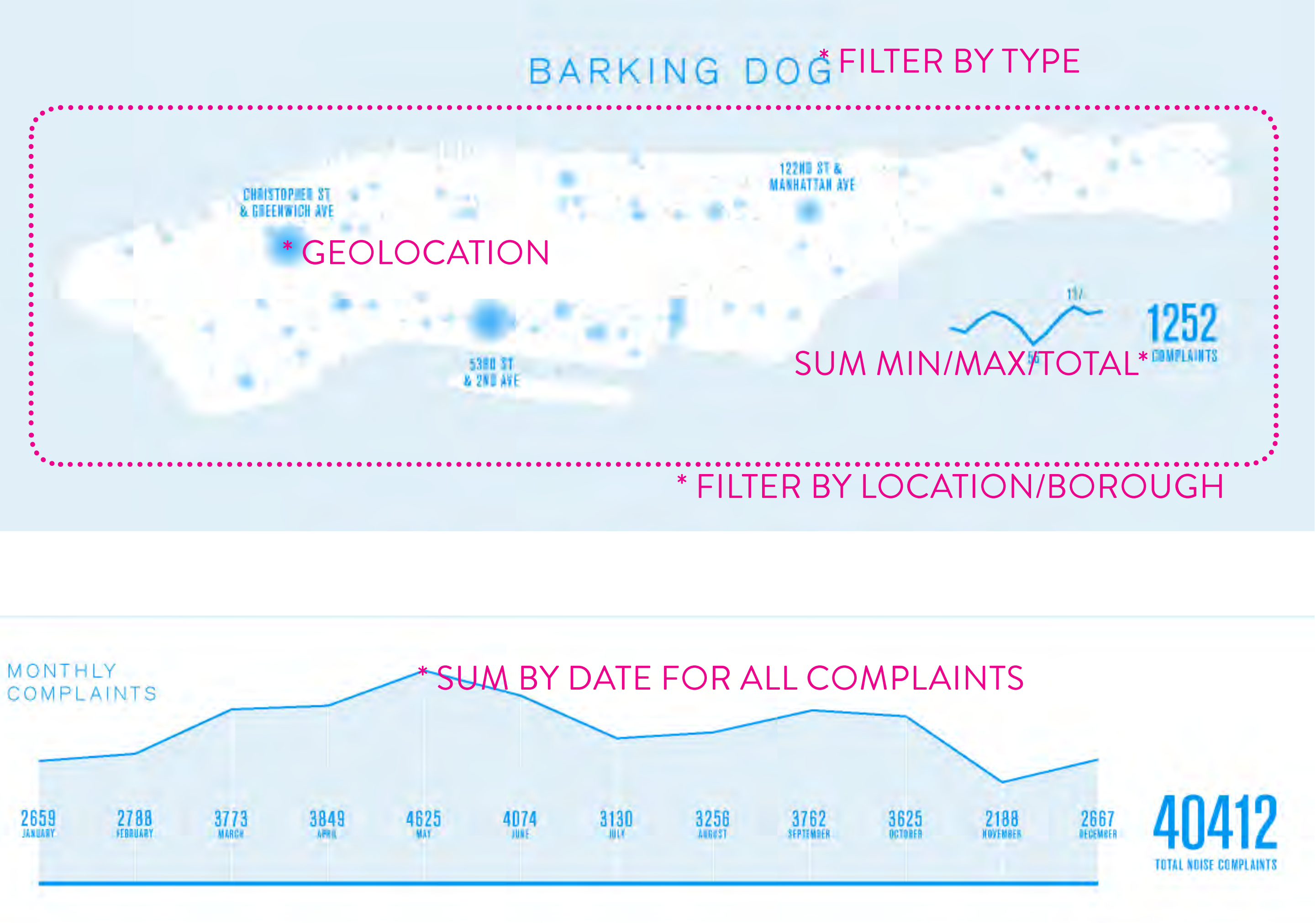
see also the video at: https://vimeo.com/70800507

WK 1

COMPUTATION STRUCTURE *(IS FAMILIAR!)*

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

- comparative structures

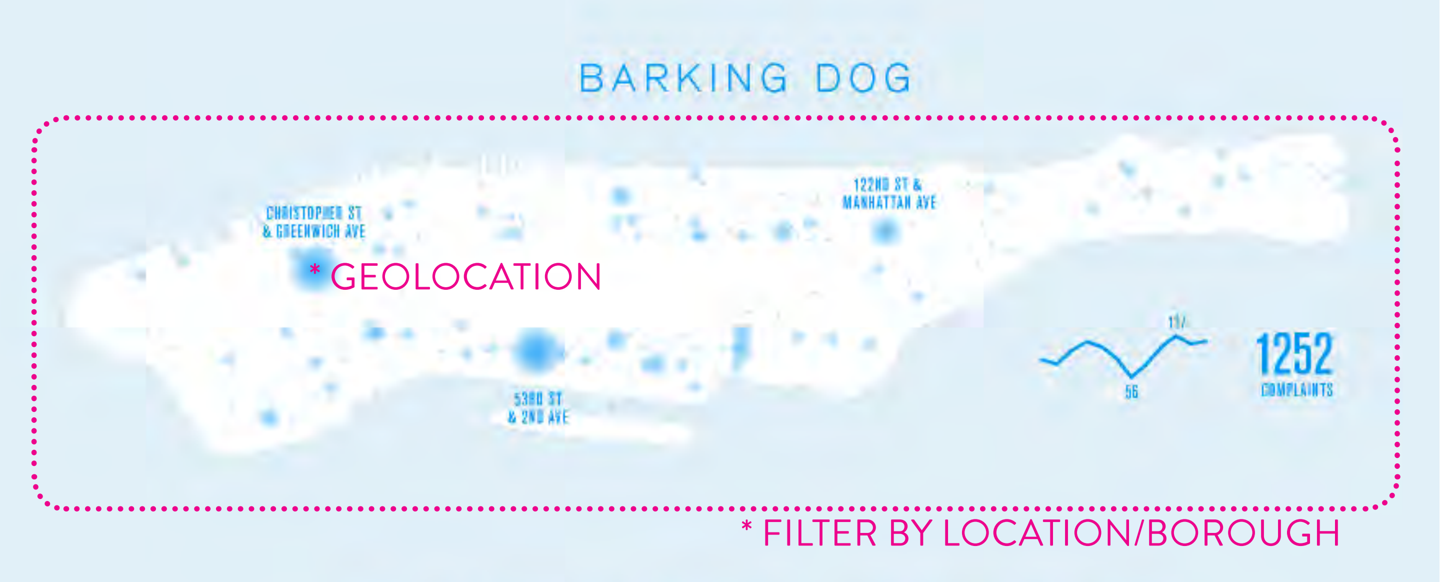


311 EXCERPT: BARKING DOG CALLS FROM 2012

FAMILIAR FEATURES:

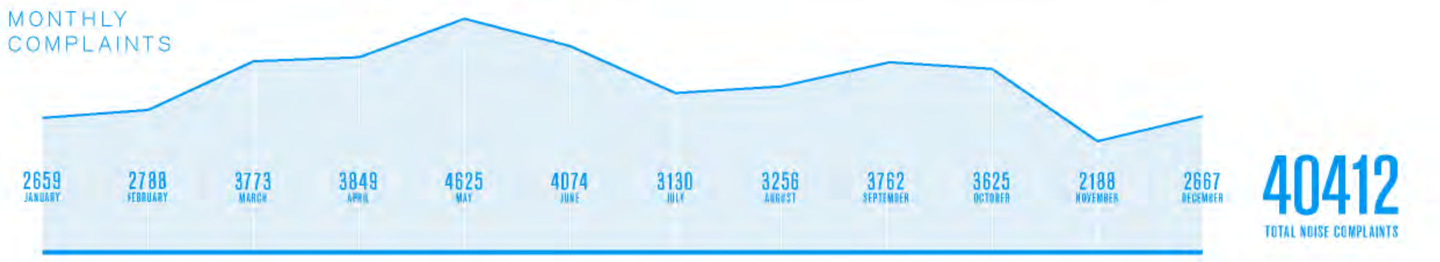
- geolocation of complaint calls by lat/long and/or NYC street grid
- filtering of all 311 calls to get dog barking complaints for Manhattan

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



* GEOLOCATION

* FILTER BY LOCATION/BOROUGH



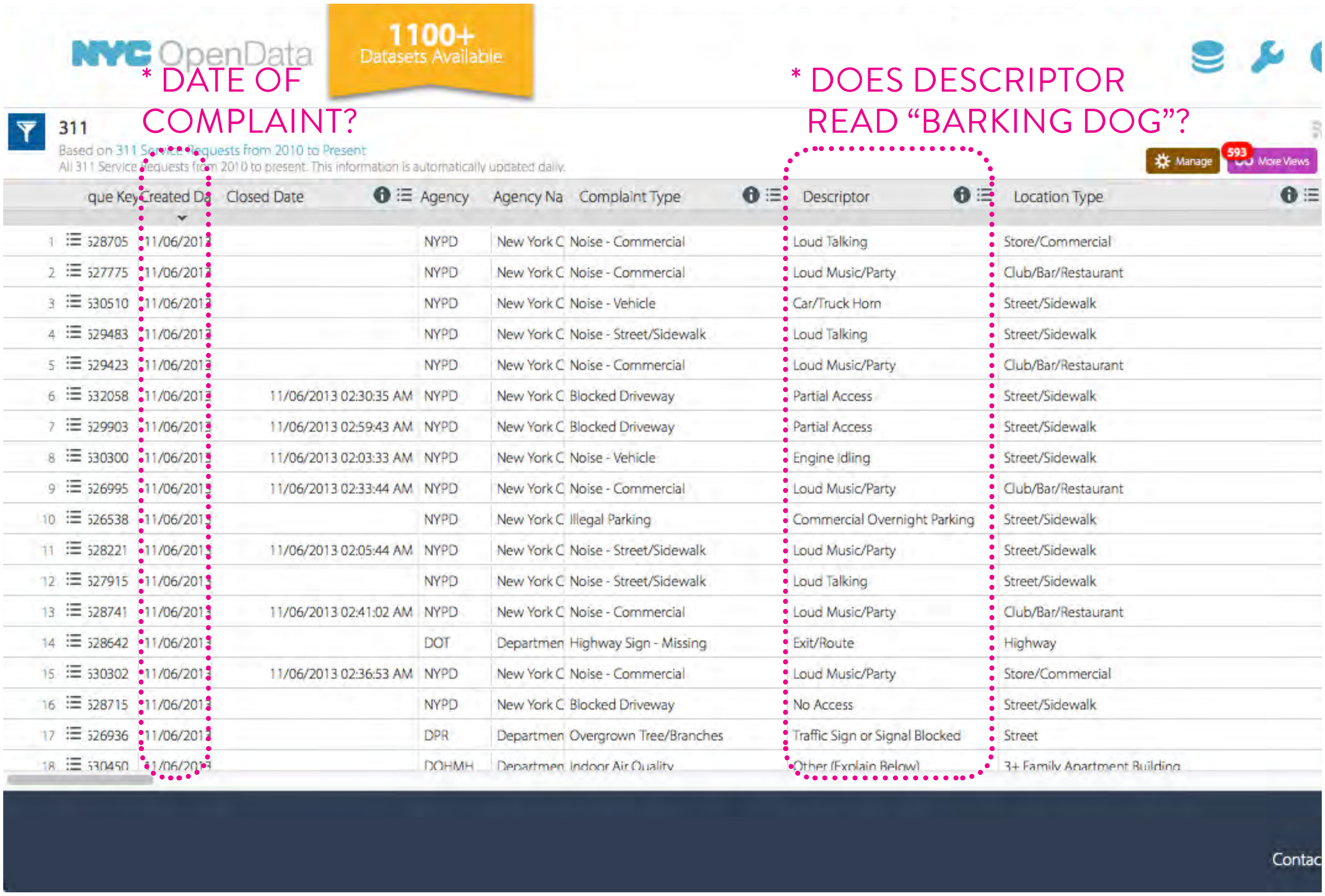
311 EXCERPT: GEOLOCATION & FILTERING

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



* DATE OF COMPLAINT?

* DOES DESCRIPTOR READ "BARKING DOG"?

311

Based on 311 Service Requests from 2010 to Present

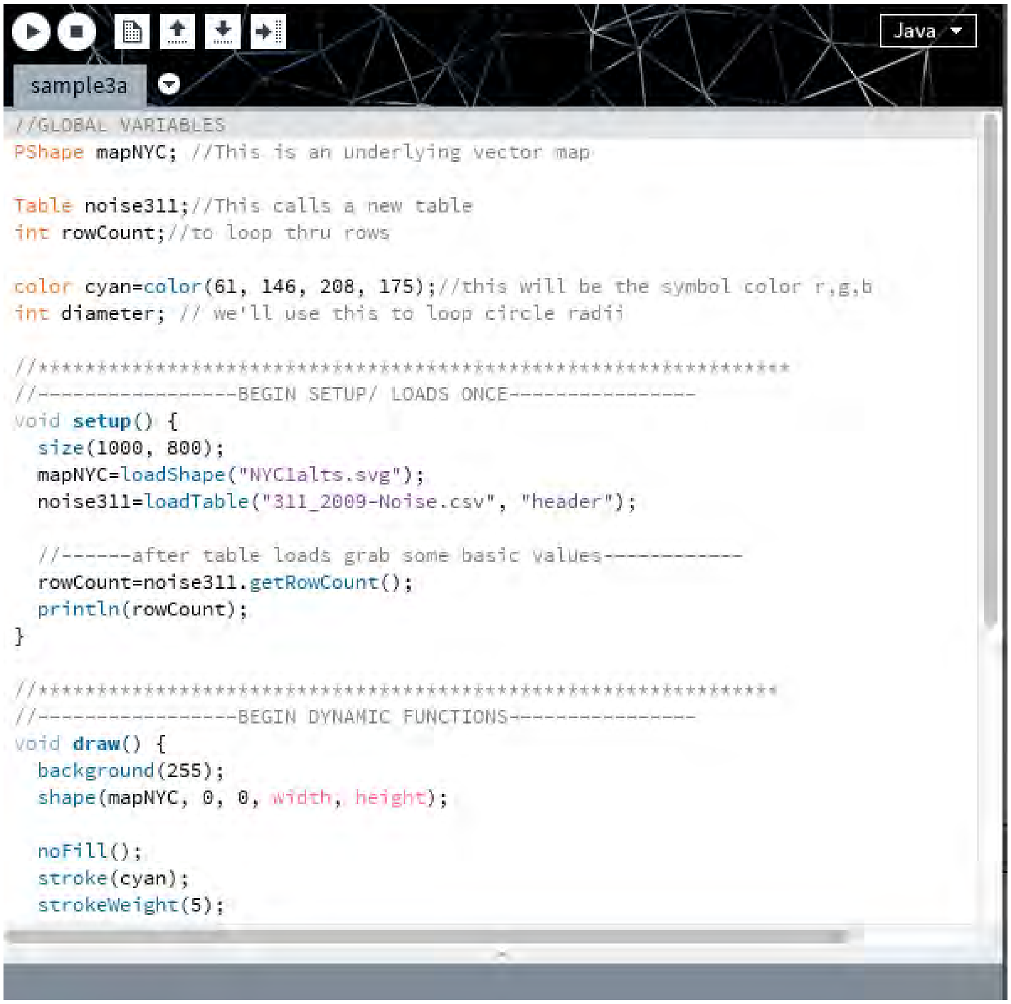
All 311 Service Requests from 2010 to present. This information is automatically updated daily.

	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

311 EXCERPT: GEOLOCATION & 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"



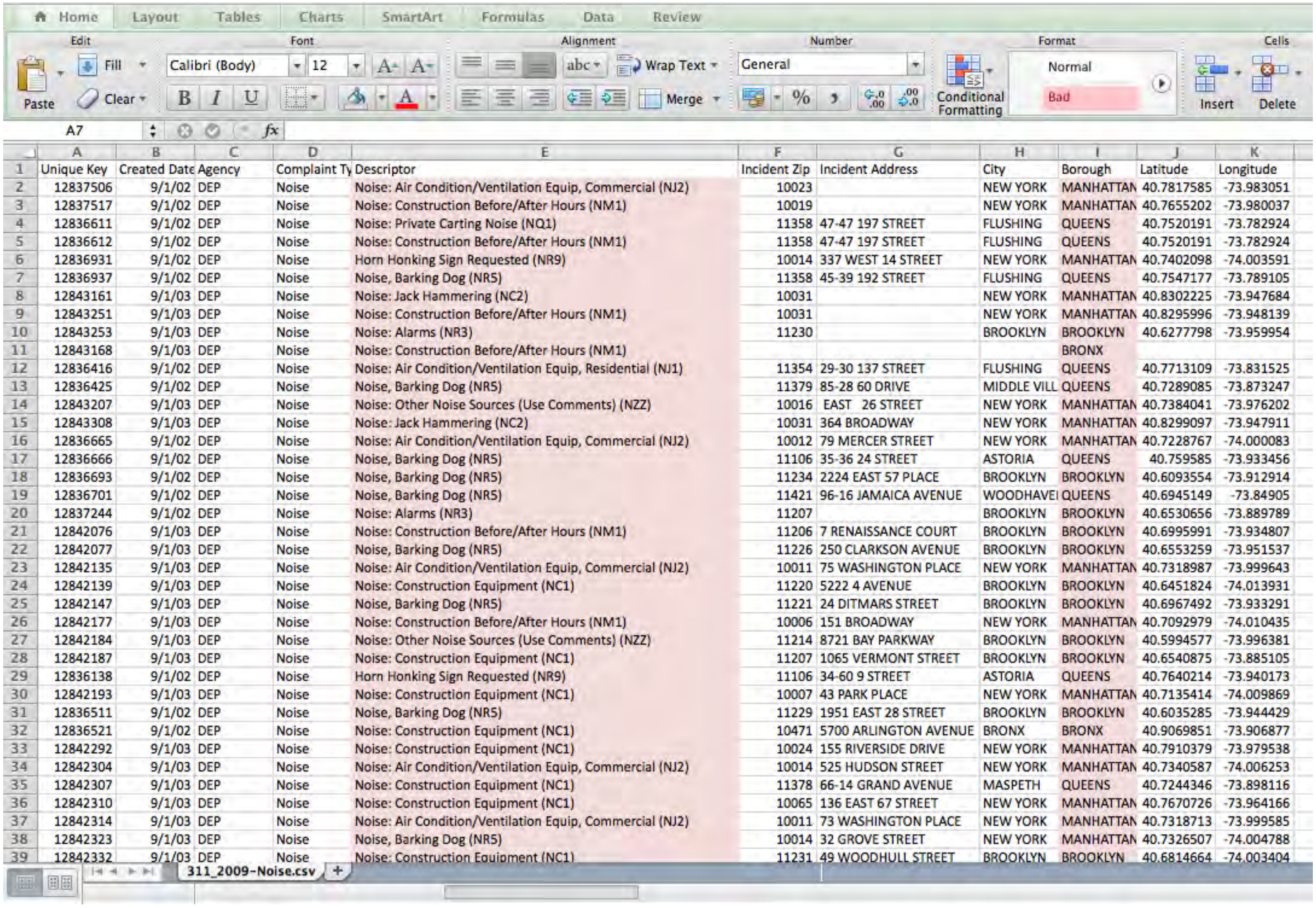
```
//GLOBAL VARIABLES
PShape mapNYC; //This is an underlying vector map

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.



	A	B	C	D	E	F	G	H	I	J	K
1	Unique Key	Created Date	Agency	Complaint Ty	Descriptor	Incident Zip	Incident Address	City	Borough	Latitude	Longitude
2	12837506	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10023		NEW YORK	MANHATTAN	40.7817585	-73.983051
3	12837517	9/1/02	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10019		NEW YORK	MANHATTAN	40.7655202	-73.980037
4	12836611	9/1/02	DEP	Noise	Noise: Private Carting Noise (NQ1)	11358	47-47 197 STREET	FLUSHING	QUEENS	40.7520191	-73.782924
5	12836612	9/1/02	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11358	47-47 197 STREET	FLUSHING	QUEENS	40.7520191	-73.782924
6	12836931	9/1/02	DEP	Noise	Horn Honking Sign Requested (NR9)	10014	337 WEST 14 STREET	NEW YORK	MANHATTAN	40.7402098	-74.003591
7	12836937	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11358	45-39 192 STREET	FLUSHING	QUEENS	40.7547177	-73.789105
8	12843161	9/1/03	DEP	Noise	Noise: Jack Hammering (NC2)	10031		NEW YORK	MANHATTAN	40.8302225	-73.947684
9	12843251	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10031		NEW YORK	MANHATTAN	40.8295996	-73.948139
10	12843253	9/1/03	DEP	Noise	Noise: Alarms (NR3)	11230		BROOKLYN	BROOKLYN	40.6277798	-73.959954
11	12843168	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)				BRONX		
12	12836416	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Residential (NJ1)	11354	29-30 137 STREET	FLUSHING	QUEENS	40.7713109	-73.831525
13	12836425	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11379	85-28 60 DRIVE	MIDDLE VILL	QUEENS	40.7289085	-73.873247
14	12843207	9/1/03	DEP	Noise	Noise: Other Noise Sources (Use Comments) (NZZ)	10016	EAST 26 STREET	NEW YORK	MANHATTAN	40.7384041	-73.976202
15	12843308	9/1/03	DEP	Noise	Noise: Jack Hammering (NC2)	10031	364 BROADWAY	NEW YORK	MANHATTAN	40.8299097	-73.947911
16	12836665	9/1/02	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10012	79 MERCER STREET	NEW YORK	MANHATTAN	40.7228767	-74.000083
17	12836666	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11106	35-36 24 STREET	ASTORIA	QUEENS	40.759585	-73.933456
18	12836693	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11234	2224 EAST 57 PLACE	BROOKLYN	BROOKLYN	40.6093554	-73.912914
19	12836701	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11421	96-16 JAMAICA AVENUE	WOODHAVEI	QUEENS	40.6945149	-73.84905
20	12837244	9/1/02	DEP	Noise	Noise: Alarms (NR3)	11207		BROOKLYN	BROOKLYN	40.6530656	-73.889789
21	12842076	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	11206	7 RENAISSANCE COURT	BROOKLYN	BROOKLYN	40.6995991	-73.934807
22	12842077	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	11226	250 CLARKSON AVENUE	BROOKLYN	BROOKLYN	40.6553259	-73.951537
23	12842135	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10011	75 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318987	-73.999643
24	12842139	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11220	5222 4 AVENUE	BROOKLYN	BROOKLYN	40.6451824	-74.013931
25	12842147	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	11221	24 DITMARS STREET	BROOKLYN	BROOKLYN	40.6967492	-73.933291
26	12842177	9/1/03	DEP	Noise	Noise: Construction Before/After Hours (NM1)	10006	151 BROADWAY	NEW YORK	MANHATTAN	40.7092979	-74.010435
27	12842184	9/1/03	DEP	Noise	Noise: Other Noise Sources (Use Comments) (NZZ)	11214	8721 BAY PARKWAY	BROOKLYN	BROOKLYN	40.5994577	-73.996381
28	12842187	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11207	1065 VERMONT STREET	BROOKLYN	BROOKLYN	40.6540875	-73.885105
29	12836138	9/1/02	DEP	Noise	Horn Honking Sign Requested (NR9)	11106	34-60 9 STREET	ASTORIA	QUEENS	40.7640214	-73.940173
30	12842193	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10007	43 PARK PLACE	NEW YORK	MANHATTAN	40.7135414	-74.009869
31	12836511	9/1/02	DEP	Noise	Noise, Barking Dog (NR5)	11229	1951 EAST 28 STREET	BROOKLYN	BROOKLYN	40.6035285	-73.944429
32	12836521	9/1/02	DEP	Noise	Noise: Construction Equipment (NC1)	10471	5700 ARLINGTON AVENUE	BRONX	BRONX	40.9069851	-73.906877
33	12842292	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10024	155 RIVERSIDE DRIVE	NEW YORK	MANHATTAN	40.7910379	-73.979538
34	12842304	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10014	525 HUDSON STREET	NEW YORK	MANHATTAN	40.7340587	-74.006253
35	12842307	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11378	66-14 GRAND AVENUE	MASPETH	QUEENS	40.7244346	-73.898116
36	12842310	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	10065	136 EAST 67 STREET	NEW YORK	MANHATTAN	40.7670726	-73.964166
37	12842314	9/1/03	DEP	Noise	Noise: Air Condition/Ventilation Equip, Commercial (NJ2)	10011	73 WASHINGTON PLACE	NEW YORK	MANHATTAN	40.7318713	-73.999585
38	12842323	9/1/03	DEP	Noise	Noise, Barking Dog (NR5)	10014	32 GROVE STREET	NEW YORK	MANHATTAN	40.7326507	-74.004788
39	12842332	9/1/03	DEP	Noise	Noise: Construction Equipment (NC1)	11231	49 WOODHULL STREET	BROOKLYN	BROOKLYN	40.6814664	-74.003404

311 EXCERPT: GEOLOCATION & FILTERING

IN EXCEL, YOU'VE LIKELY DONE SOMETHING SIMILAR:

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

311 EXCERPT: **VARIABLES FOR HOLDING INFO**

VARIABLES types:

variable definition

variable example

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

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

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

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

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

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

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

long =a longer version of interger with over 8 places of info
ex. long x=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]

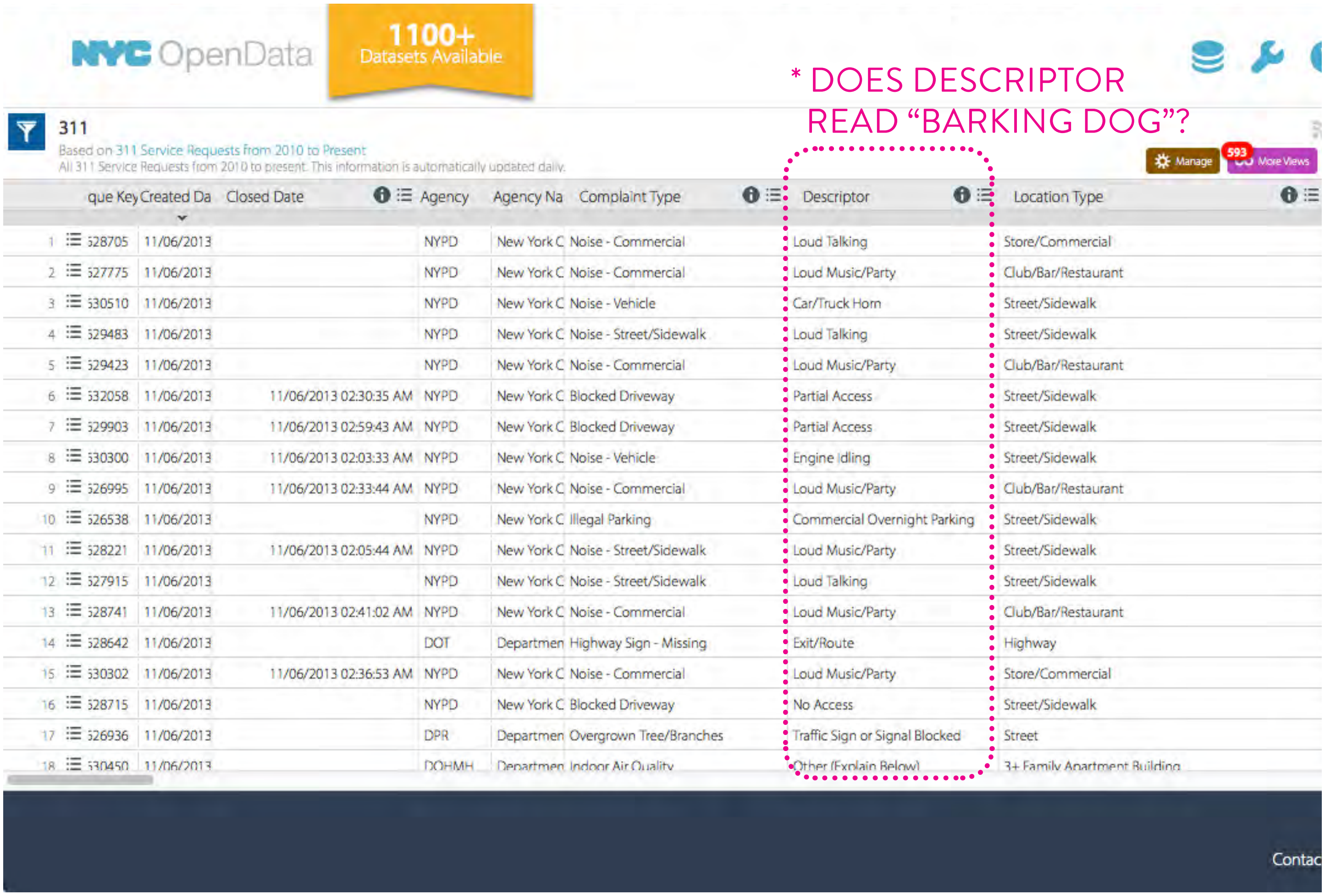
ex.

```
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 exsiting value
z=x+y;
// here int z=5;
```

a few notes on syntax:

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

variables nest and can be dynamically redefined to build complexity



* DOES DESCRIPTOR READ “BARKING DOG”?

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

311 EXCERPT: **CONDITIONALS, CODING USE**

CONDITIONAL USE: *coding syntax*

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

Does the column title read “Longitude”, if so...?

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

311 EXCERPT: **CONDITIONALS FOR FILTERING INFO**

CONDITIONAL types:

conditional definition
conditional example

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

ex.

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

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

ex.

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

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

ex.

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

switch= choses between finite options (not often used)

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

311 EXCERPT: **OPERATORS FOR CONDITIONALS**

OPERATOR types: *operator definition*

operator examples (true statements)

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

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

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

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

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

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

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

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

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

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

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

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

311 EXCERPT: **MORE ON CODING SYNTAX**

CODING SYNTAX: *more basics*

( ) (parentheses) mathematical use, brackets for variables and statements

```
ex. float x=(32-21.45)/5.25;
```

[ ] (square brackets) for array use, allows more than one variable

```
ex. float [ ] x=new float[ 3];
        x[0]=.25;
        x[1]=.50;
        x[2]=.75;

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

{ } (curly brackets) encloses a block of code, like conditionals, arrays, functions

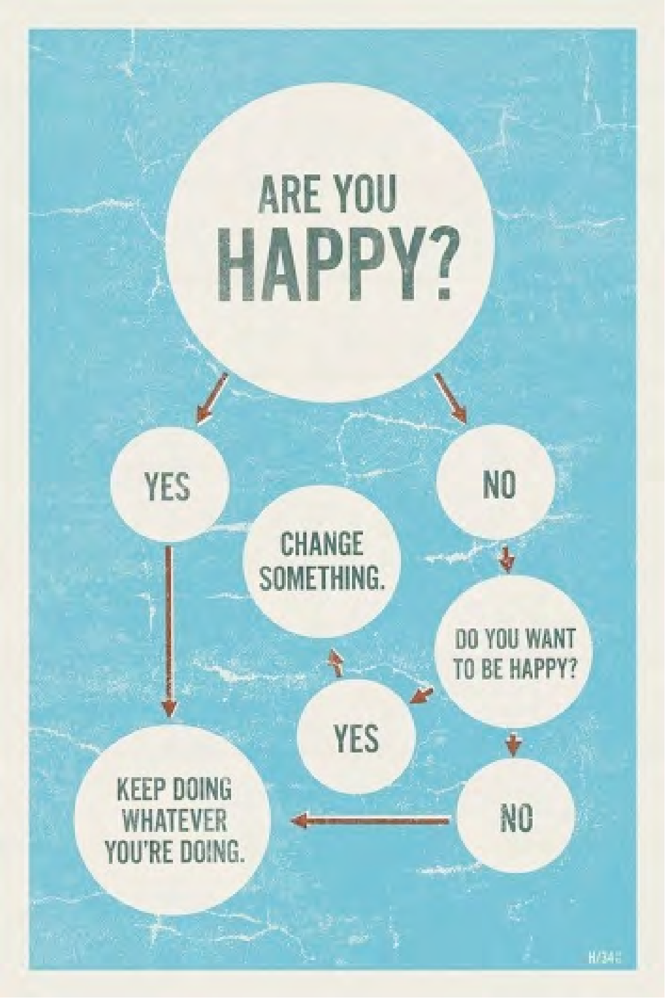
```
ex. if (x==45){
        String y="true";
        }  // conditional

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

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

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

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

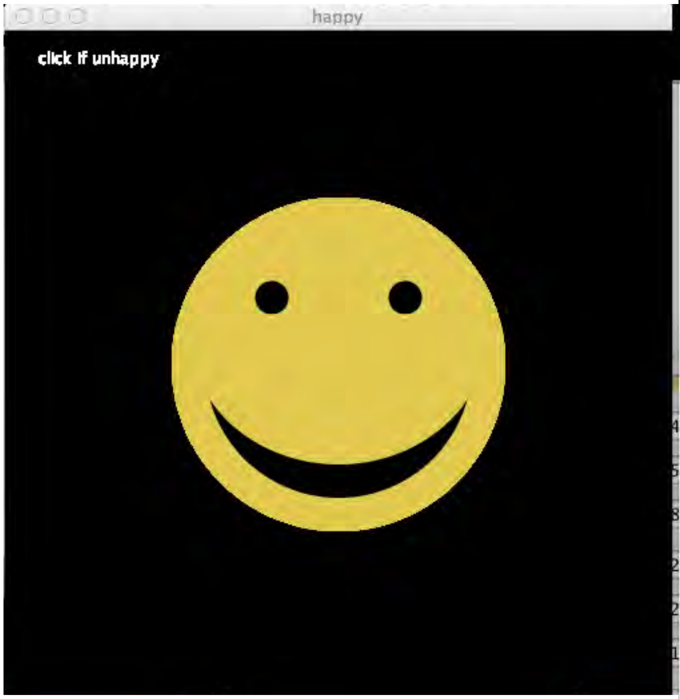
find @ http://www.processing.org/references/

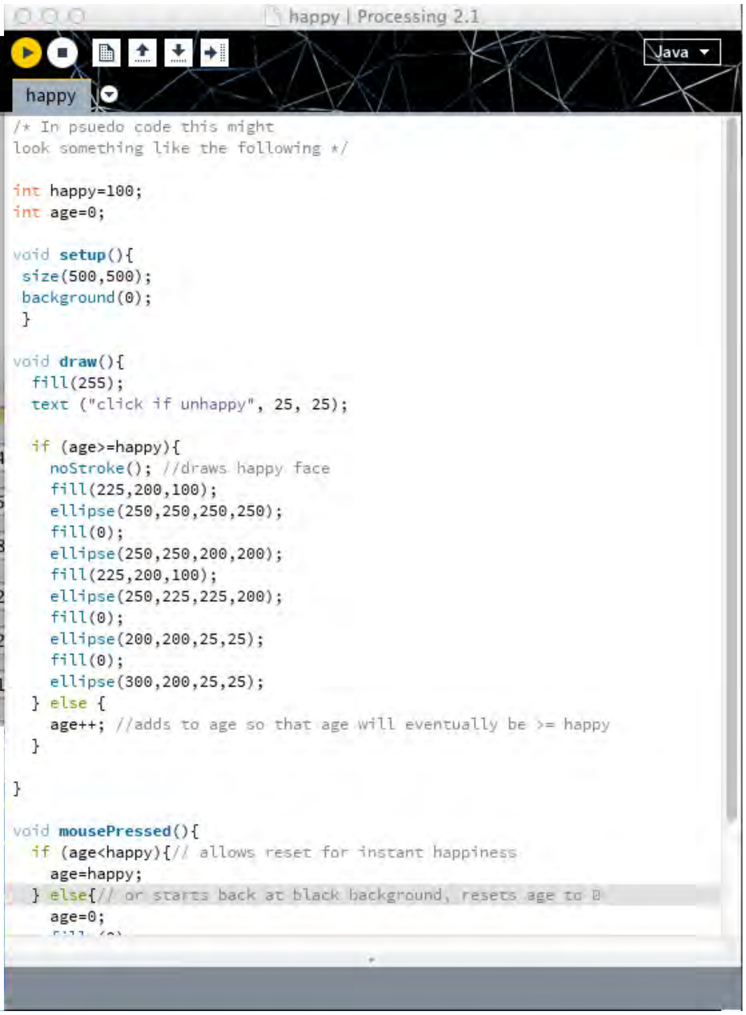


CONCEPTUAL STRUCTURE: **CODE STRUCTURE**

Processing is designed to give visual feedback for learning code.

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





```
/* 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;
```

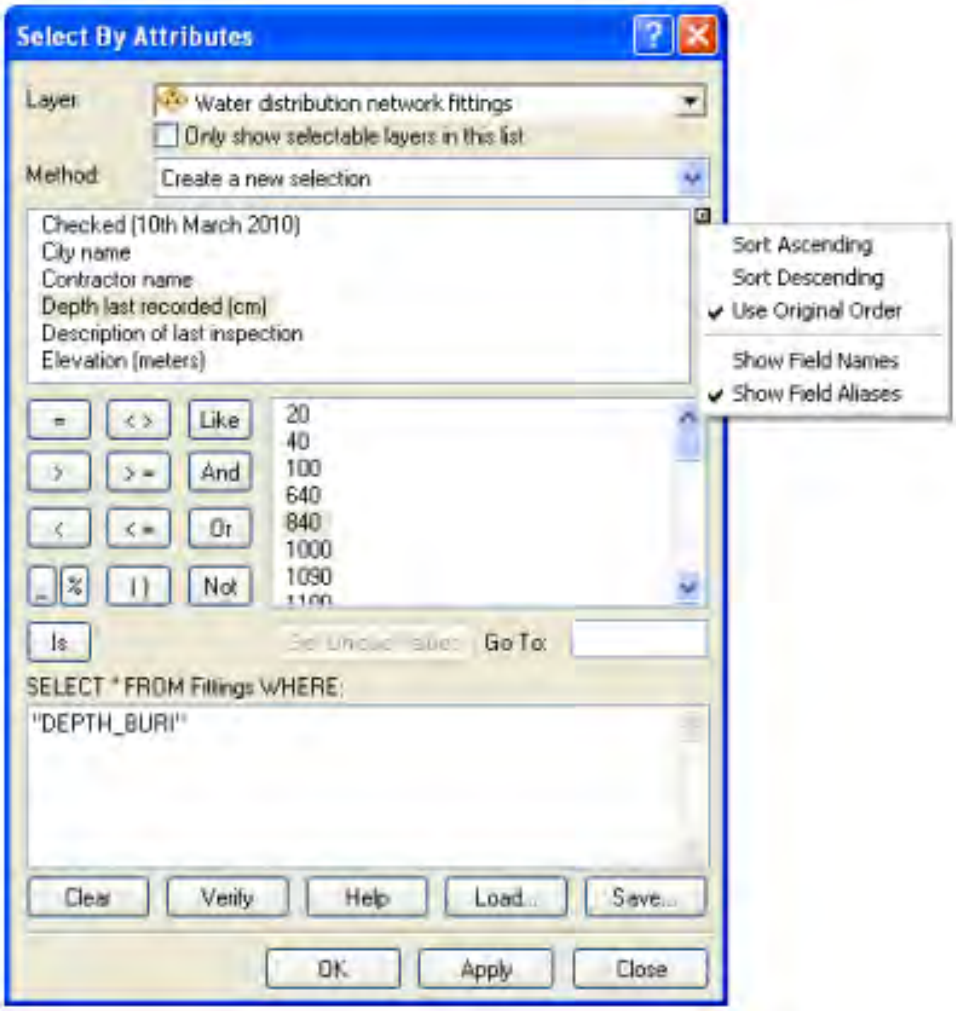
INPUT: **THE CODE BEHIND THE VISUALS**

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

Explanations will by typed in this area below.

This has a few benefits, despite being a pain:

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



Select By Attributes

Layer: Water distribution network fittings

□ Only show selectable layers in this list

Method: Create a new selection

Checked (10th March 2010)
City name
Contractor name
Depth last recorded (cm)
Description of last inspection
Elevation (meters)

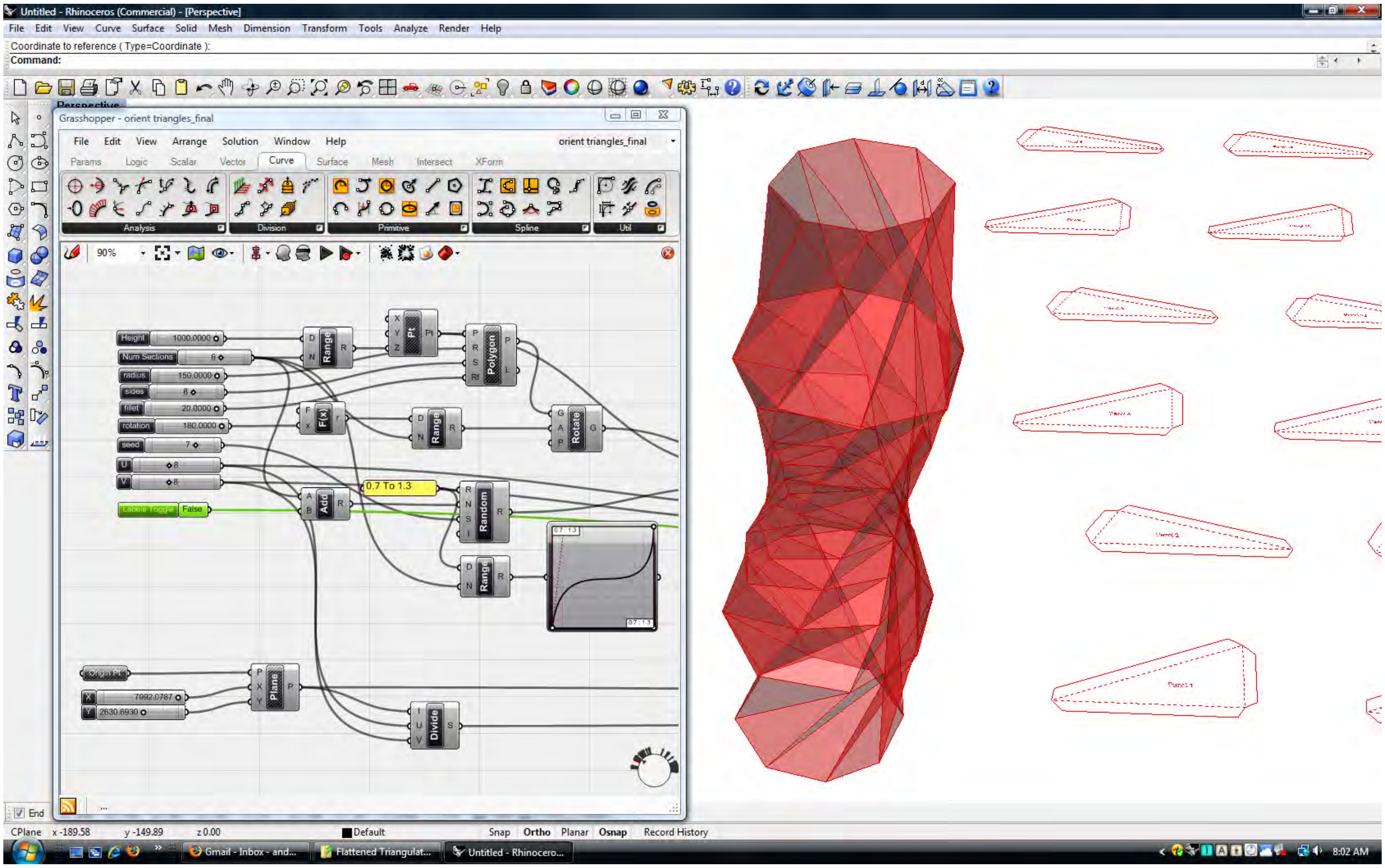
20
40
100
640
840
1000
1090
1100

SELECT * FROM Fittings WHERE:

"DEPTH_BURI"

FIELD FILTERING **SECTIONS FROM TABLES**

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



Coordinate to reference (Type=Coordinate):
Command:

Height	1000.0000
Num Sections	6
radius	150.0000
sides	6
fillet	20.0000
rotation	180.0000
seed	7
U	8
V	8
Labels Toggle	False
Origin Pt	
X	7992.0787
Y	2630.6930

Untitled - Rhinoceros (Commercial) - [Perspective]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Help
Perspective
Grasshopper - orient triangles_final
File Edit View Arrange Solution Window Help
orient triangles_final
Params Logic Scalar Vector Curve Surface Mesh Intersect XForm
Analysis Division Primitive Spline Util
90%
Range
Pt
Polygon
F(x)
Rotate
0.7 To 1.3
Add
Random
Plane
Divide
End
CPlane x -189.58 y -149.89 z 0.00 Default Snap Ortho Planar Osnap Record History
Gmail - Inbox - and... Flattened Triangulat... Untitled - Rhinocero...
8:02 AM

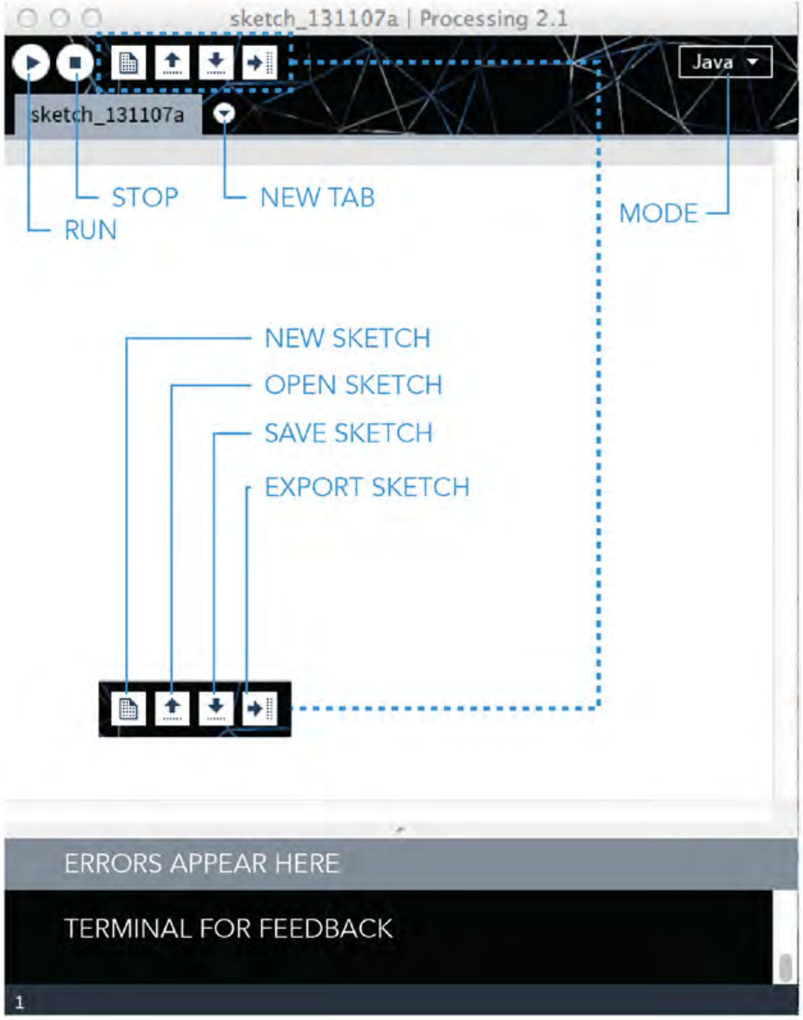
PARAMETER ADJUSTMENT **GRASSHOPPER OBJECTS**

Basic re-working of object parameters for formal manipulations.

WK 1

BASIC PROCESSING FOR PLOTTING

- interface
- shapes
- display
- loops



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

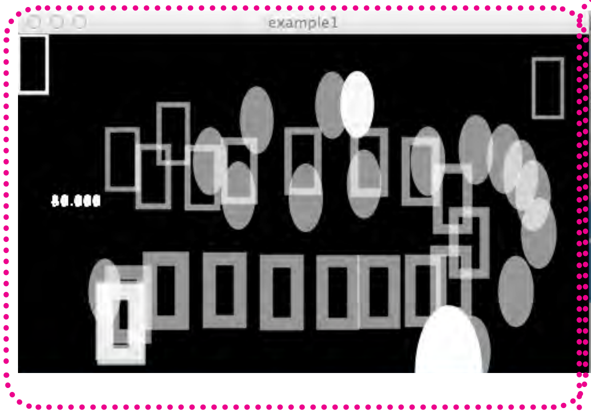
PROCESSING/PREFERENCES

FILE/SKETCHBOOK FILE/SAMPLES

EDIT/FORMAT EDIT/COMMENT

SKETCH/IMPORT LIBRARY

TOOLS/CREATE FONT TOOLS/COLOR SELECTOR



* APP OUTPUT

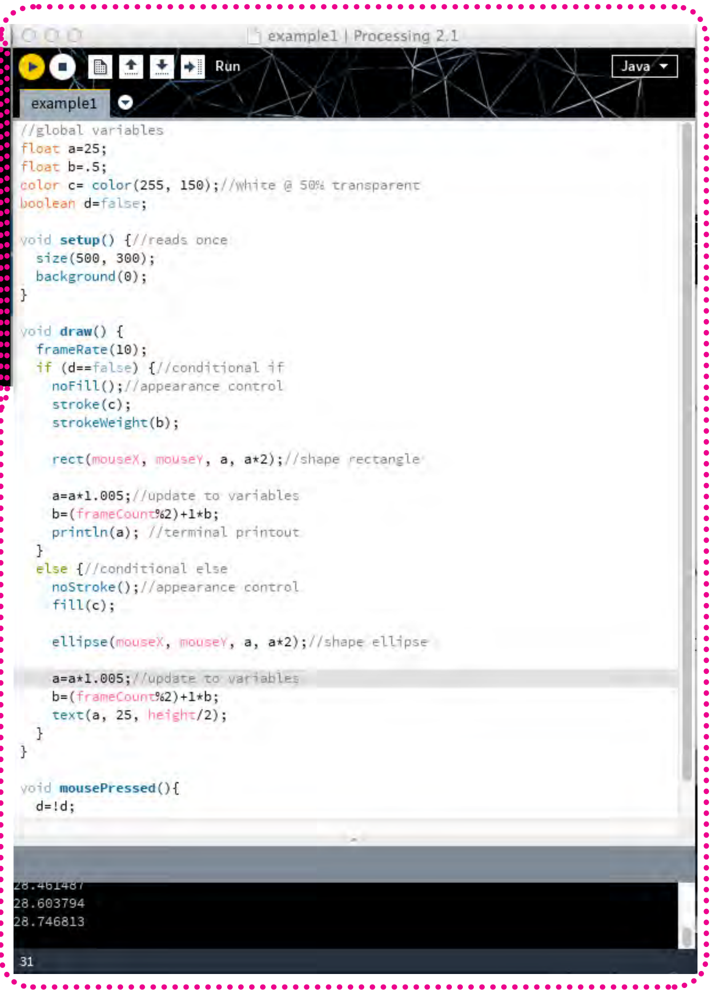
HIGHLIGHTS*

ORANGE-VARIABLE TYPES

BLUE- PREDEFINED FUNCTIONS

PINK- PREDEFINED SPATIAL/TEMPORAL PARAMETERS

ADJUST IN PREFERENCES



```
//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;
```

```
28.461487
28.603794
28.746813
```

31

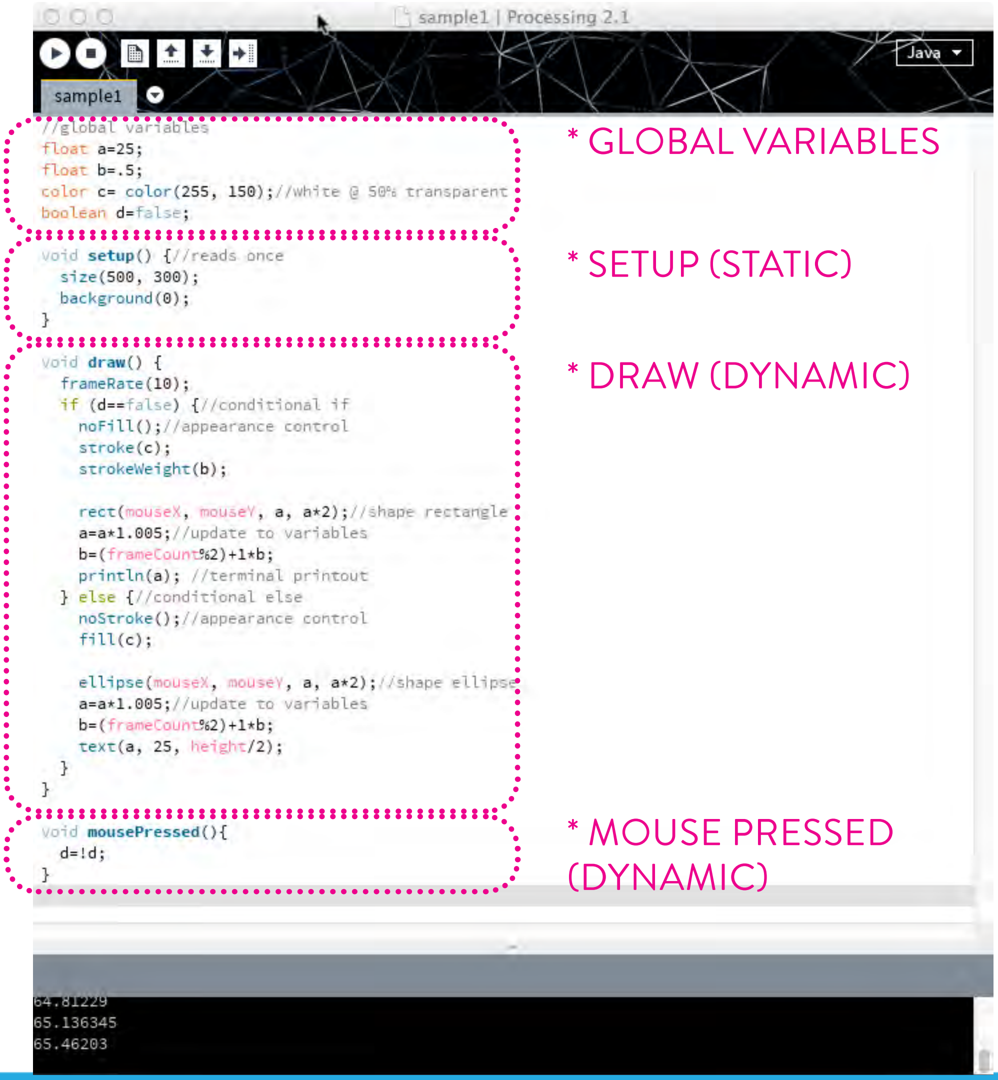
* PDE PROCESSING DEVELOPMENT ENVIRONMENT

SKETCH STRUCTURE **WORKING IN PDE**

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

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

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



```
//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
```

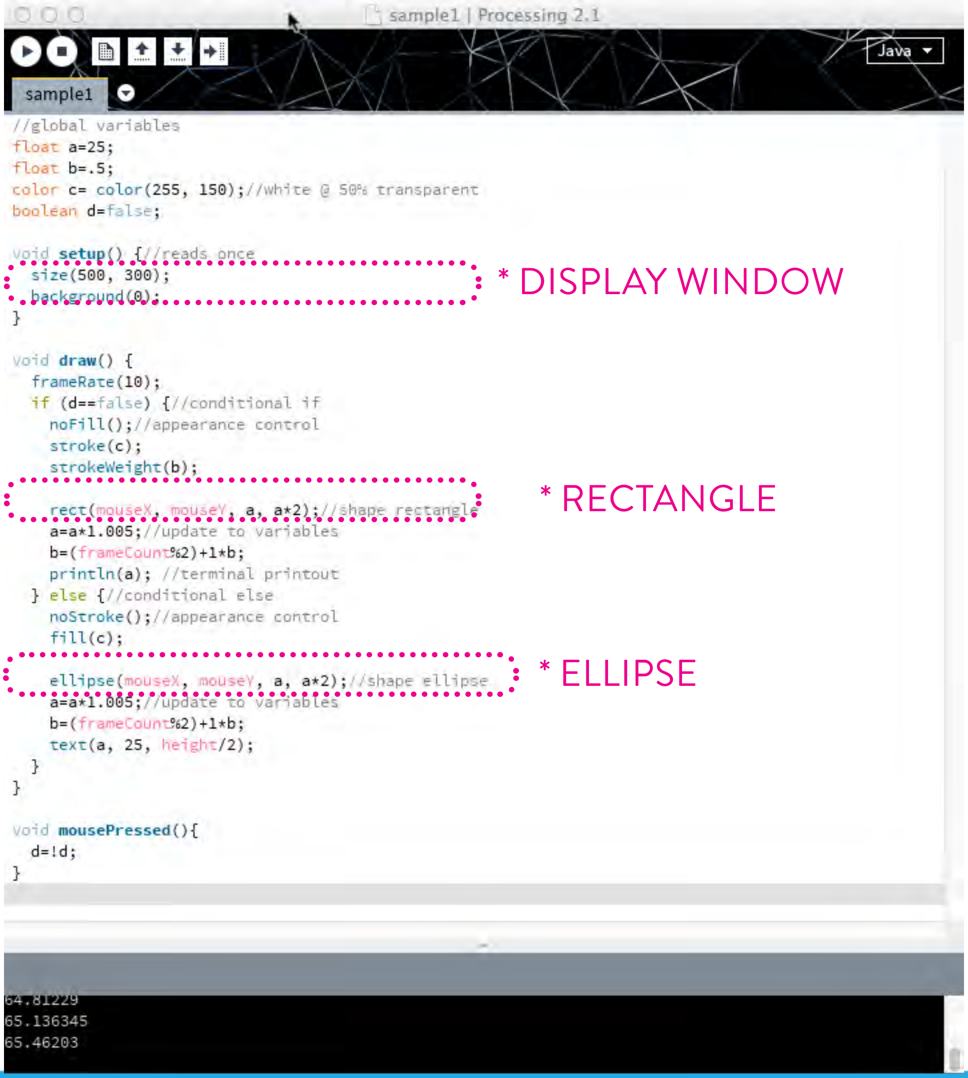
SKETCH: EXAMPLE 1 **PARTS/ORDER**

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

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

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

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



```
//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
```

* DISPLAY WINDOW

* RECTANGLE

* ELLIPSE

SKETCH: EXAMPLE 1 **SHAPES**

This simple sketch has two shapes drawn against a background.

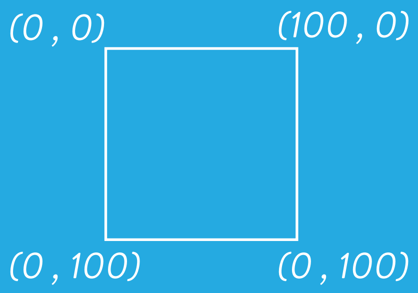
size (width, height); defines the window for the drawing. Background (0) defines its' color (black). These are static and do not refresh.

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

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

SKETCH: EXAMPLE 1 **PRIMITIVE SHAPES**

SPACE in processing: (x coordinate , y coordinate)



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

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

SHAPE types: *shape definition*

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

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

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

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

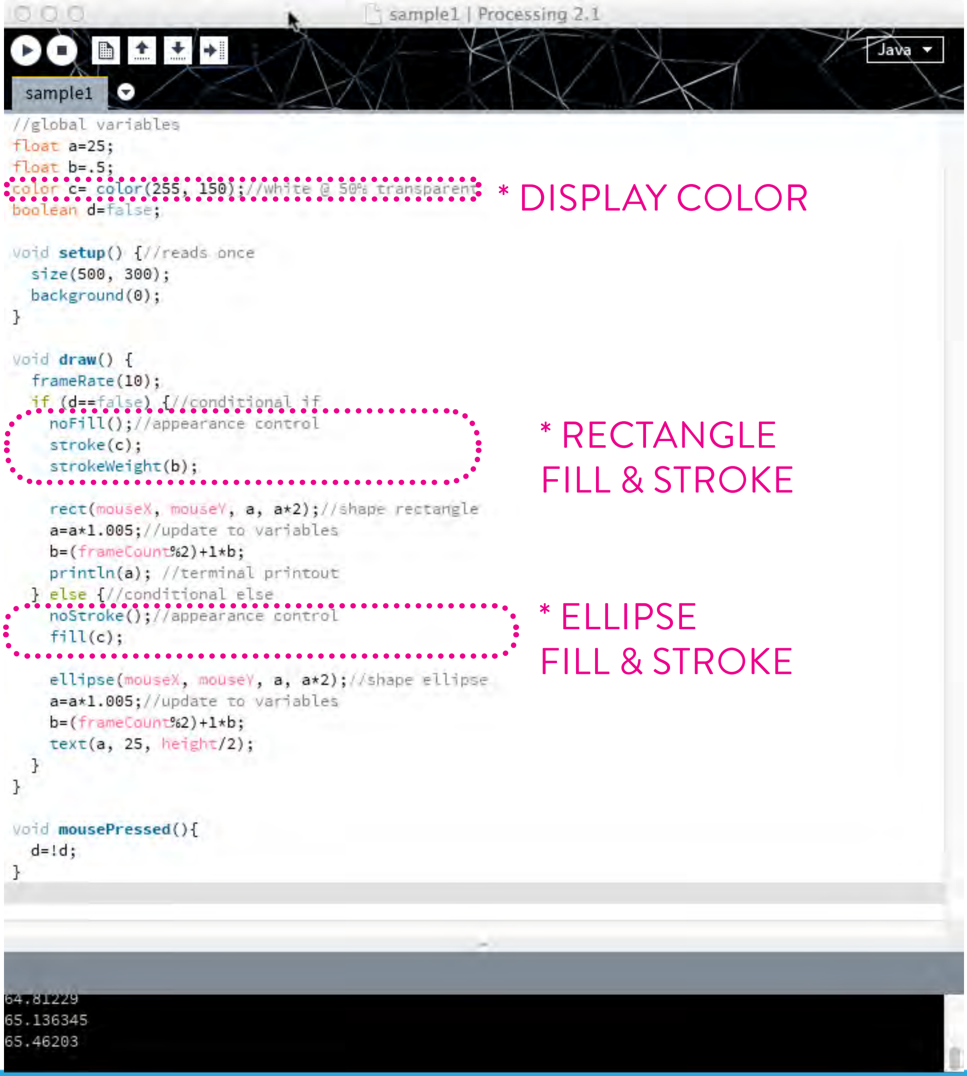
ellipse (x, y, diameter, diameter); ellipse, default centered

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

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

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

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



```
//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
```

* DISPLAY COLOR

* RECTANGLE
FILL & STROKE

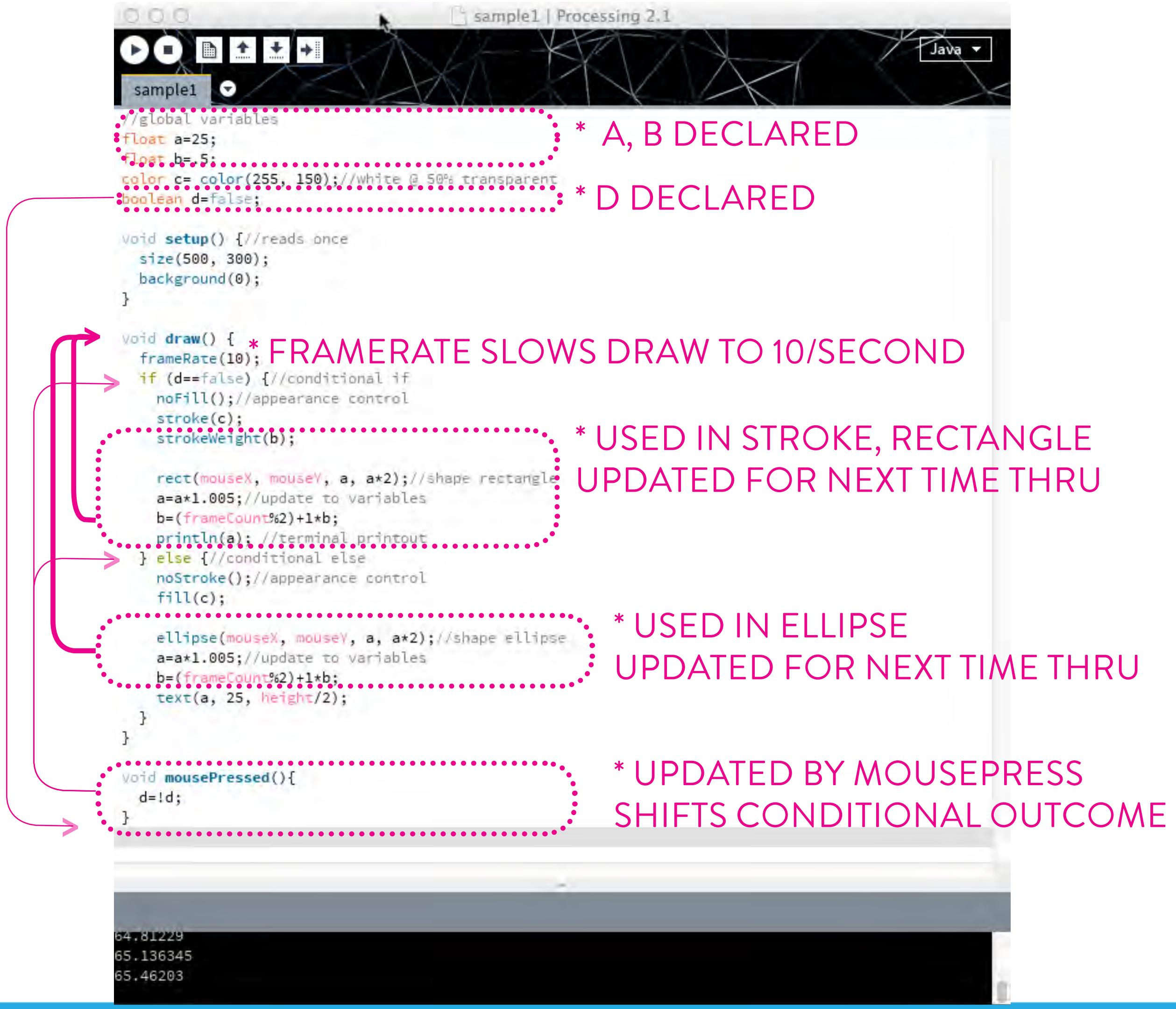
* ELLIPSE
FILL & STROKE

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;
}
```

```
64.81229
65.136345
65.46203
```

* A, B DECLARED

* D DECLARED

* FRAMERATE SLOWS DRAW TO 10/SECOND

* USED IN STROKE, RECTANGLE UPDATED FOR NEXT TIME THRU

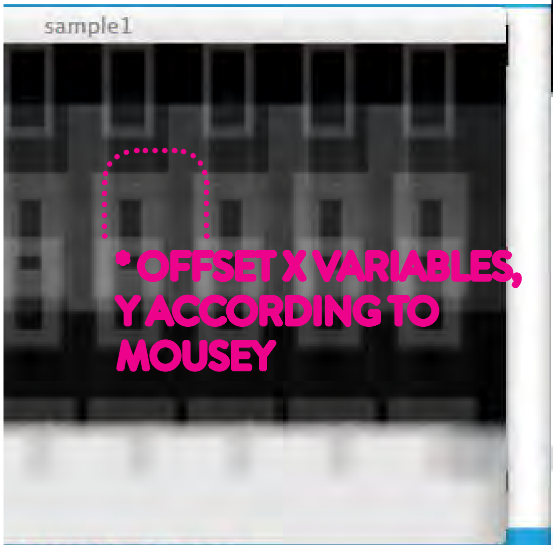
* USED IN ELLIPSE UPDATED FOR NEXT TIME THRU

* UPDATED BY MOUSEPRESS SHIFTS CONDITIONAL OUTCOME

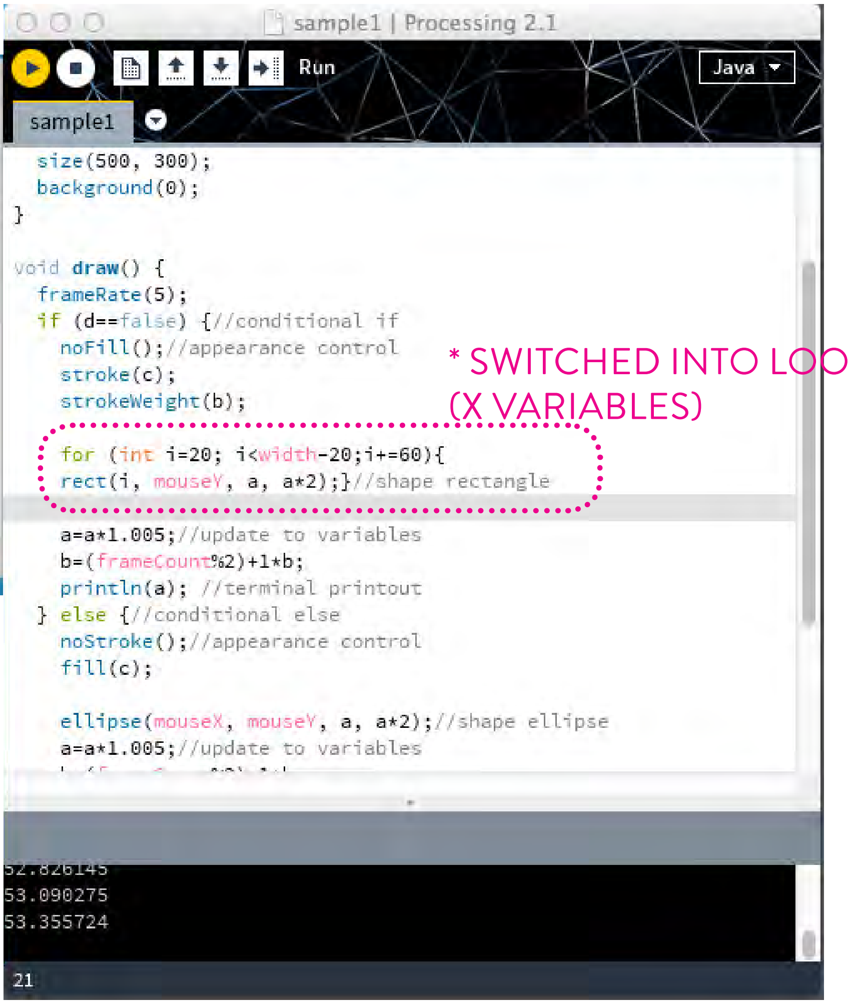
SKETCH: EXAMPLE 1 **DYNAMIC UPDATES**

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 is strokeWeight and figure size, but you could also reset variables back to their original value for a stable drawing.

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



* OFFSET X VARIABLES, Y ACCORDING TO MOUSEY



```
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(5);
  if (d==false) {//conditional if
    noFill();//appearance control
    stroke(c);
    strokeWeight(b);

    for (int i=20; i<width-20;i+=60){
    rect(i, 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
```

```
52.826145
53.090275
53.355724
```

* SWITCHED INTO LOOP (X VARIABLES)

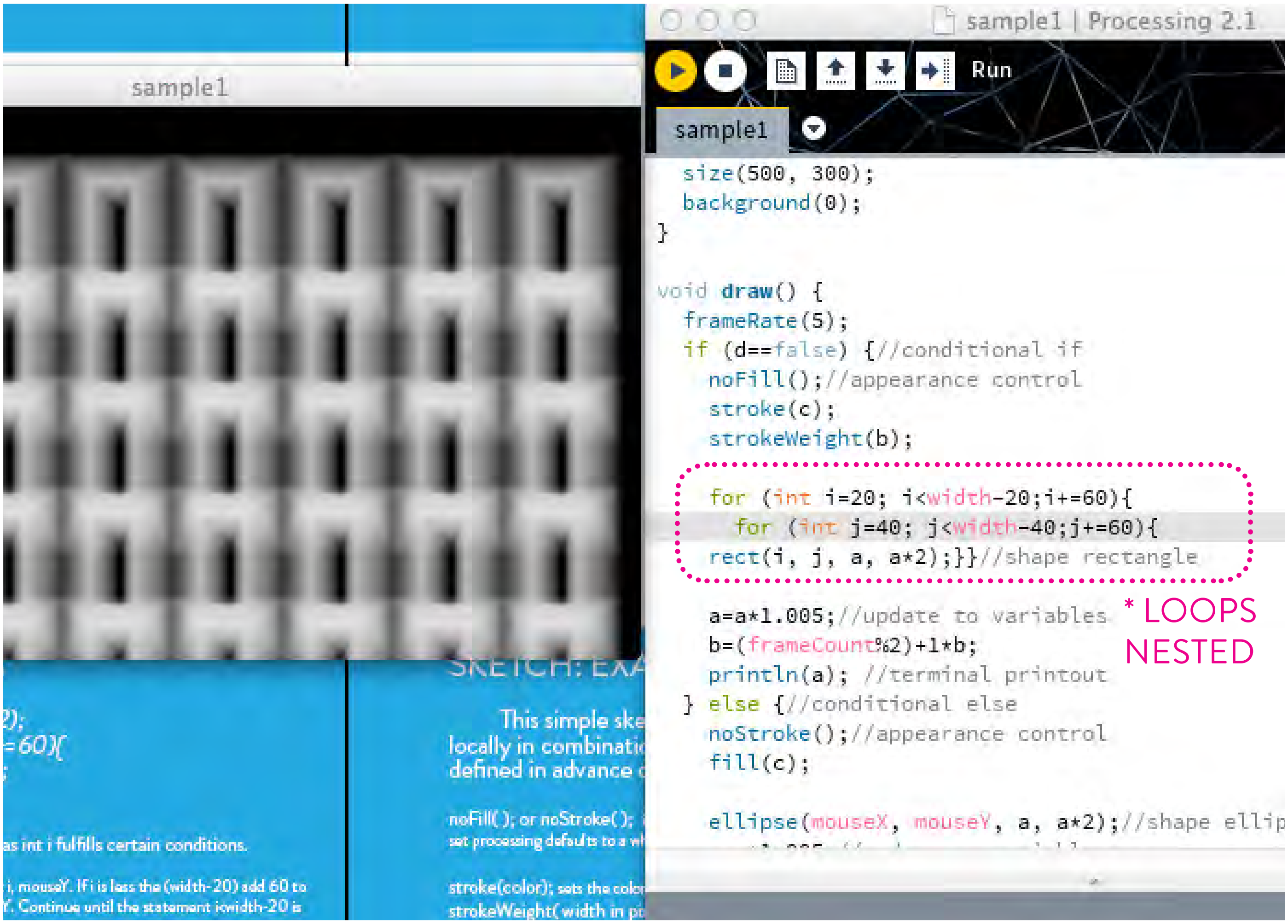
SKETCH: EXAMPLE 1B **LOOPS**

change *rect(mouseX,mouseY, a, a*2);*

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

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

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


sample1
sample1 | Processing 2.1
Run
sample1
SKETCH: EXA
2);
=60){
;
as int i fulfills certain conditions.
i, mouseY. If i is less the (width-20) add 60 to
Y. Continue until the statement i<width-20 is
This simple ske
locally in combinatio
defined in advance o
noFill( ); or noStroke( );
set processing defaults to a wh
stroke(color); sets the color
strokeWeight(width in po
* LOOPS NESTED

```
  size(500, 300);
  background(0);
}

void draw() {
  frameRate(5);
  if (d==false) {//conditional if
    noFill();//appearance control
    stroke(c);
    strokeWeight(b);

    for (int i=20; i<width-20;i+=60){
      for (int j=40; j<width-40;j+=60){
    rect(i, j, 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 ellip
```

SKETCH: EXAMPLE 1B **LOOPS NESTED**

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

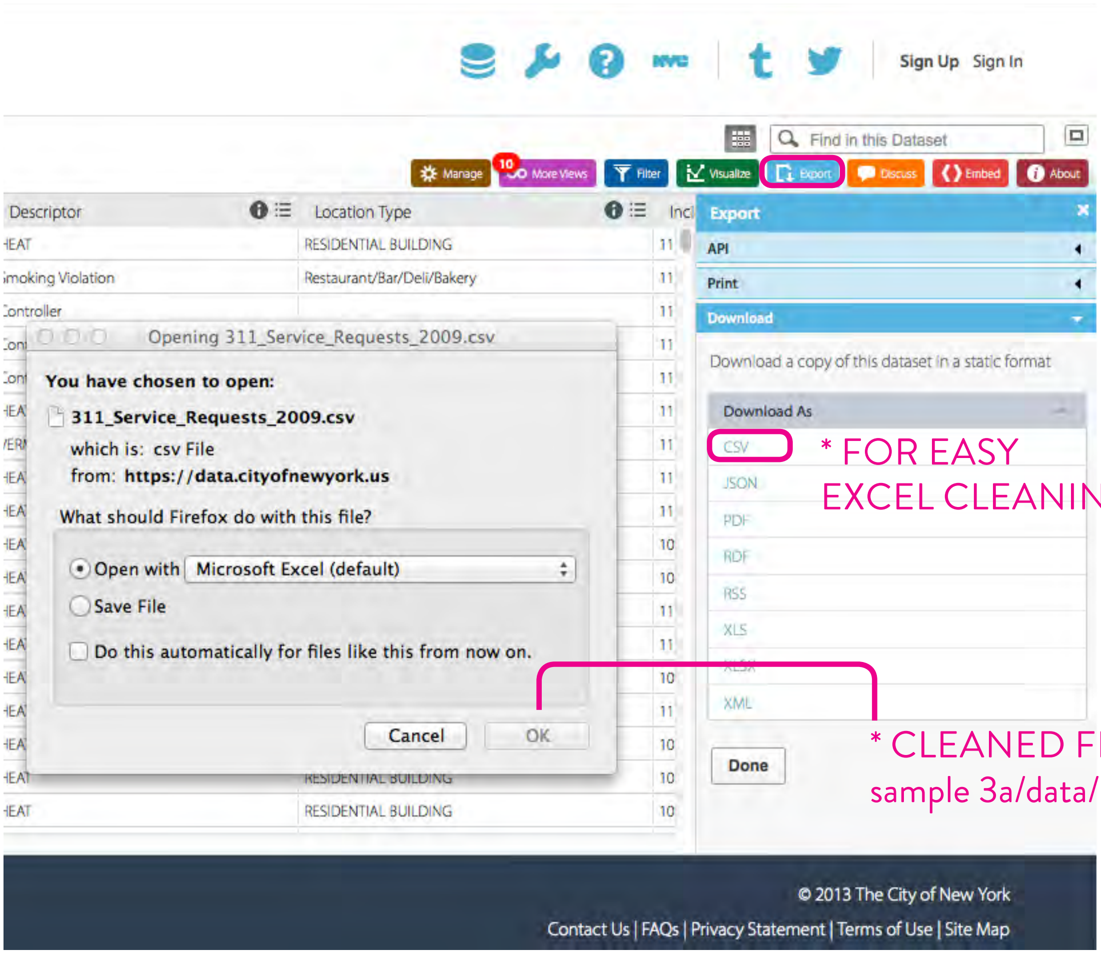
The grid of rectangles that results draws a series of columns of rectangle across the page. For i=20, it executes all at j positions and then moves to the next i variable, i=80... and so on.

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

WK 1

RECONSTRUCTING 311 EXAMPLE

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



Descriptor	Location Type
HEAT	RESIDENTIAL BUILDING
Smoking Violation	Restaurant/Bar/Deli/Bakery
Controller	
HEAT	RESIDENTIAL BUILDING
HEAT	RESIDENTIAL BUILDING

Opening 311_Service_Requests_2009.csv

You have chosen to open:

311_Service_Requests_2009.csv

which is: csv File

from: https://data.cityofnewyork.us

What should Firefox do with this file?

● Open with Microsoft Excel (default)

○ Save File

□ Do this automatically for files like this from now on.

Download a copy of this dataset in a static format

© 2013 The City of New York

Contact Us | FAQs | Privacy Statement | Terms of Use | Site Map

* FOR EASY EXCEL CLEANING

* CLEANED FILE AT sample 3a/data/311_2009-Noise.csv

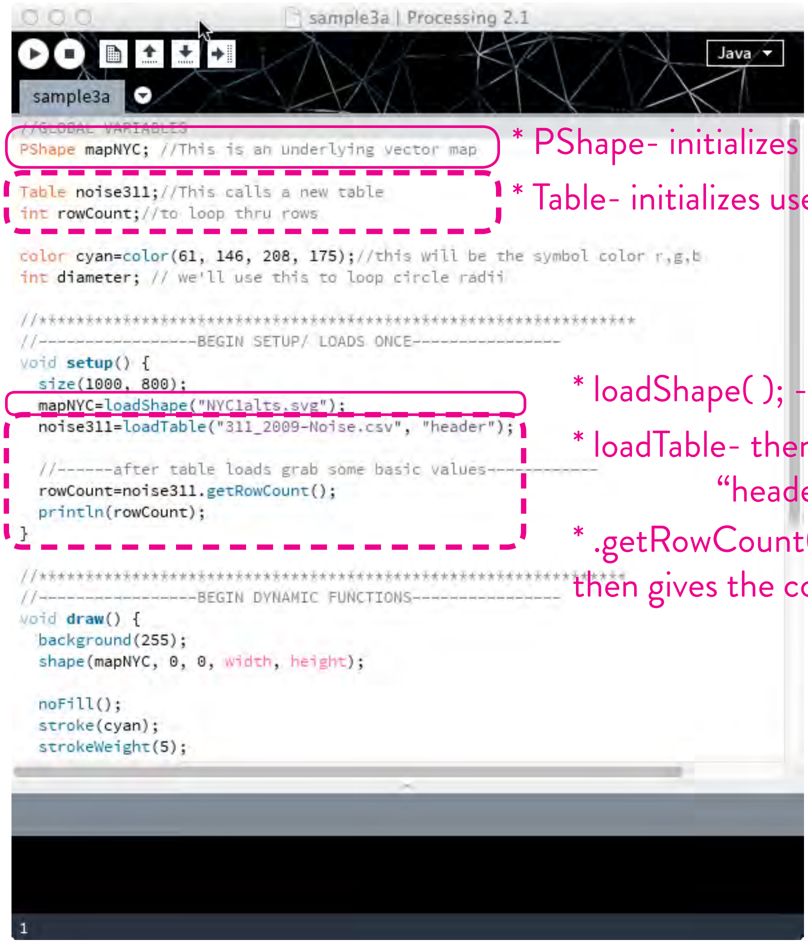
SKETCH: EXAMPLE 3A **ORIGINAL & CLEANED SOURCES**

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("NYClalts.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);
```

```
1
```

* PShape- initializes use of vector files

* Table- initializes use of csv files

* loadShape( ); - then loads svg files

* loadTable- then loads csv files,
"header" notes column titles

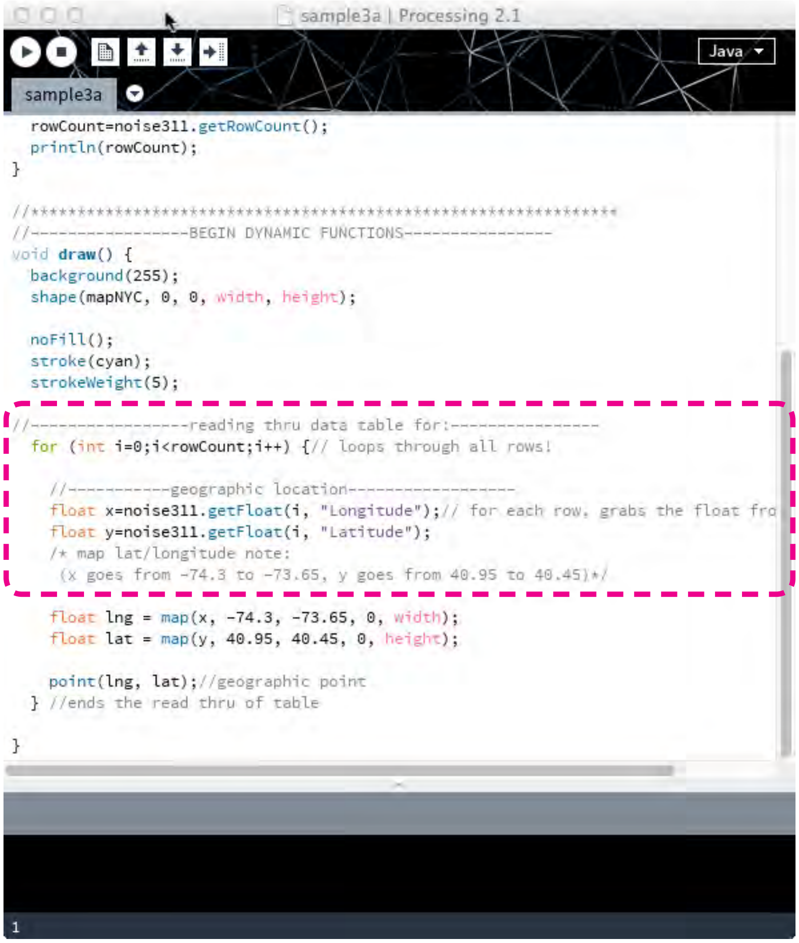
* .getRowCount( );
then gives the count of those rows for later use

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

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

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

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



```
  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

}
```

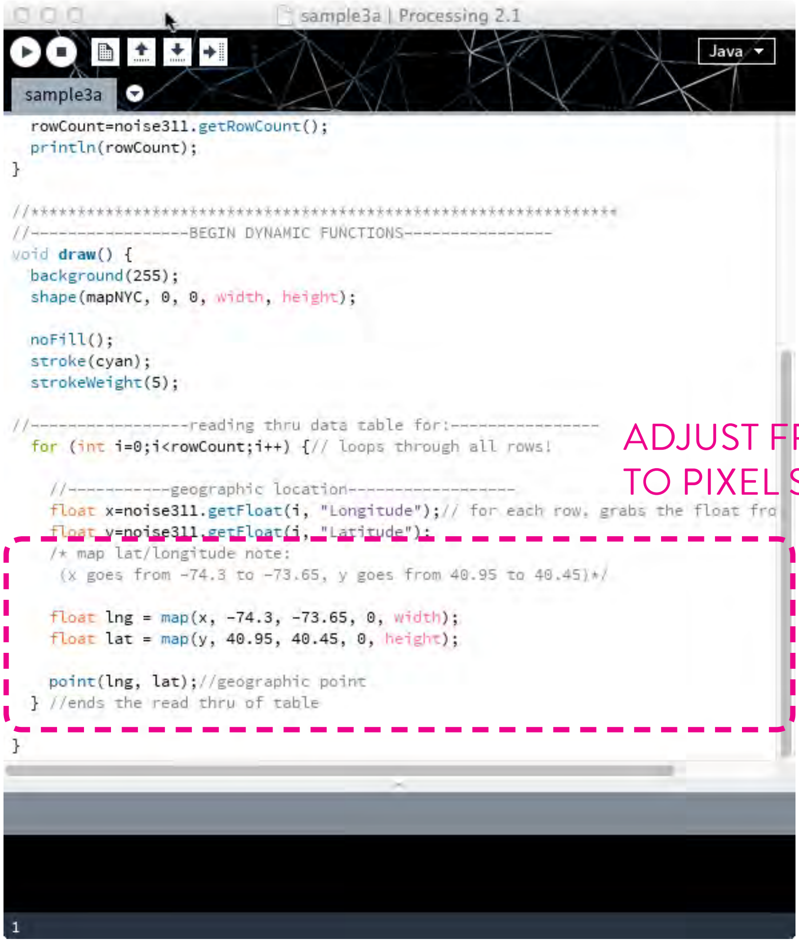
*loop through table rows to get values

* .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).



```
  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
}
```

ADJUST FROM GEOGRAPHIC
TO PIXEL SPACE OF THE SKETCH

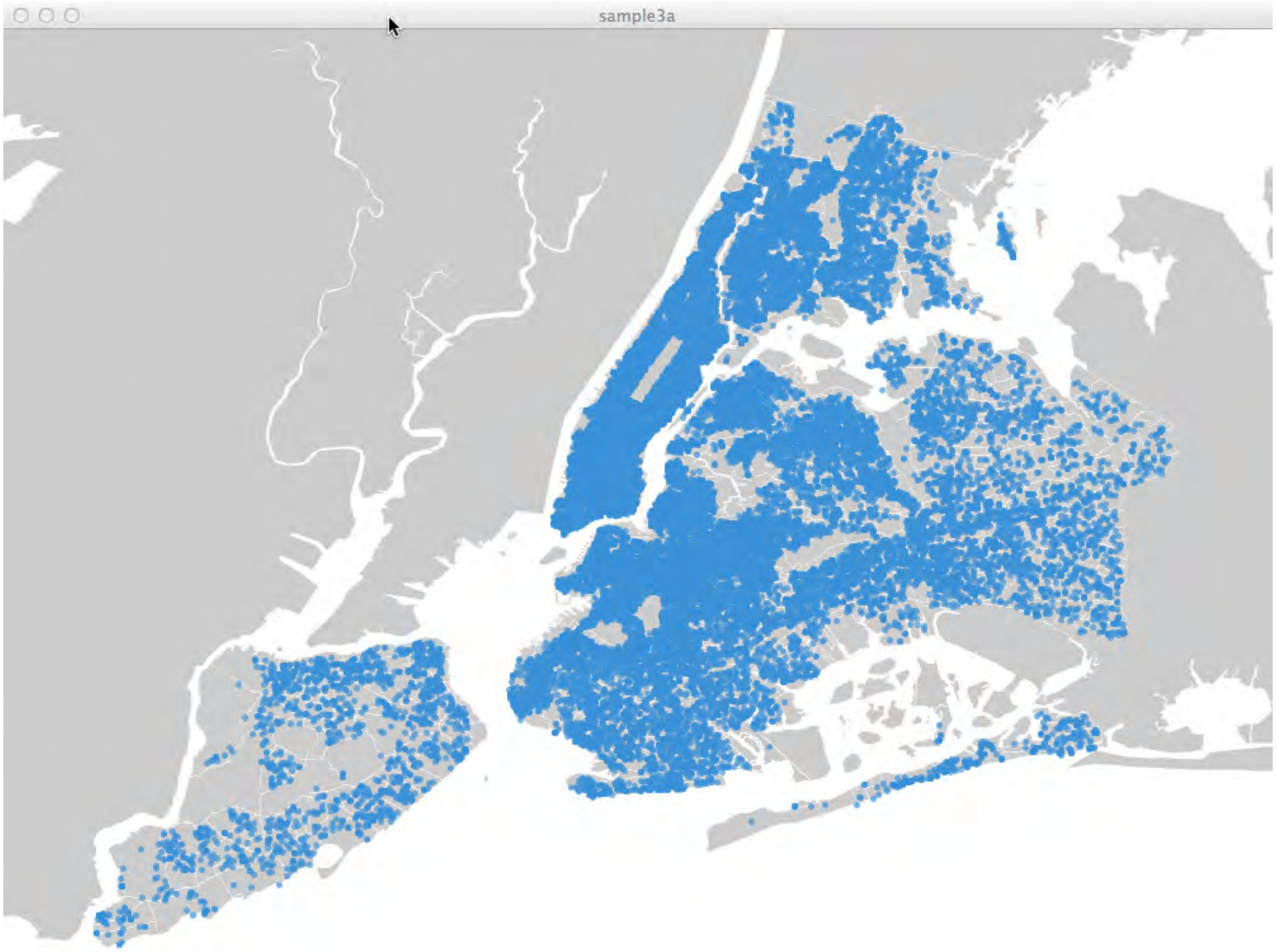
* map (variable, x, y, new x, new y)

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

SKETCH: EXAMPLE 3A **TABLE READING**

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

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



SKETCH: EXAMPLE 3A **POINTS MAPPED**

The basic result is shown above.

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

TABLES: **MORE ON COMMON READING MECHANISMS**

TABLE FUNCTIONS:

more basics for
use with loadTable(), saveTable()

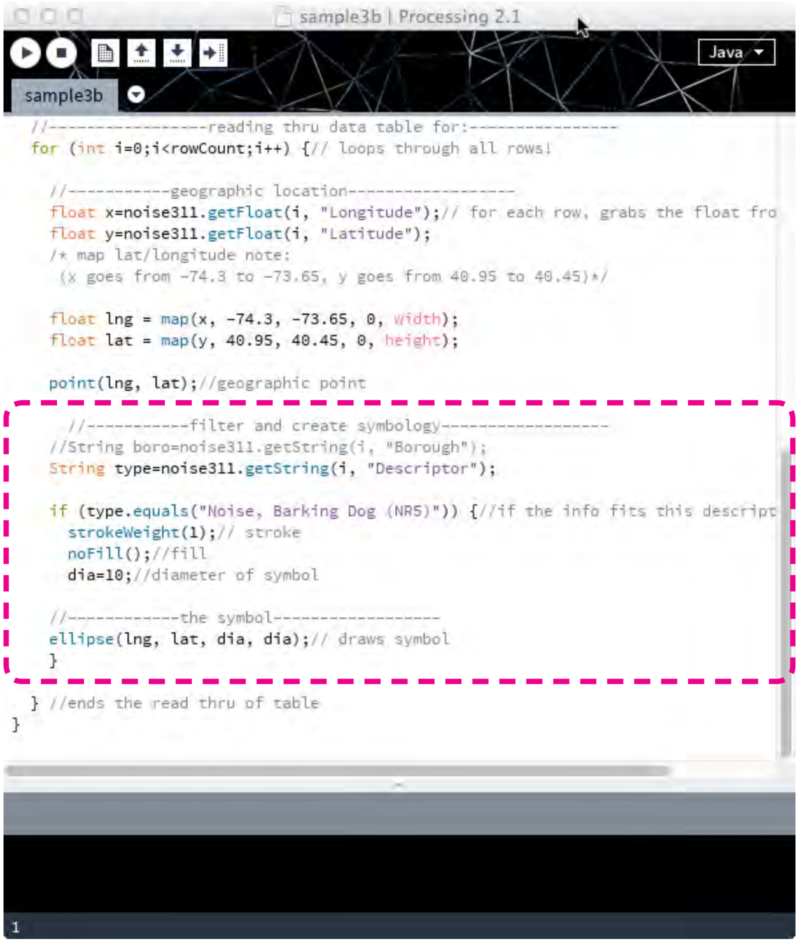
addColumn()	*Adds a new column to a table*
removeColumn()	*Removes a column from a table*
getColumnCount()	*Gets the number of columns in a table*
getRowCount()	*Gets the number of rows in a table*
clearRows()	*Removes all rows from a table*
addRow()	*Adds a row to a table*
removeRow()	*Removes a row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

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

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

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

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



```
  //-----------------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 descript
      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
}
```

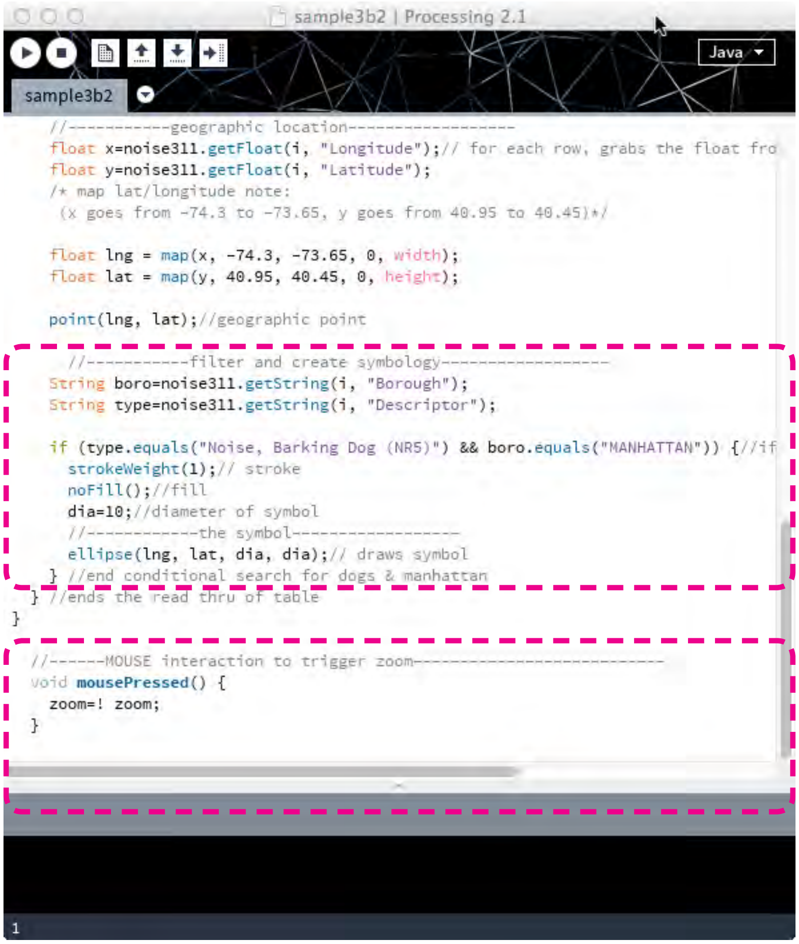
* .getString(row, column by header)
works like .getFloat( )

*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.



```
    //-----------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=! zoom;
  }
```

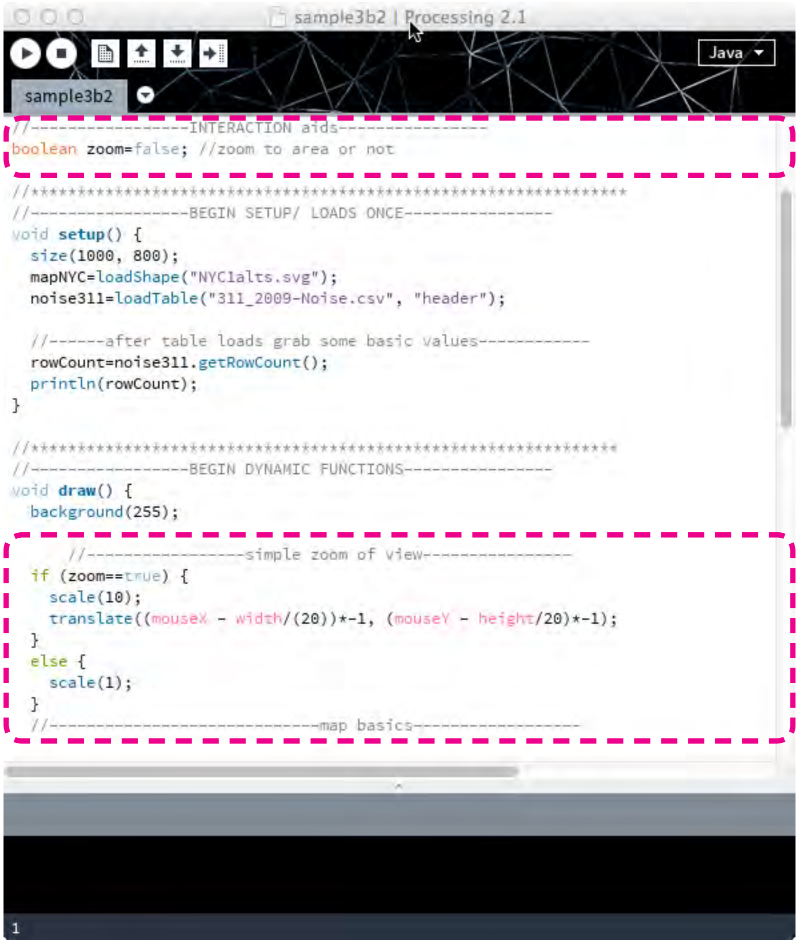
Create another variable boro and add to type.equals with a logical && to filter for “MANHATTAN” as well as Dog Barking

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("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);

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

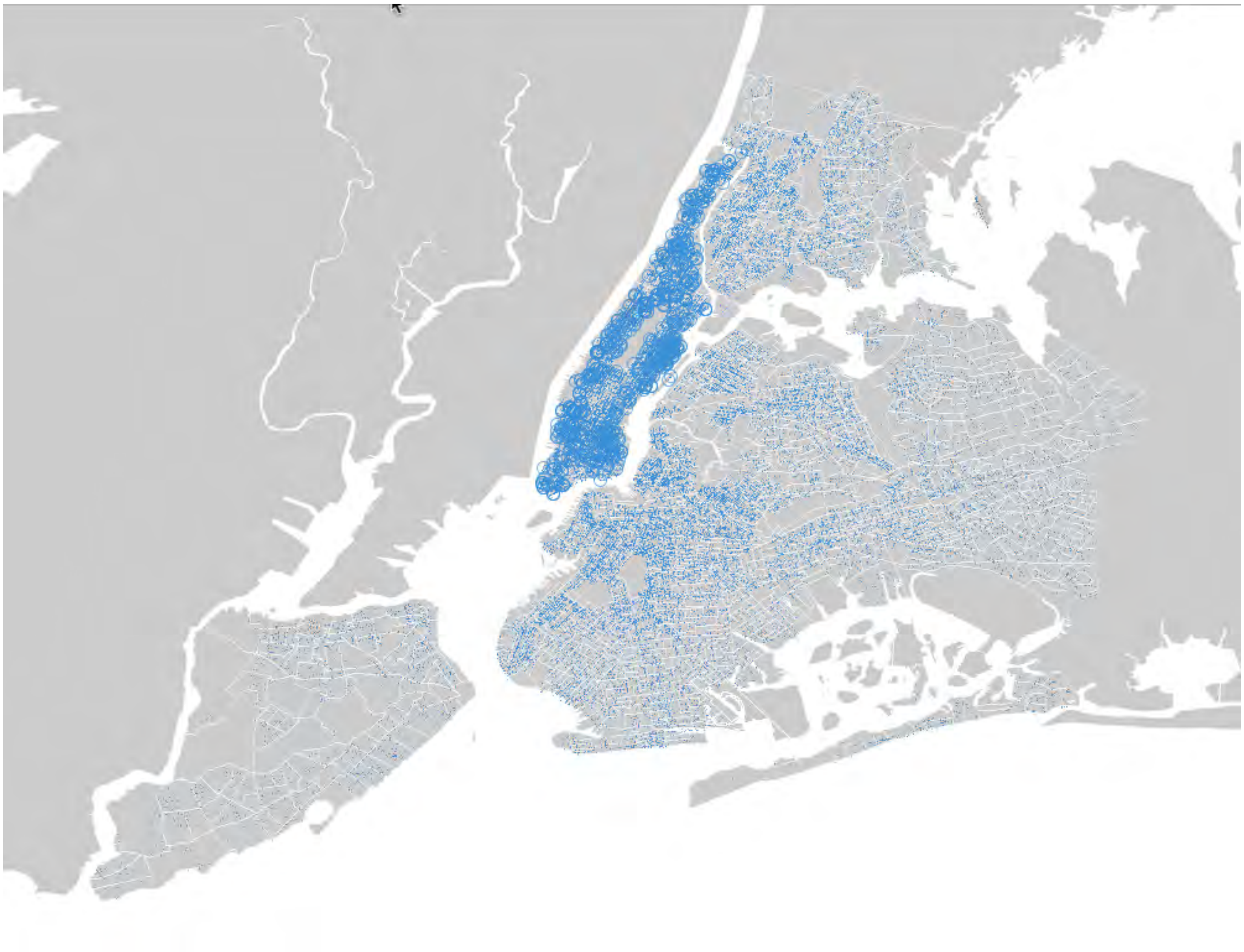
Add “Boolean zoom=false;” up at the top

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.

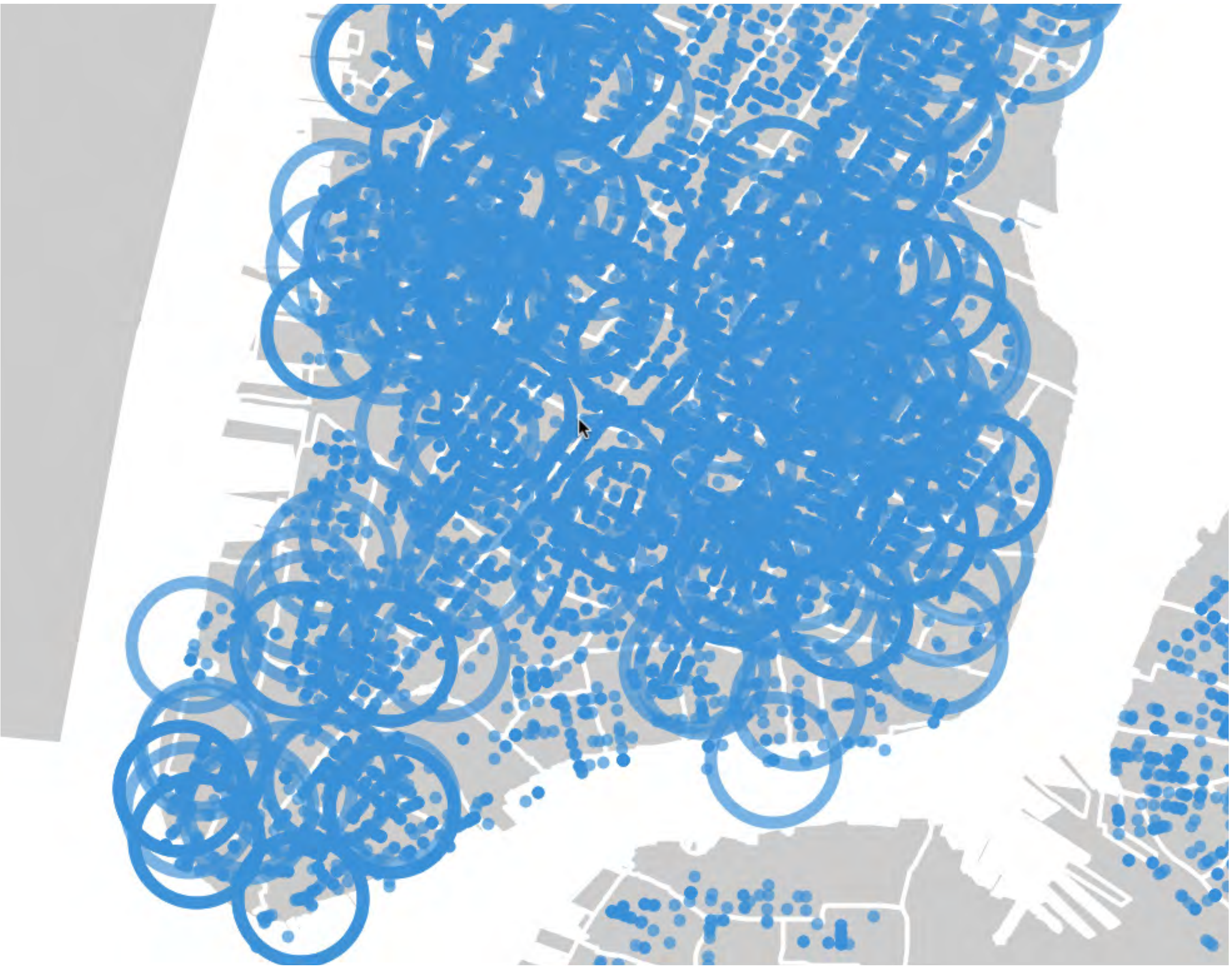
SKETCH: EXAMPLE 3B2 **INTERACT**

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



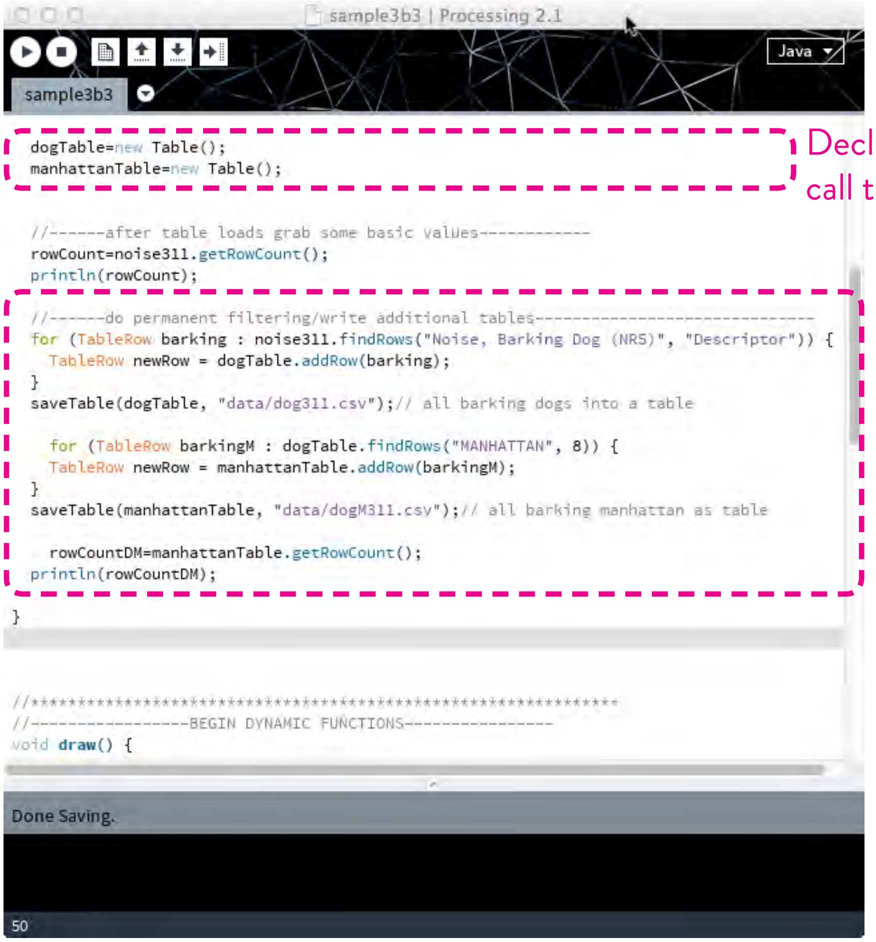
SKETCH: EXAMPLE 3B2 **UNZOOMED**

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 (NR5)", "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);

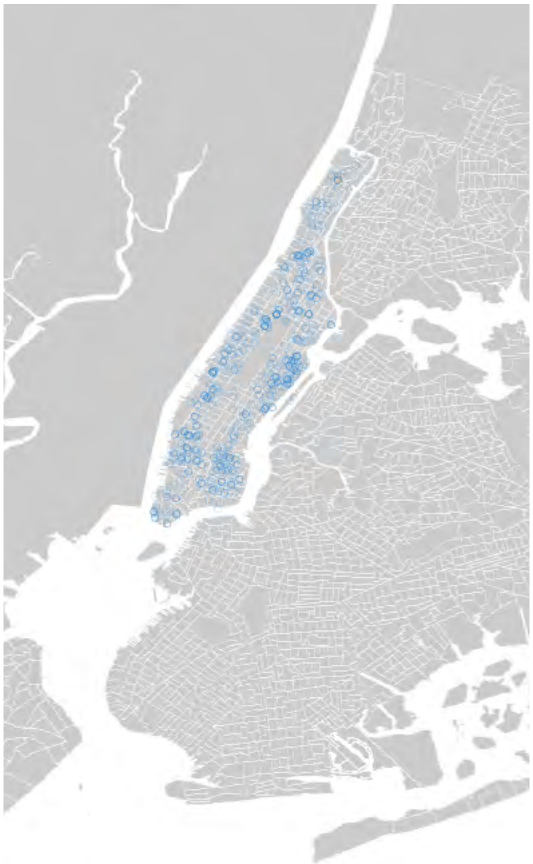
*saveTable(table, “filename”);

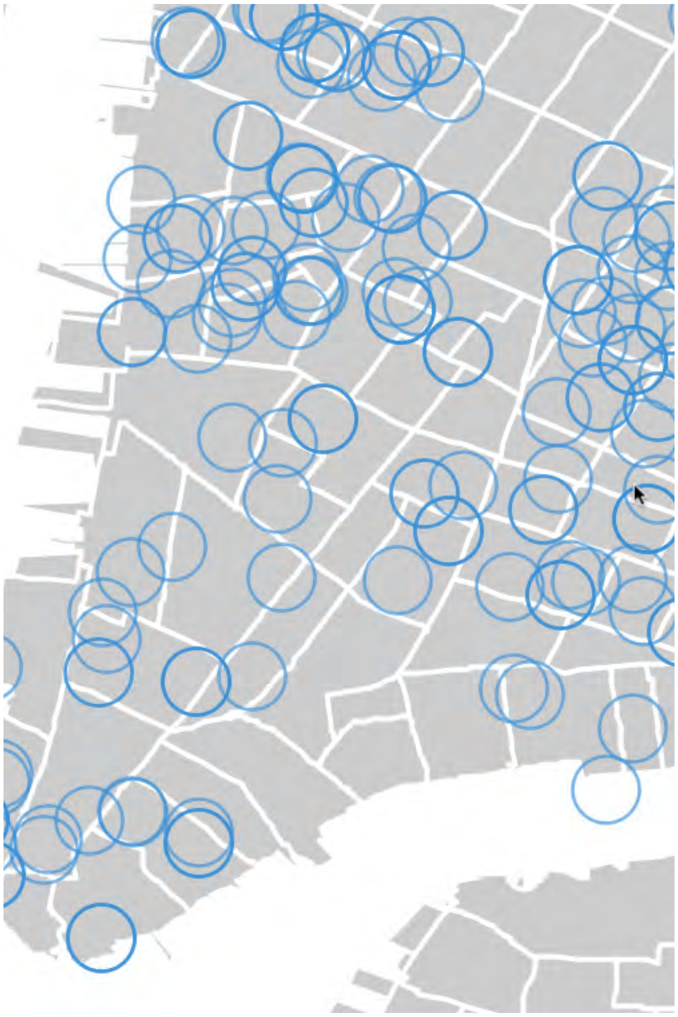
SKETCH: EXAMPLE 3B3 **CREATING SORTED TABLES**

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. . .

WK 2-MATH, INTERACTIONS, & MULTIPLE DISPLAY FORMS

  - shifting from 311 dog calls to other NYC Open Data

- simple math from tables
  - StringLists
  - InterDicts
  - Arrays

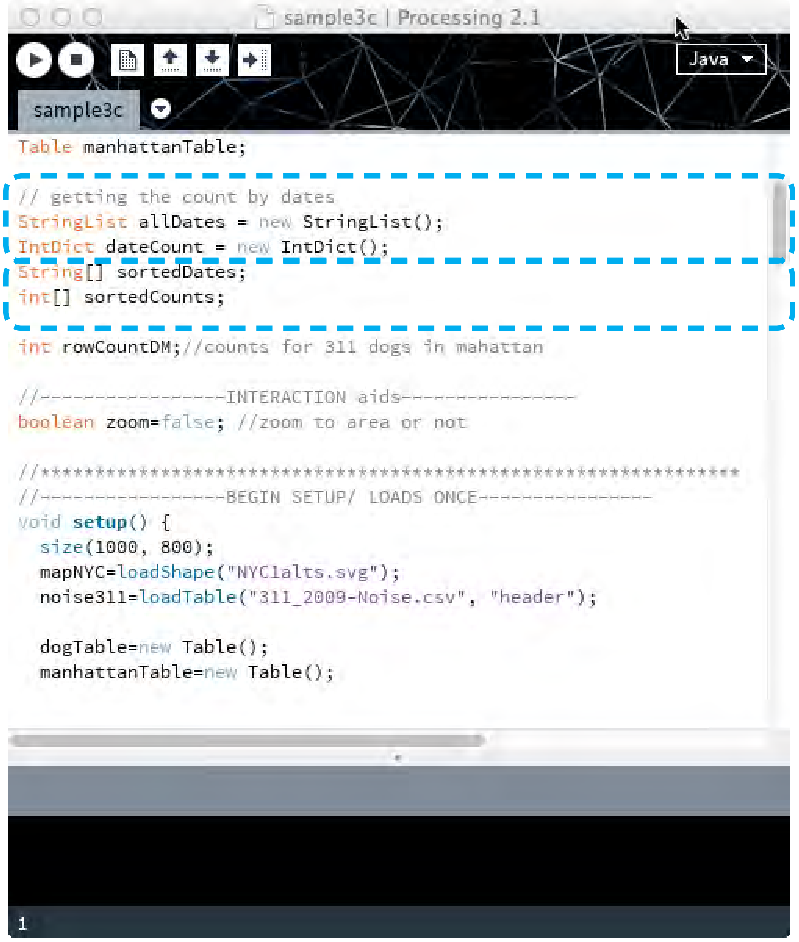
- legible & interactive information
  - labels and font use
  - mouse and keyboard functions
  - printing and export options for jpg, png, and pdf

- 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



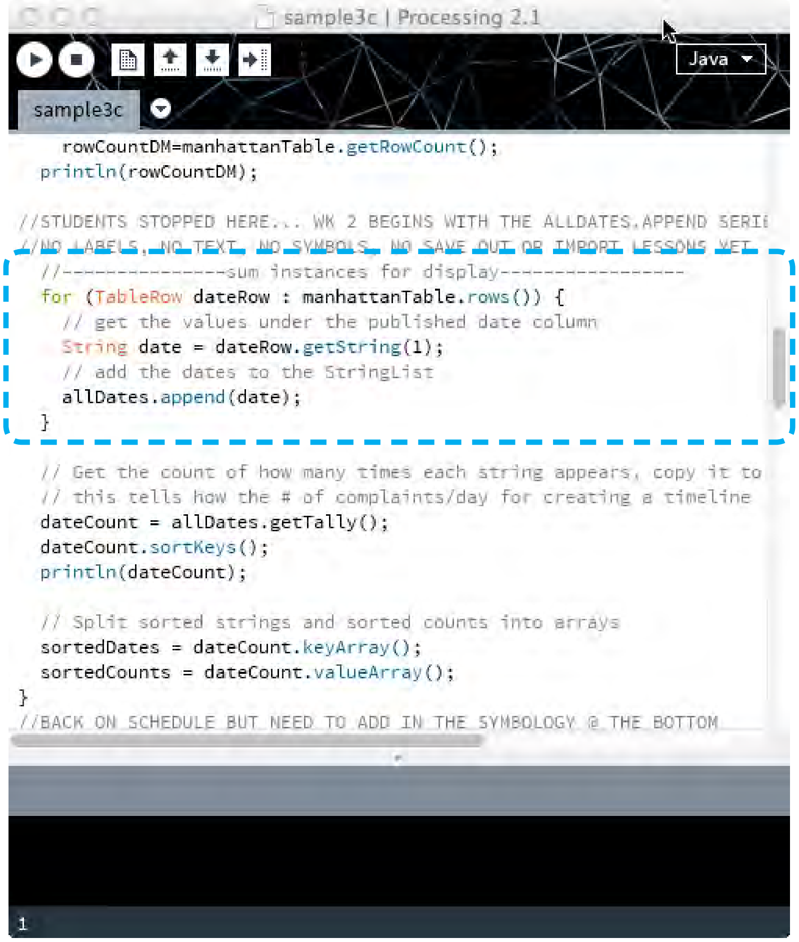
```
Table manhattanTable;

// getting the count by dates
StringList allDates = new StringList();
IntDict dateCount = new IntDict();
String[] sortedDates;
int[] sortedCounts;

int rowCountDM;//counts for 311 dogs in mahattan

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

//****************************************************************
//-----------------BEGIN SETUP/ LOADS ONCE----------------
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYC1alts.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.keyArray();
  sortedCounts = dateCount.valueArray();
}
//BACK ON SCHEDULE BUT NEED TO ADD IN THE SYMBOLOGY @ THE BOTTOM
```

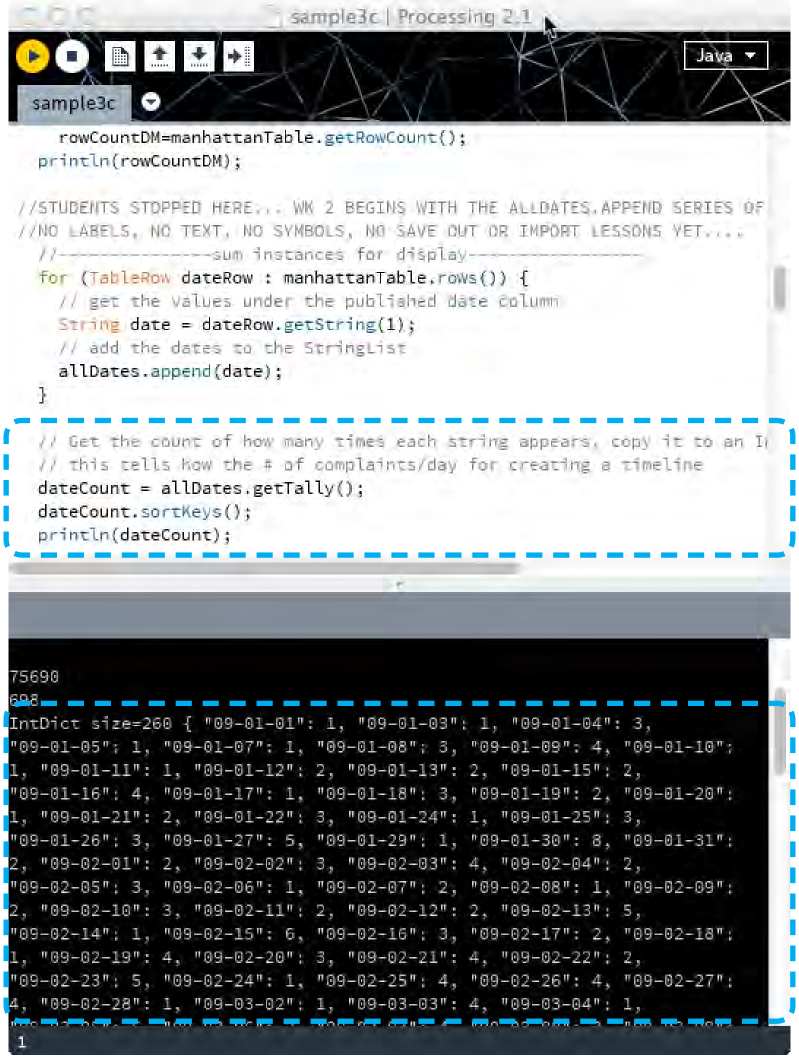
Using the “for (TableRow x: table.rows( ))” we can then read through all rows in the manhattan table in order to then pull the date value of each complaint by using .getString(1). (There 1 is the column of complaint dates.)

.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.



```
    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 I
  // 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,
```

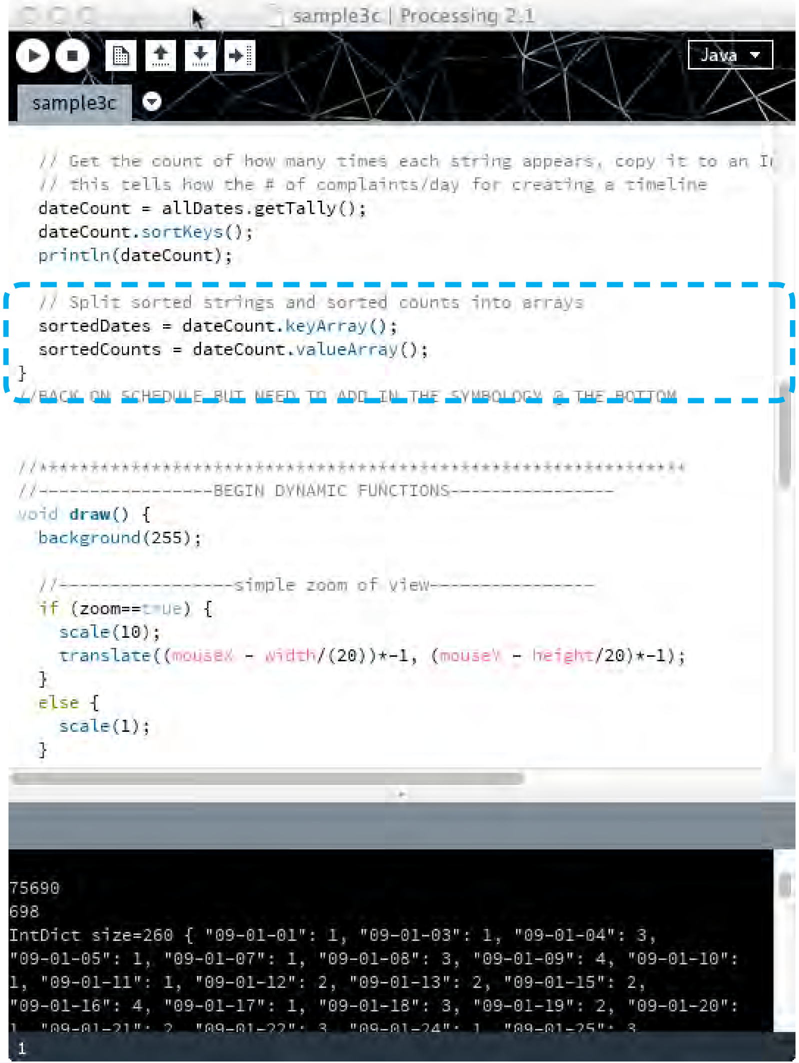
The function .getTally( ) compares the dates in the StringList and returns a condensed list of dates and the number of calls per date. To store these linked values date/#of calls we will use the IntDict dateCount.

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. SortKeys 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 I
  // 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.keyArray();
  sortedCounts = dateCount.valueArray();
}
//BACK ON SCHEDULE BUT NEED TO ADD IN THE SYMBOLOGY @ 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
```

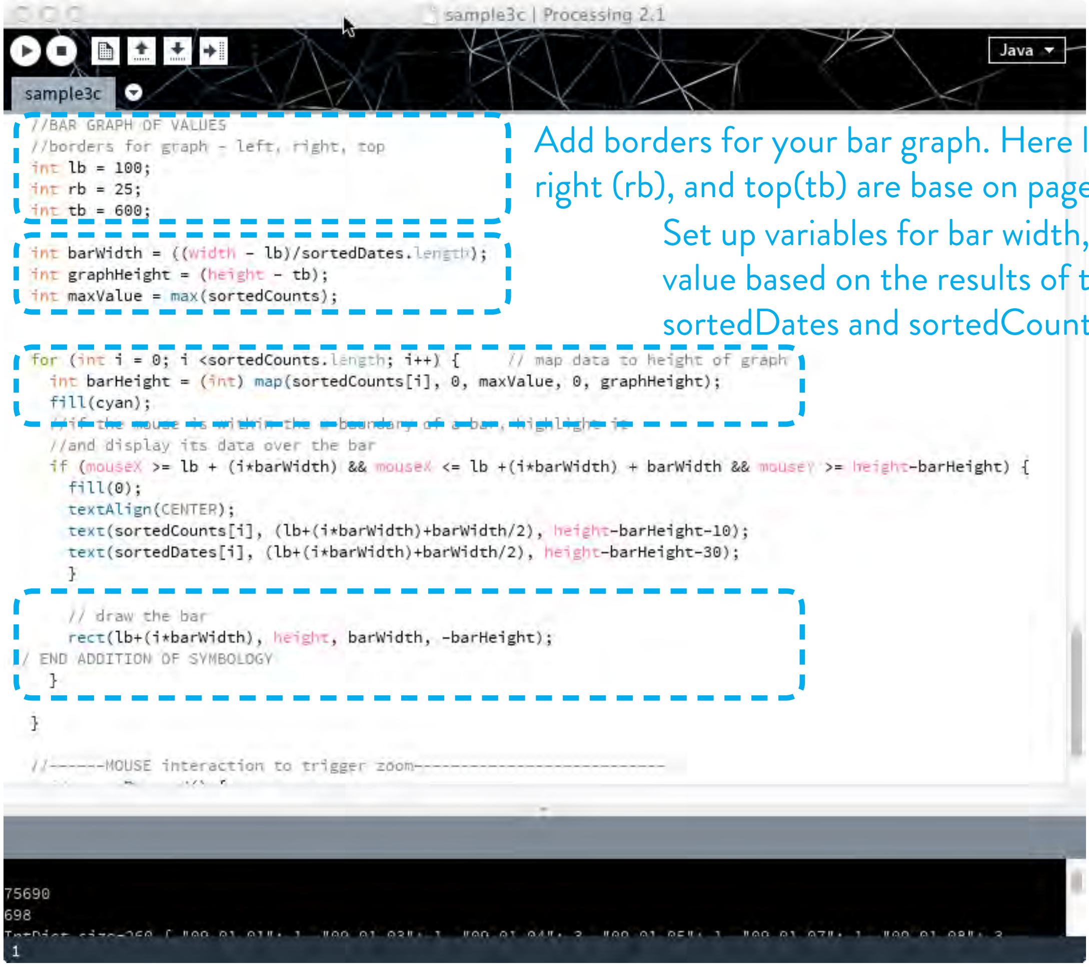
The function .keyArray( ) makes an array of those keys, i.e. dates. The function .valueArray ( ) then holds the values, i.e. count of incidents, in a second array.

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 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 the 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);
// END ADDITION OF SYMBOLOGY
  }

}

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

```
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
1
```

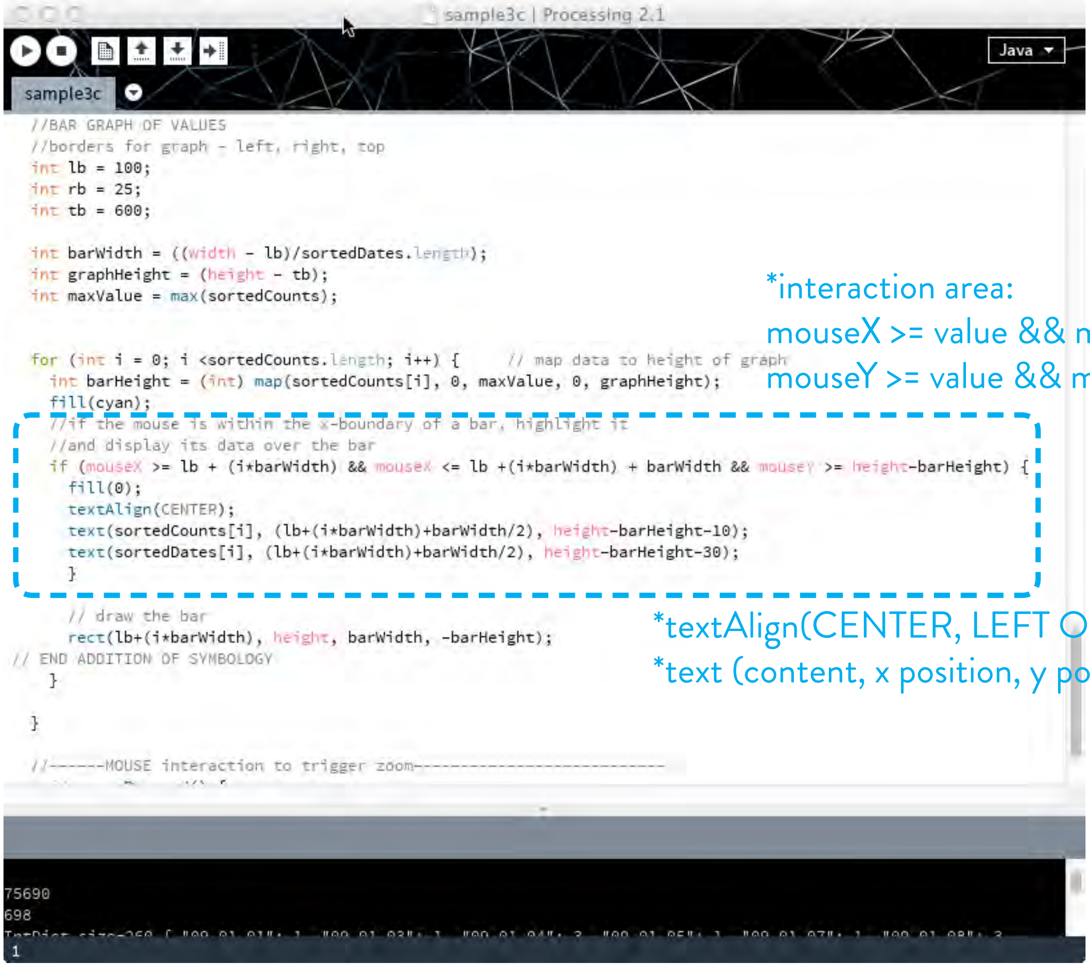
Add borders for your bar graph. Here left (lb), right (rb), and top(tb) are base on page size.

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 SYMBOLOGY
  }

}

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

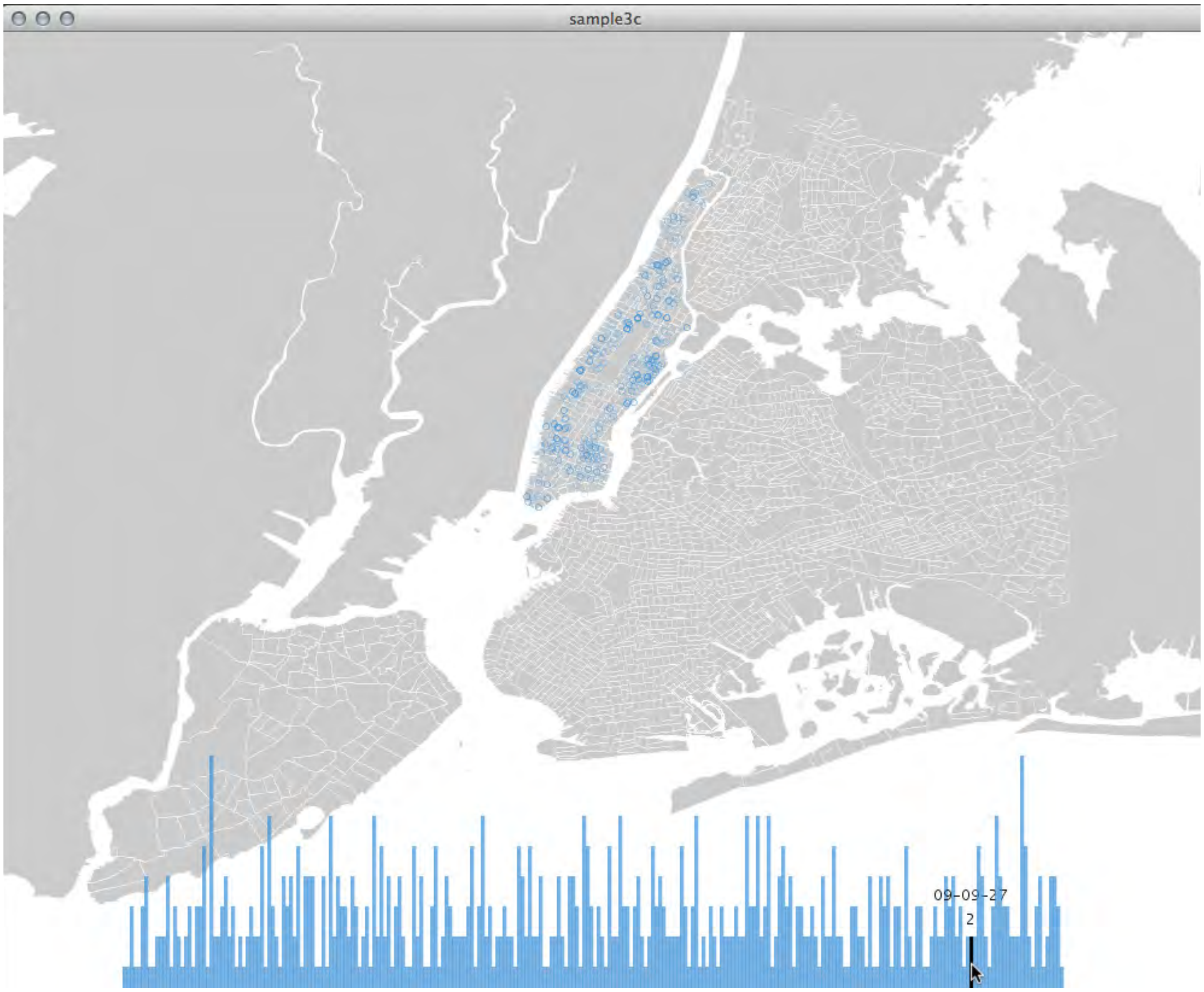
```
75690
698
IntDict size=260 { "00 01 01": 1, "00 01 03": 1, "00 01 04": 3, "00 01 05": 1, "00 01 07": 1, "00 01 08": 3
1
```

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

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

SKETCH: EXAMPLE 3C **ADDING INTERACTION**

For interaction, we want to logically limit the space the mouse is within (as equivalent to the bar space). Then, we can define alternate fill (black) and the text alignment and content for pop-up labels. See the specific structure of each of those conditional statements and text structures above.



SKETCH: EXAMPLE 3C **INTERACTIVE LABELS**

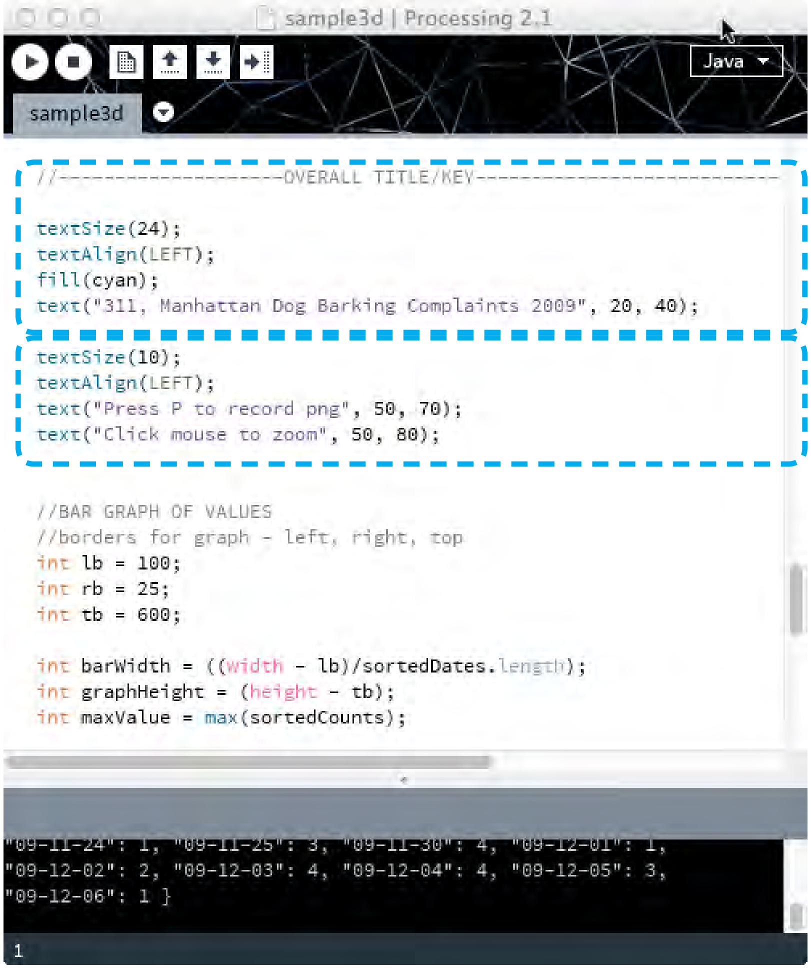
As seen at the bottom right, a hoovering mouse triggers a black fill and the display of the date and # of complaints.

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



```
//-------------------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 }
```

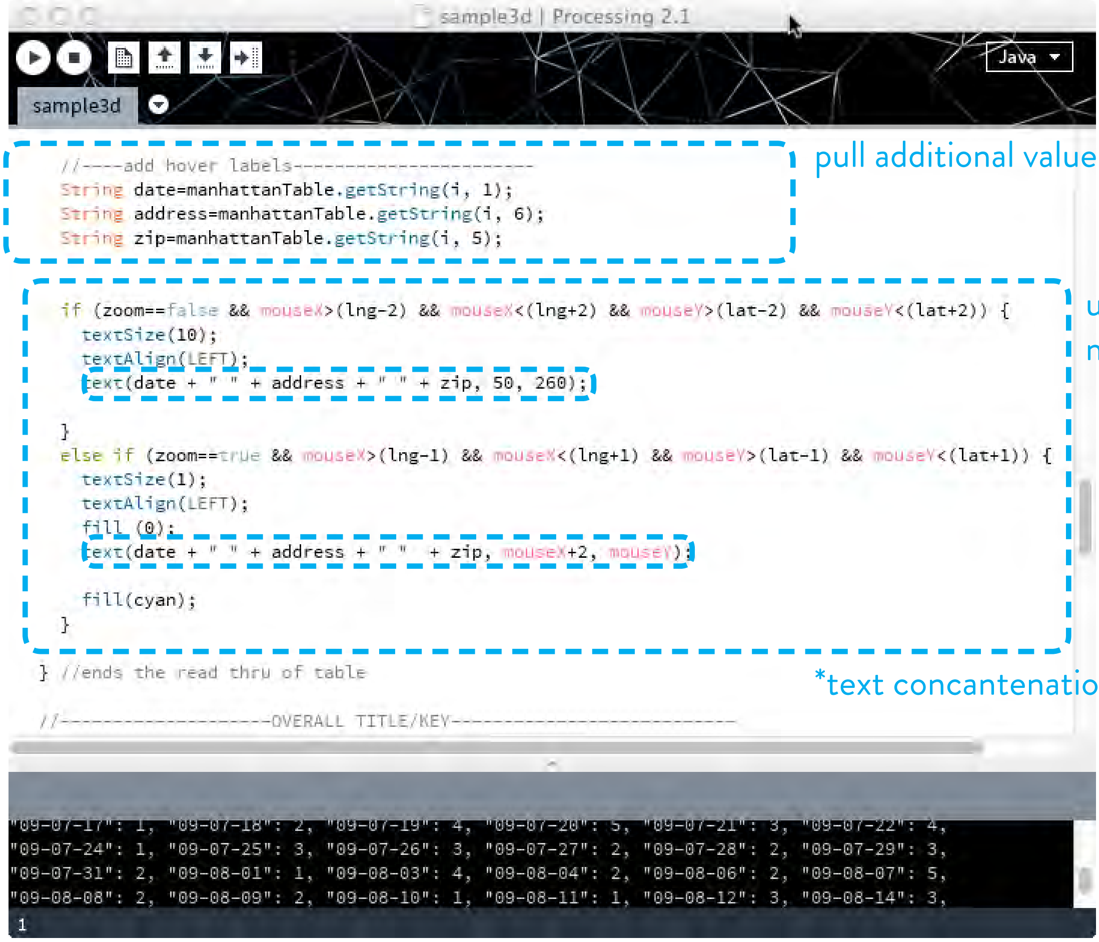
```
1
```

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----------------------------
```

```
"09-07-17": 1, "09-07-18": 2, "09-07-19": 4, "09-07-20": 5, "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, "09-08-12": 3, "09-08-14": 3,
```

pull additional value from the original table

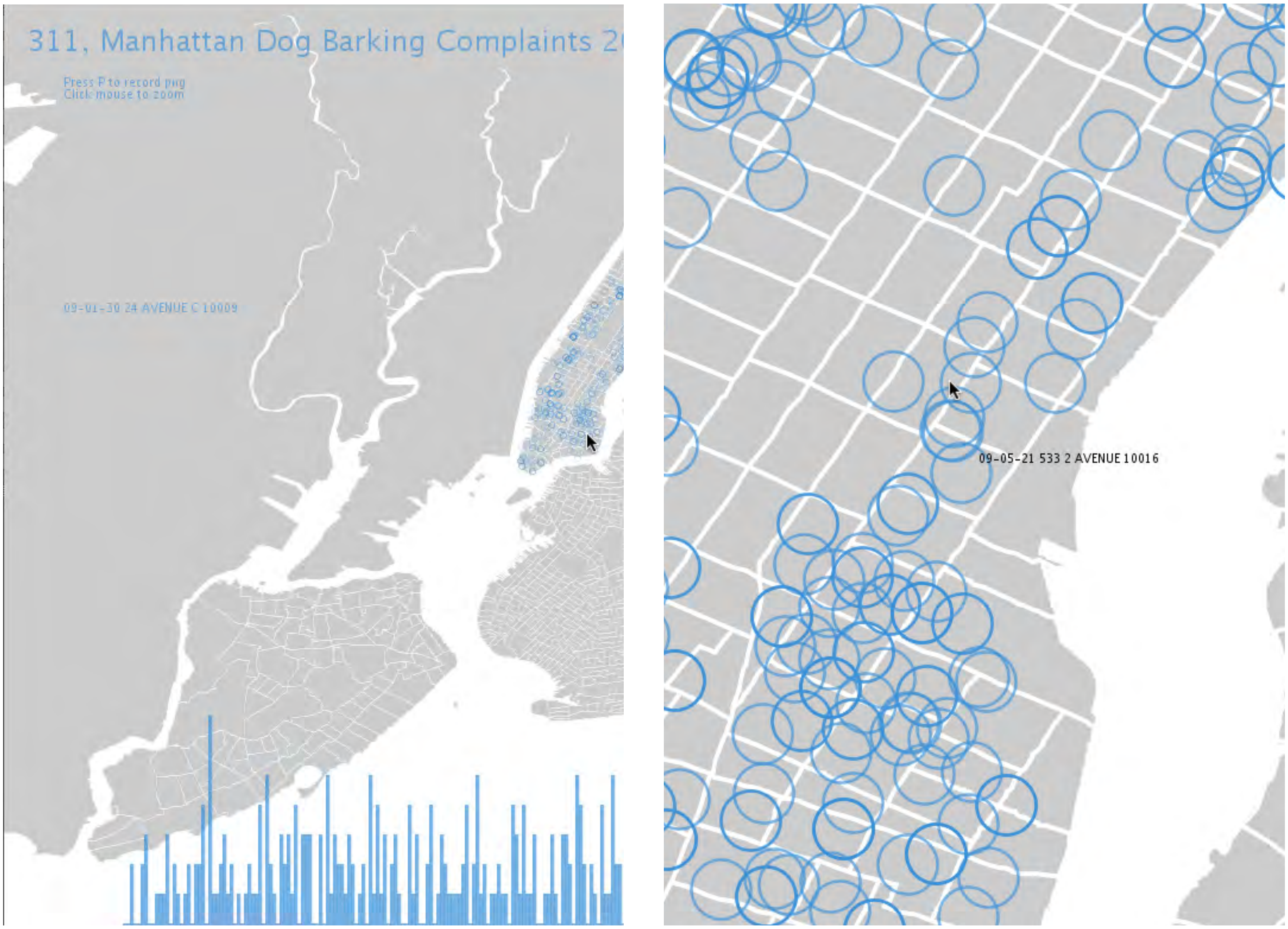
using conditionals for mouse position & zoom

*text concantenation with +

SKETCH: EXAMPLE 3D **MORE INTERACTIONS**

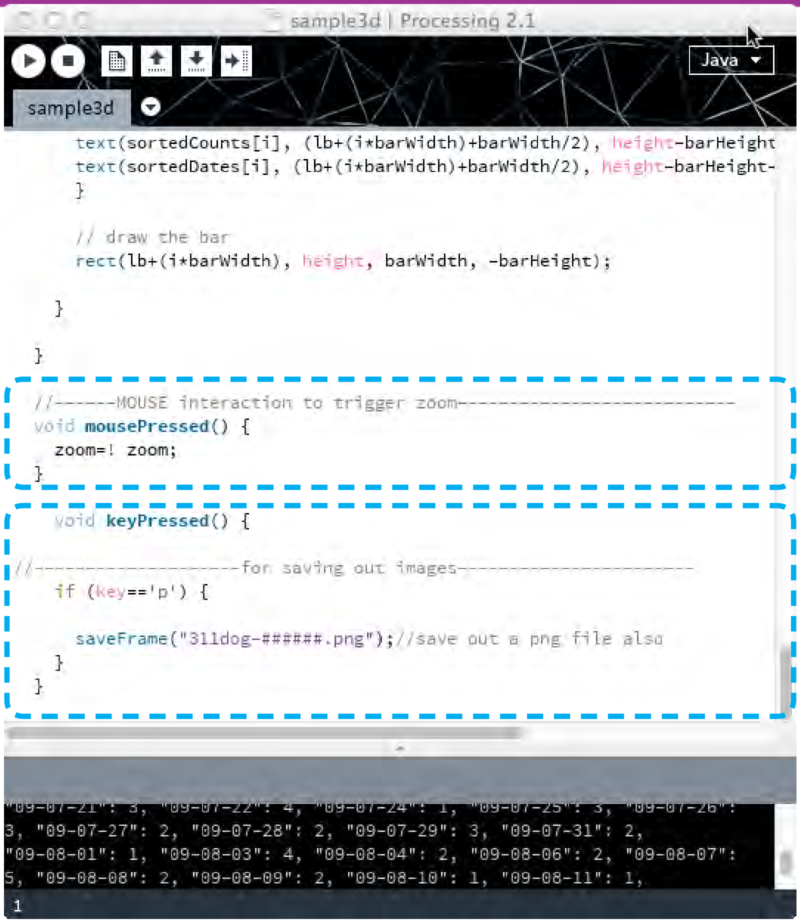
For hoovering labels on the main map, we'll do something similar to the bar-graph area.

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 concantenated, 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.


311, Manhattan Dog Barking Complaints 2
Press P to record png
Click mouse to zoom
09-01-30 24 AVENUE C 10009
09-05-21 533 2 AVENUE 10016

SKETCH: EXAMPLE 3D **INTERACTION**

The results should look like the above screen shots.



```
      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=! zoom;
  }
    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, jgp, 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”.

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

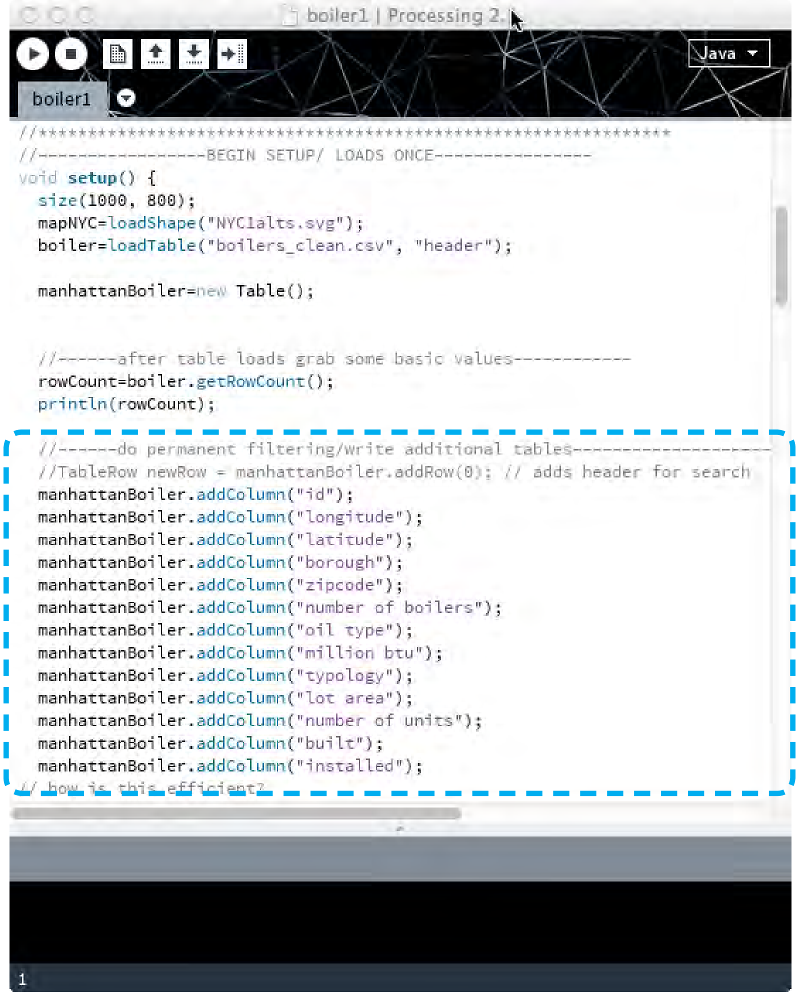
WK 2

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



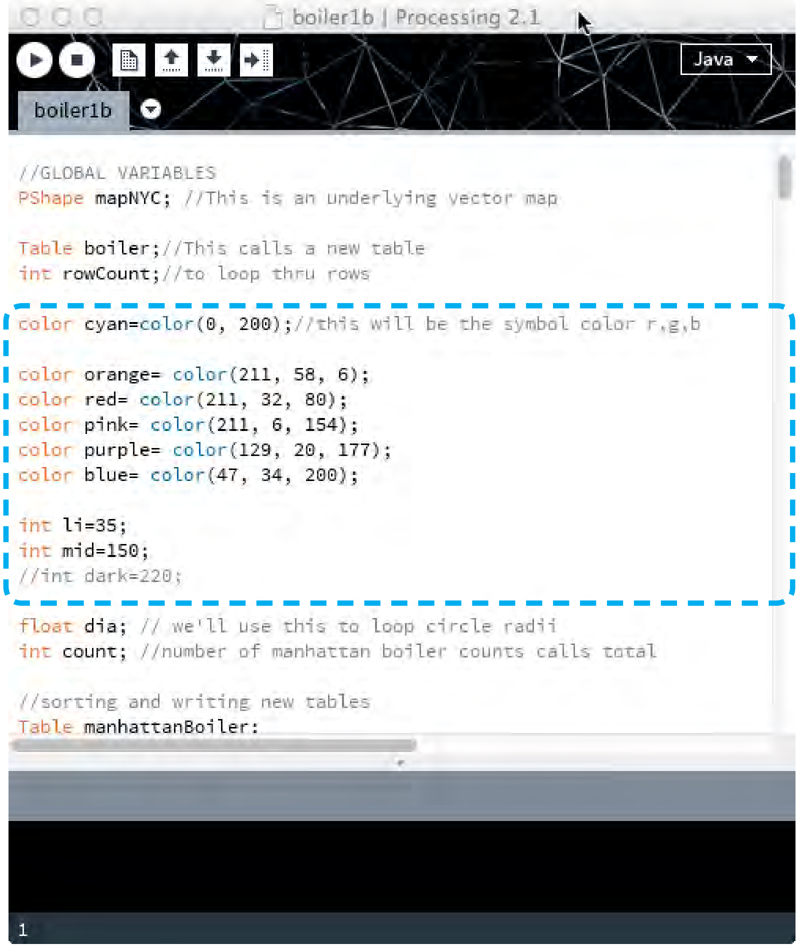
```
//*****************************************************************
//-----------------BEGIN SETUP/ LOADS ONCE----------------
void setup() {
  size(1000, 800);
  mapNYC=loadShape("NYC1alts.svg");
  boiler=loadTable("boilers_clean.csv", "header");

  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 manhattan-Boiler 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.



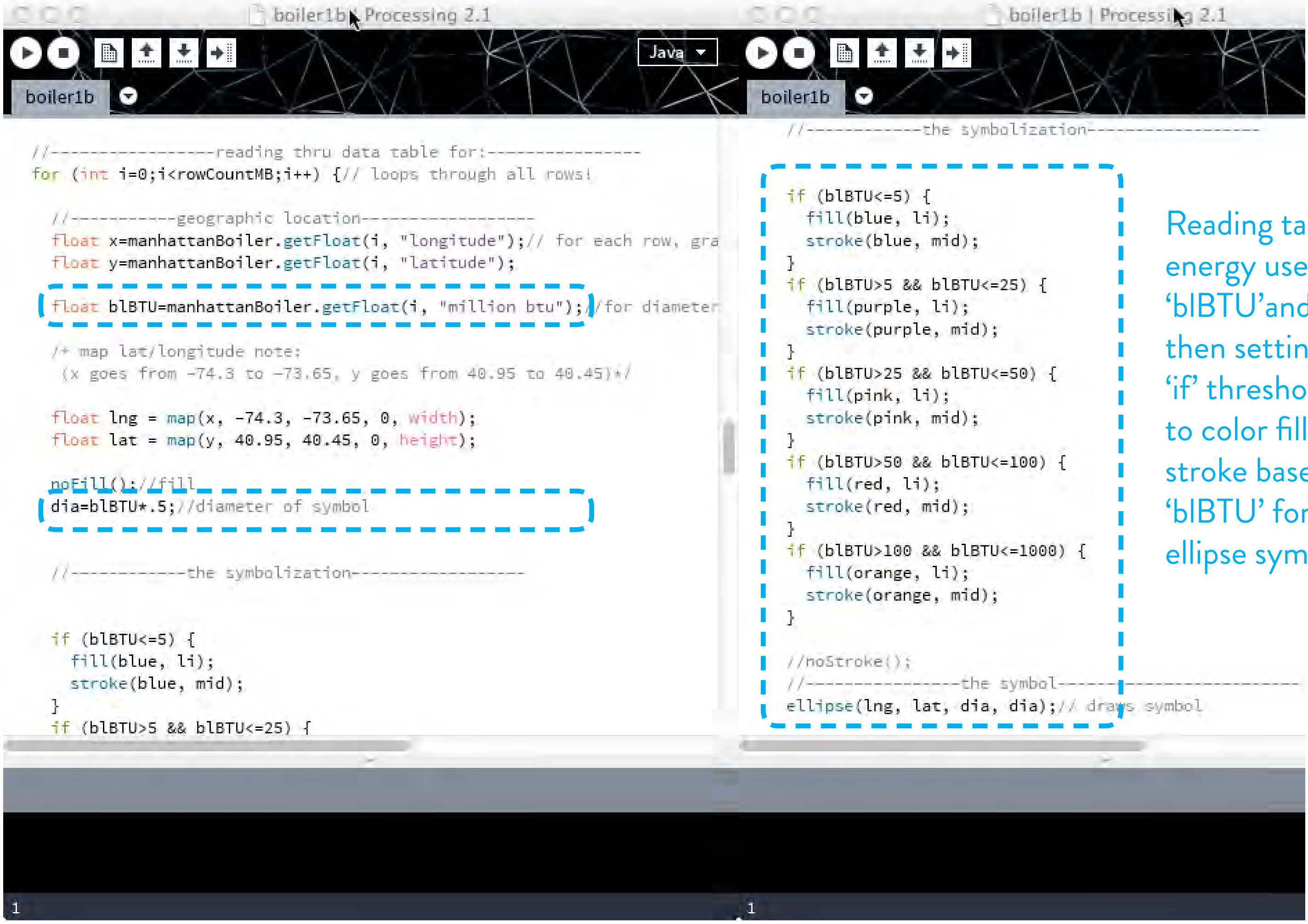
```
//GLOBAL VARIABLES
PShape mapNYC; //This is an underlying vector map

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. . .



```
//------------------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) {
```

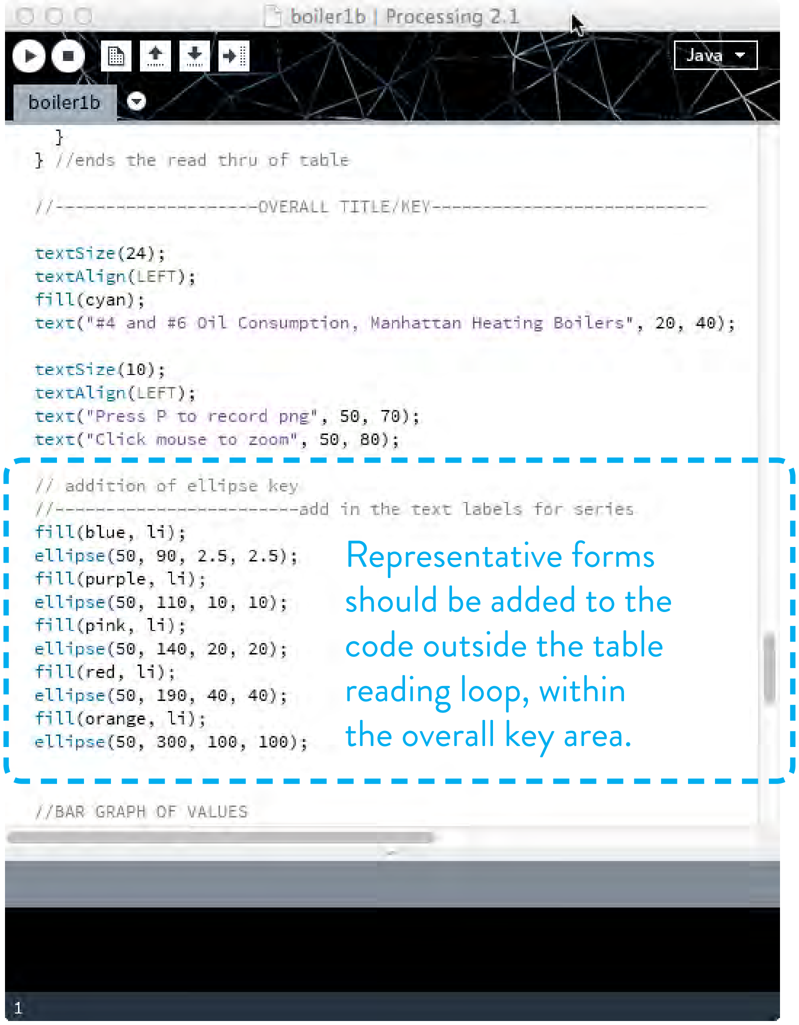
```
  //------------the symbolization------------------

  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 'blBTU'and then setting 'if' thresholds to color fill and stroke based on 'blBTU' 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.



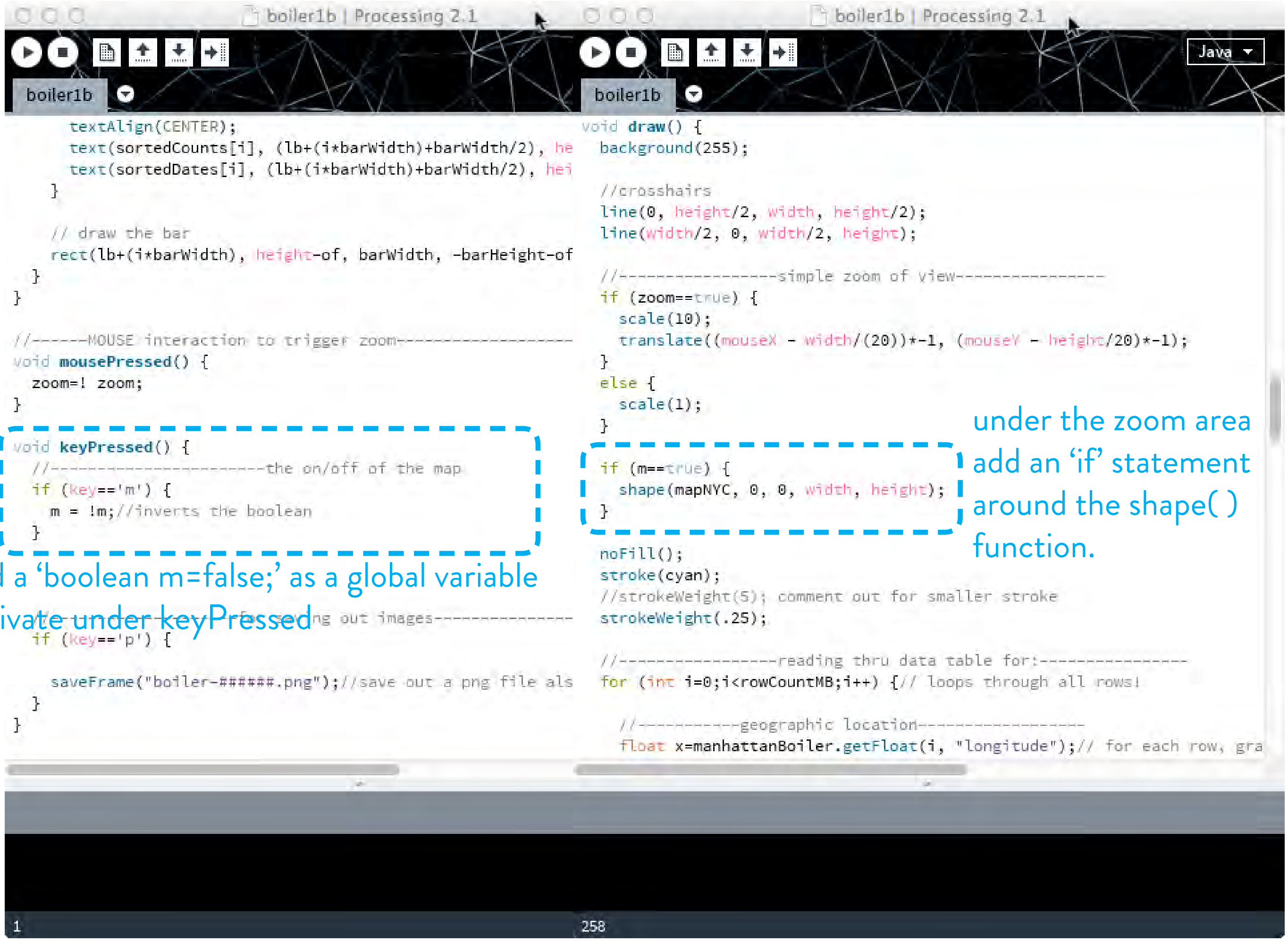
```
    }
  } //ends the read thru of table

  //--------------------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
```

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

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

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), he
      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
  }
  //----------For saving out images---------------
  if (key=='p') {

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

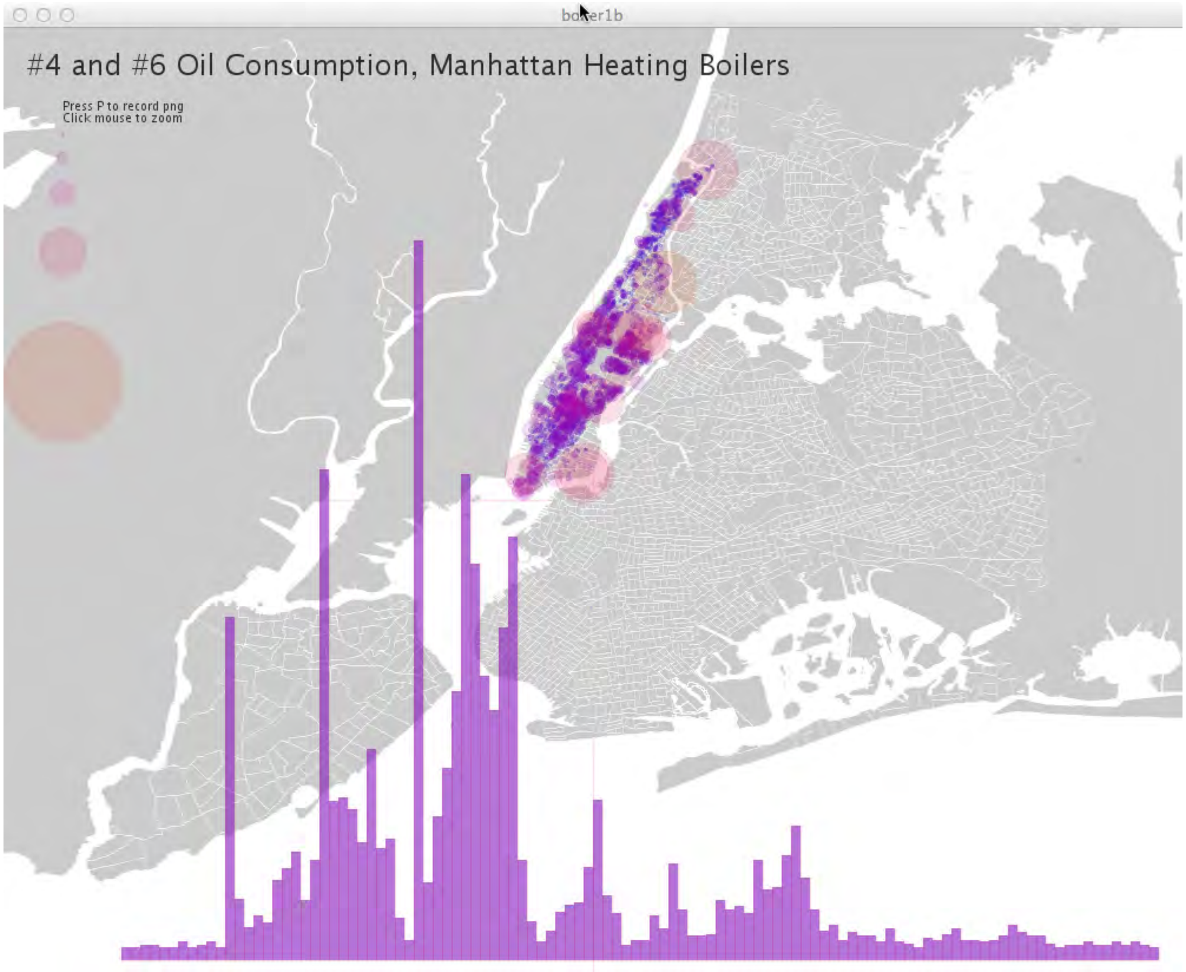
add a ‘boolean m=false;’ as a global variable
activate under keyPressed

```
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.

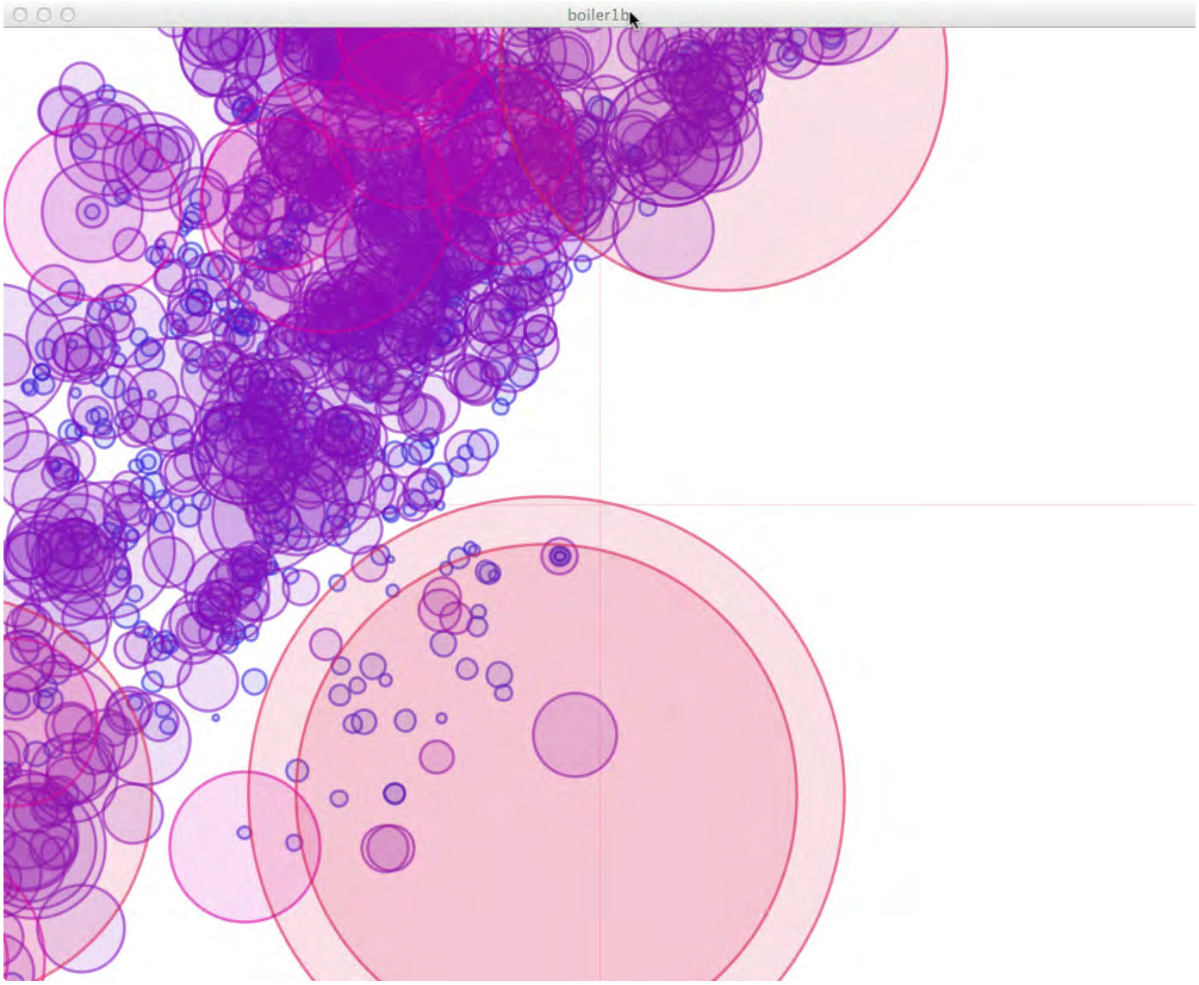


boiler1b

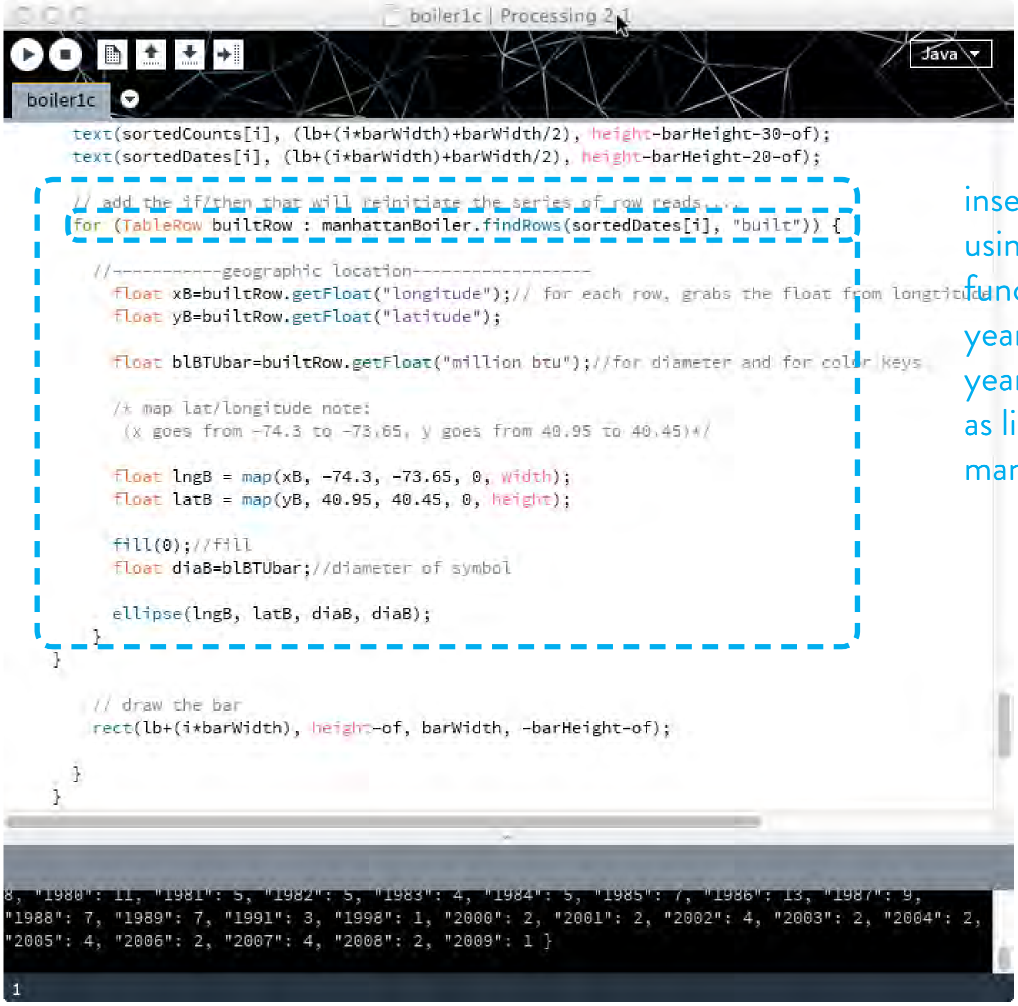
#4 and #6 Oil Consumption, Manhattan Heating Boilers

Press P to record png
Click mouse to zoom

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



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



```
    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 longtitude
        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 }
```

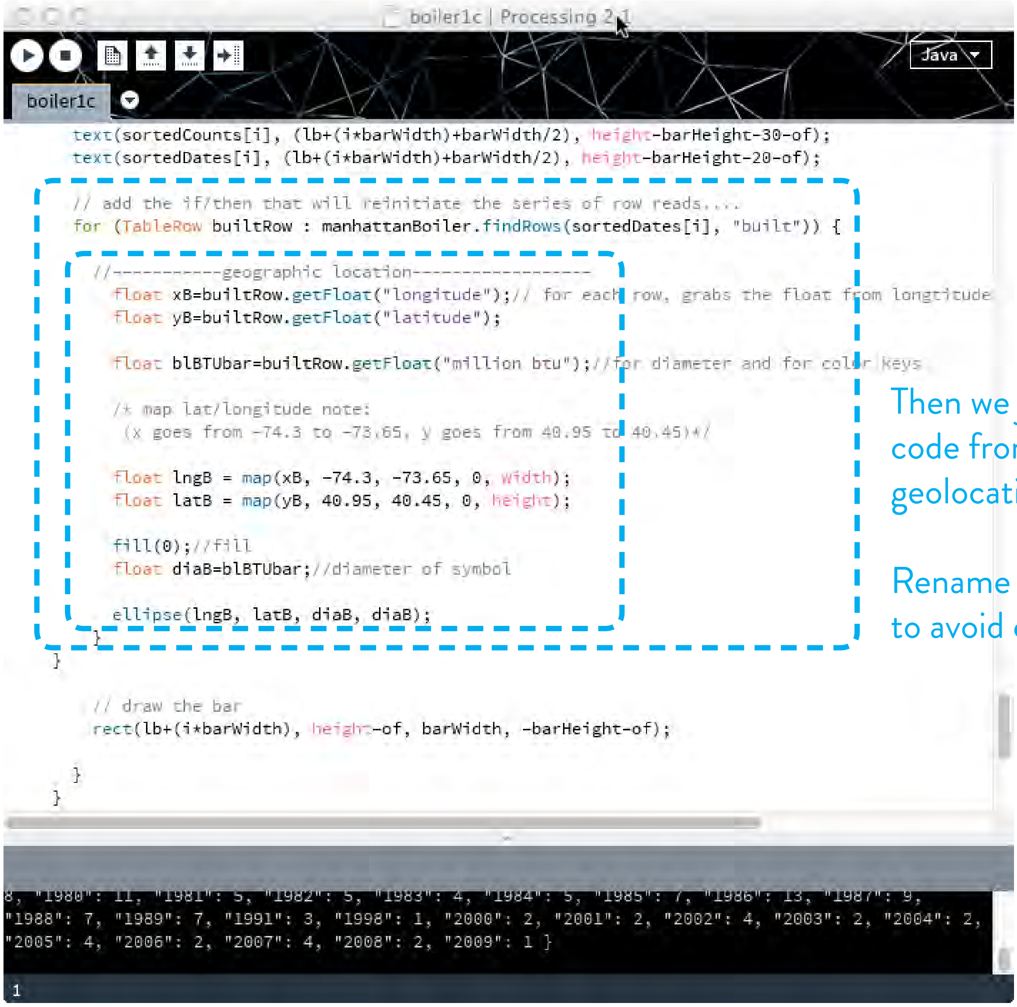
1

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 longtitude
      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 }
```

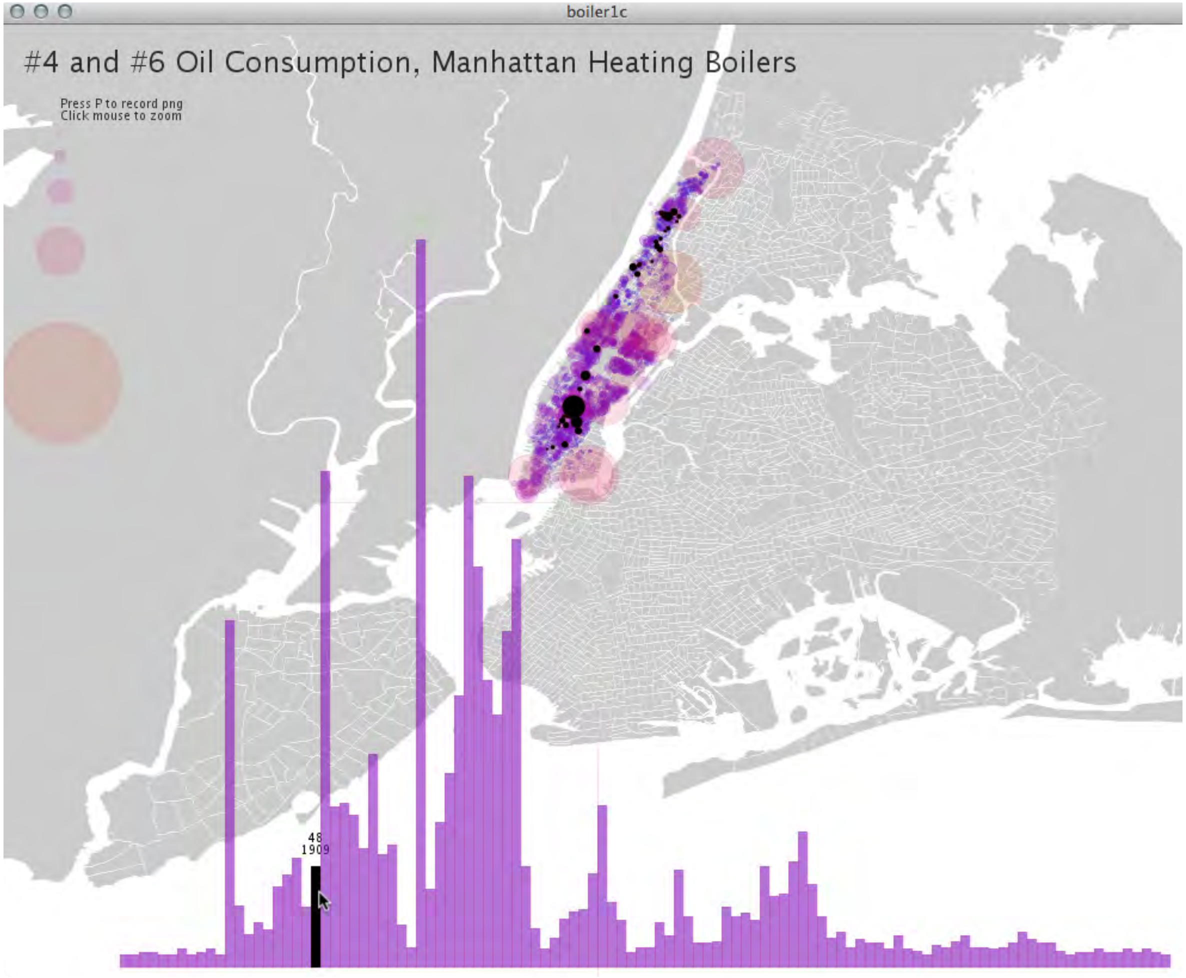
```
1
```

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.

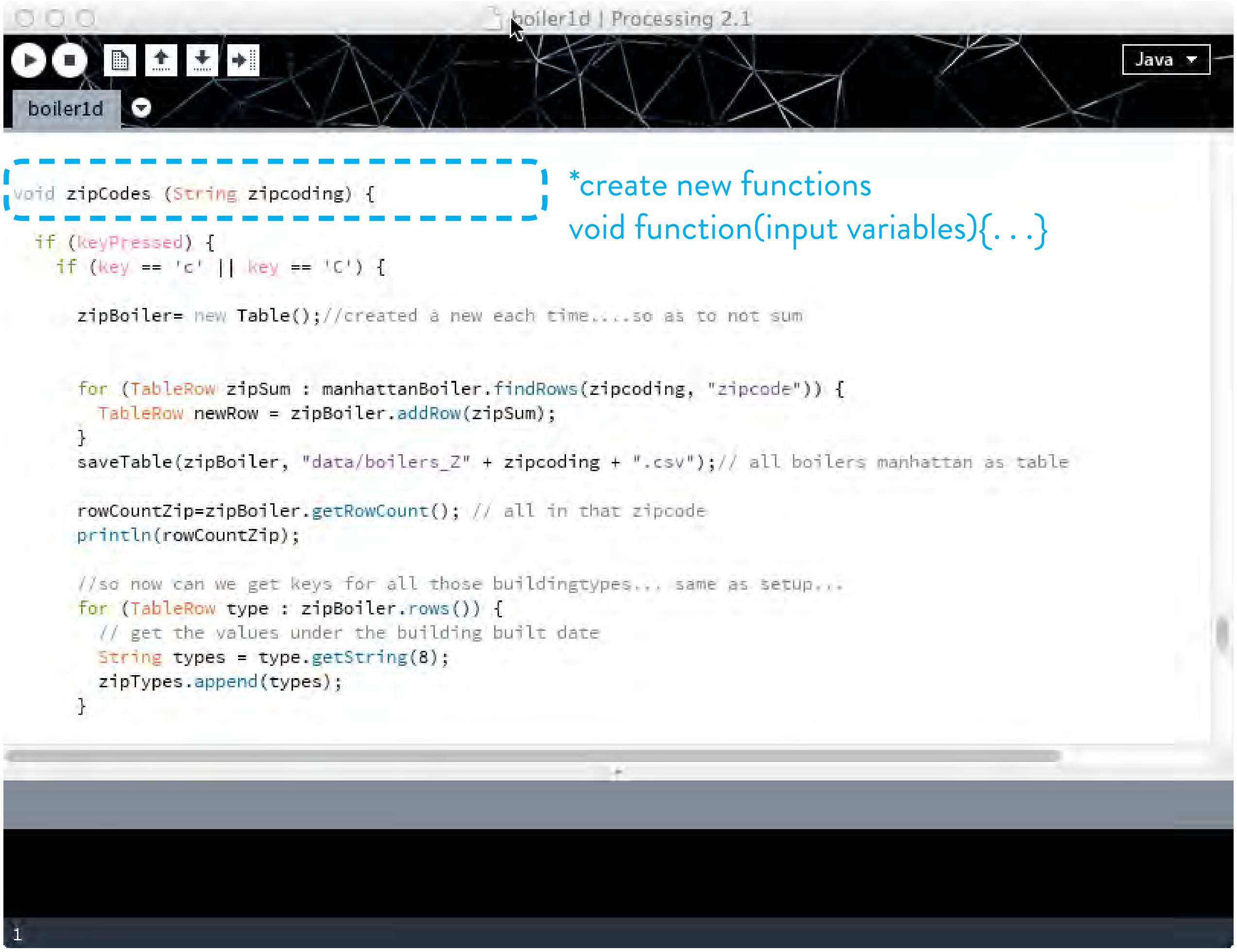


boiler1c

#4 and #6 Oil Consumption, Manhattan Heating Boilers

Press P to record png
Click mouse to zoom

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



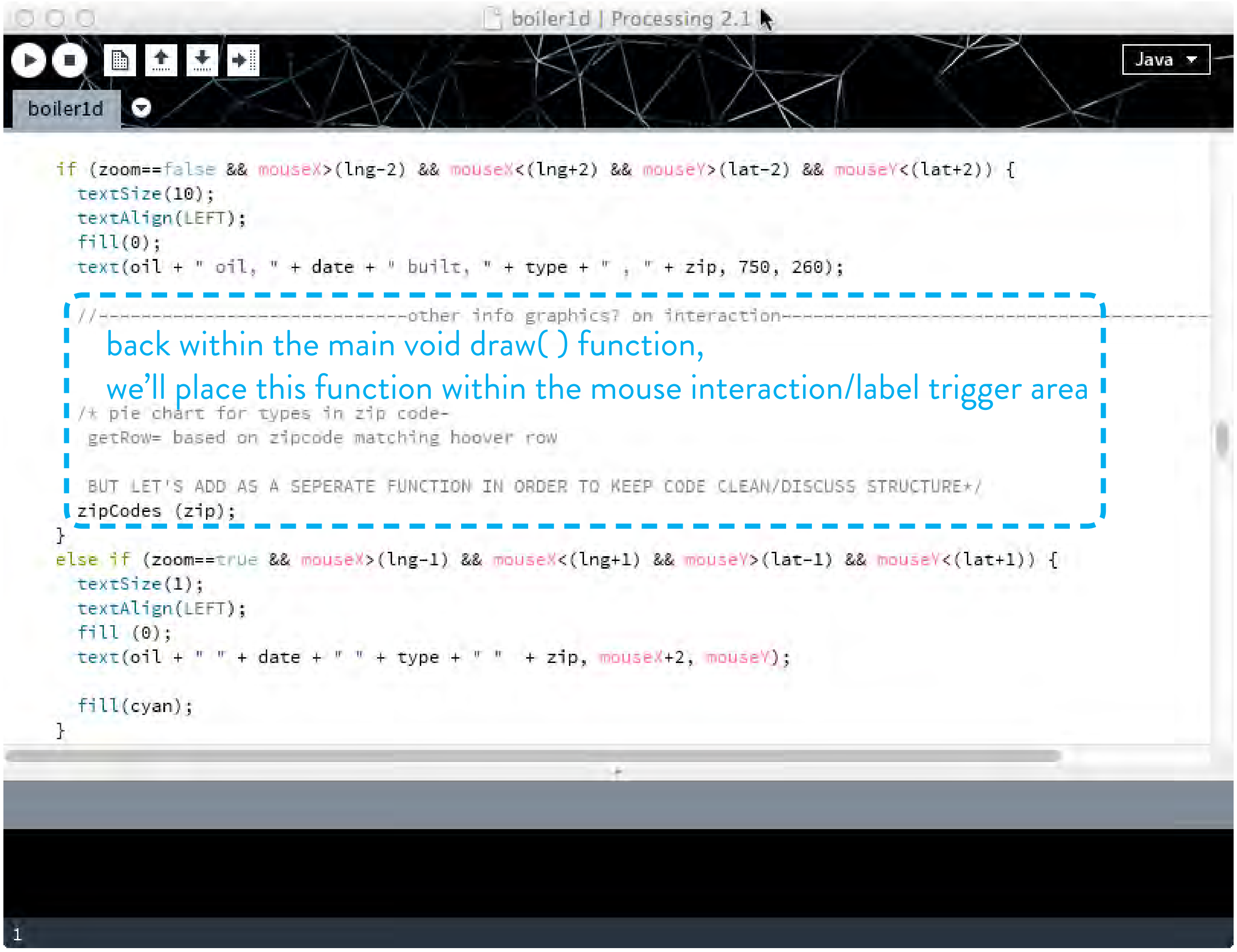
```
void zipCodes (String zipcoding) {

  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);
      }
```

*create new functions
void function(input variables){. . .}

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 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.



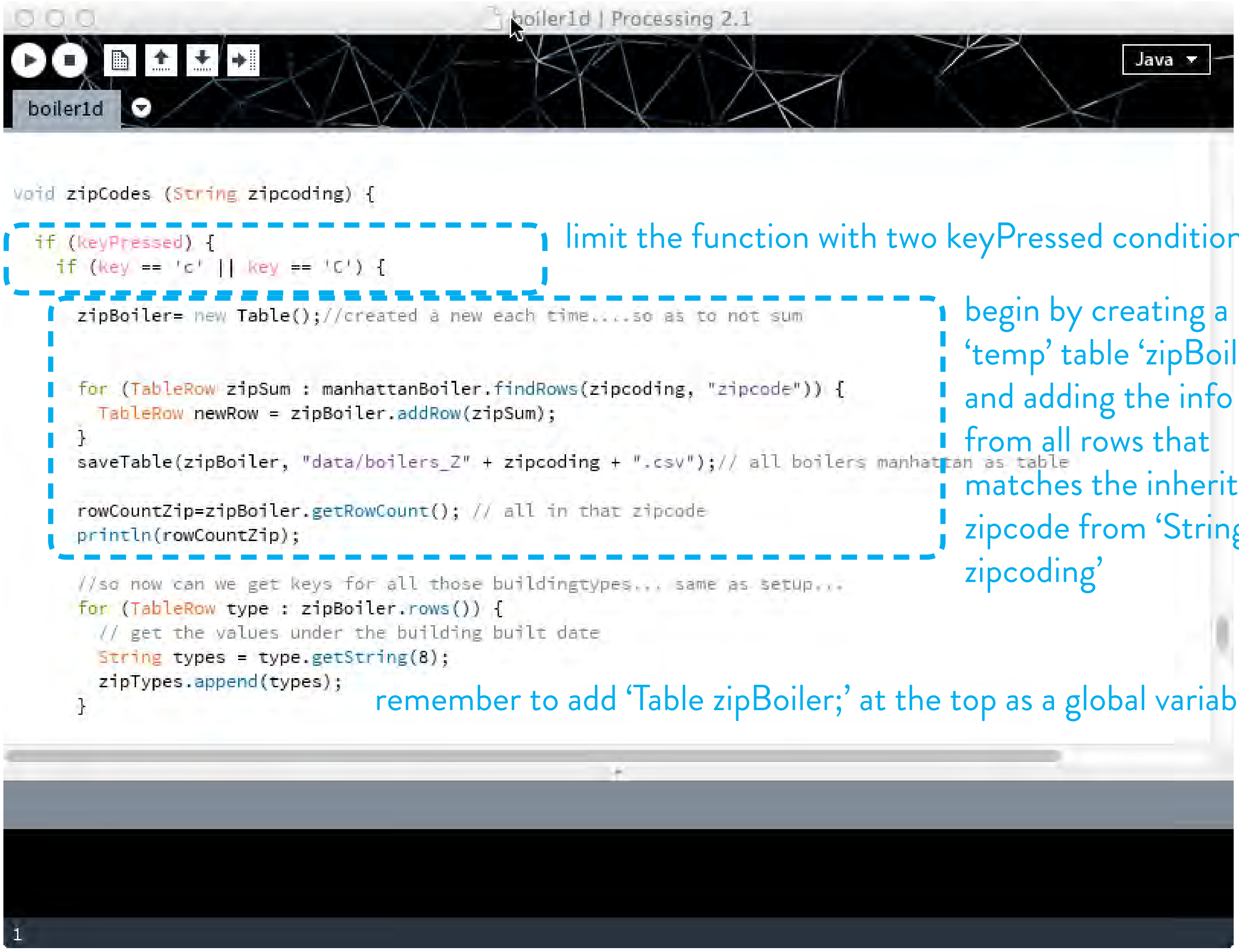
```
if (zoom==false && mouseX>(lng-2) && mouseX<(lng+2) && mouseY>(lat-2) && mouseY<(lat+2)) {
  textSize(10);
  textAlign(LEFT);
  fill(0);
  text(oil + " oil, " + date + " built, " + type + " , " + zip, 750, 260);

  //-----------------------------other info graphics? on interaction----------------------------------------------

  /* 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);
}
```

back within the main void draw( ) function,
we'll place this function within the mouse interaction/label trigger area

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

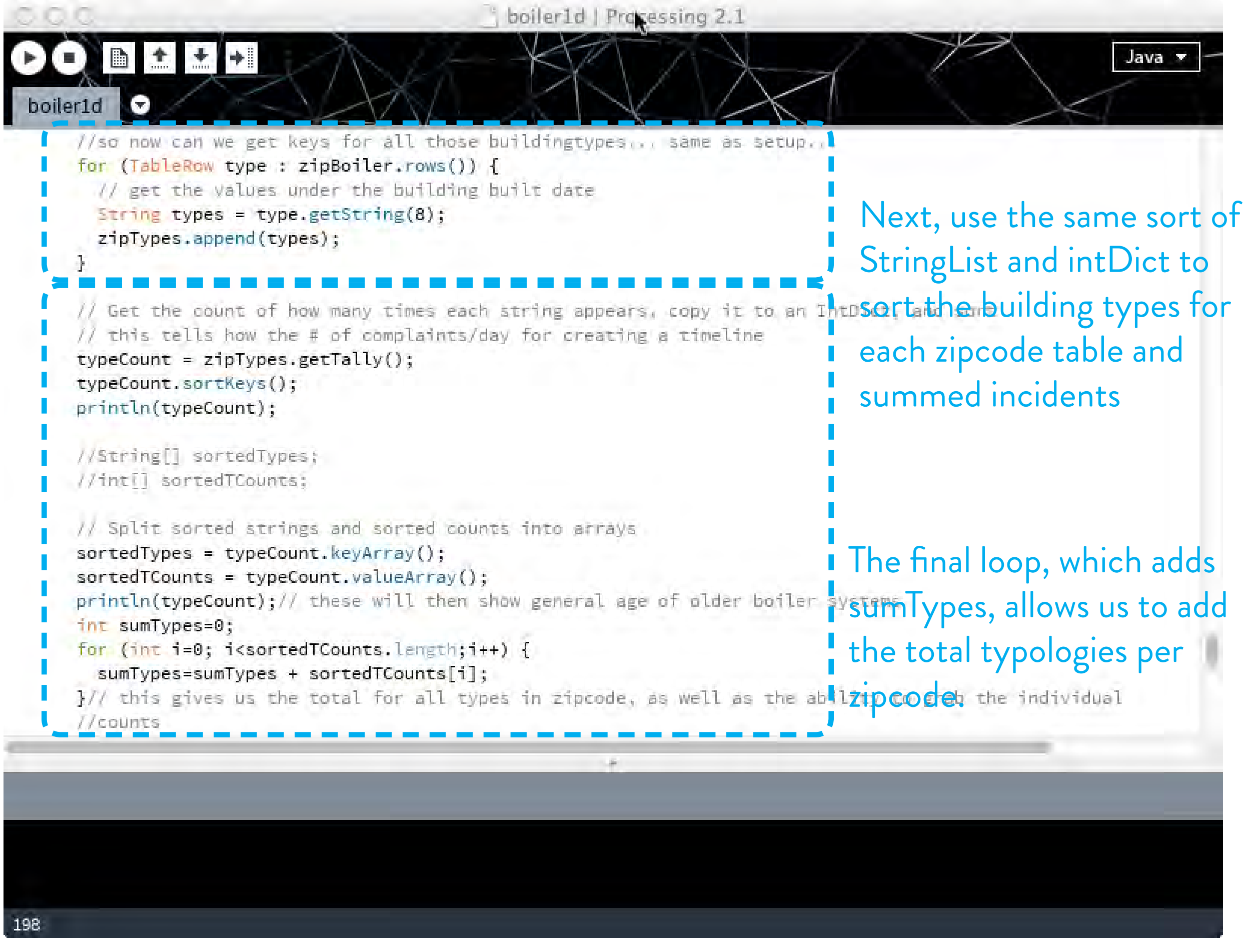
begin by creating a ‘temp’ table ‘zipBoiler’ and adding the info from all rows that matches the inherited zipcode from ‘String zipcoding’

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 hoover over a zipcoded area more than once, but the data in that file will be the same.



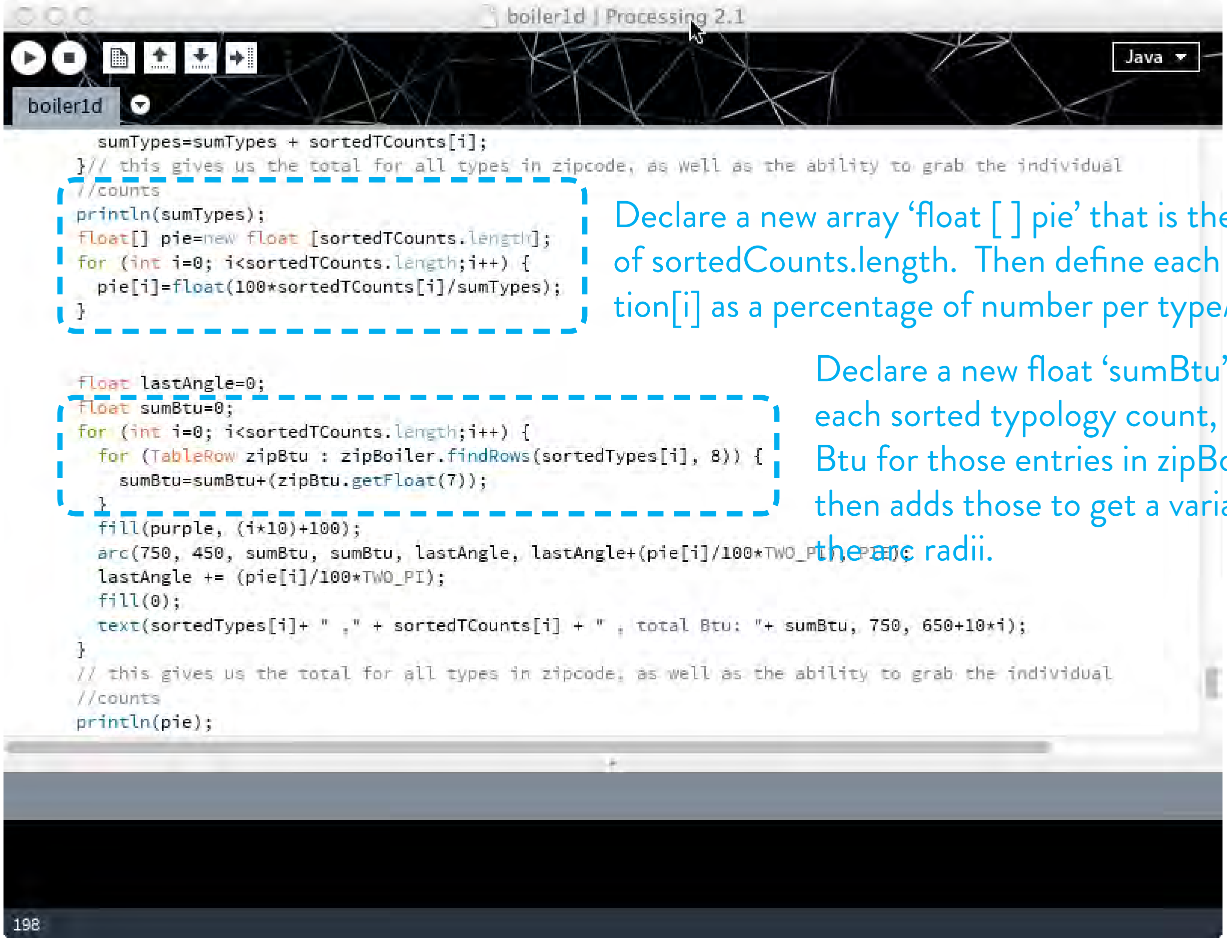
```
//so now can we get keys for all those buildingtypes... same as setup..
for (TableRow type : zipBoiler.rows()) {
  // get the values under the building built date
  String types = type.getString(8);
  zipTypes.append(types);
}

// Get the count of how many times each string appears, copy it to an IntDict, and sum
// 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.keyArray();
sortedTCounts = typeCount.valueArray();
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 grab 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.



```
    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);
```

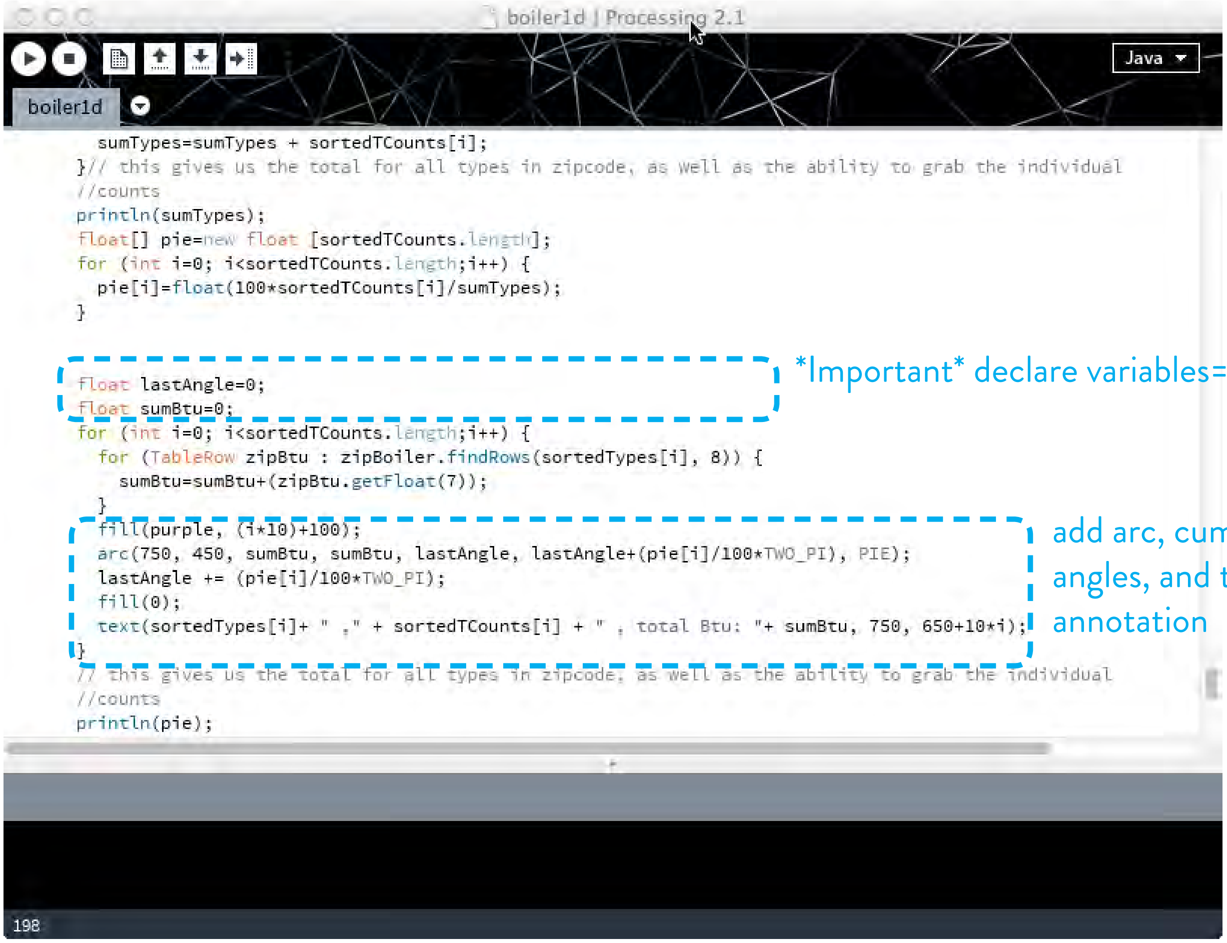
Declare a new array ‘float [ ] pie’ that is the lenght of sortedCounts.length. Then define each pie position[i] as a percentage of number per type/all types.

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.

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.



```
    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);
```

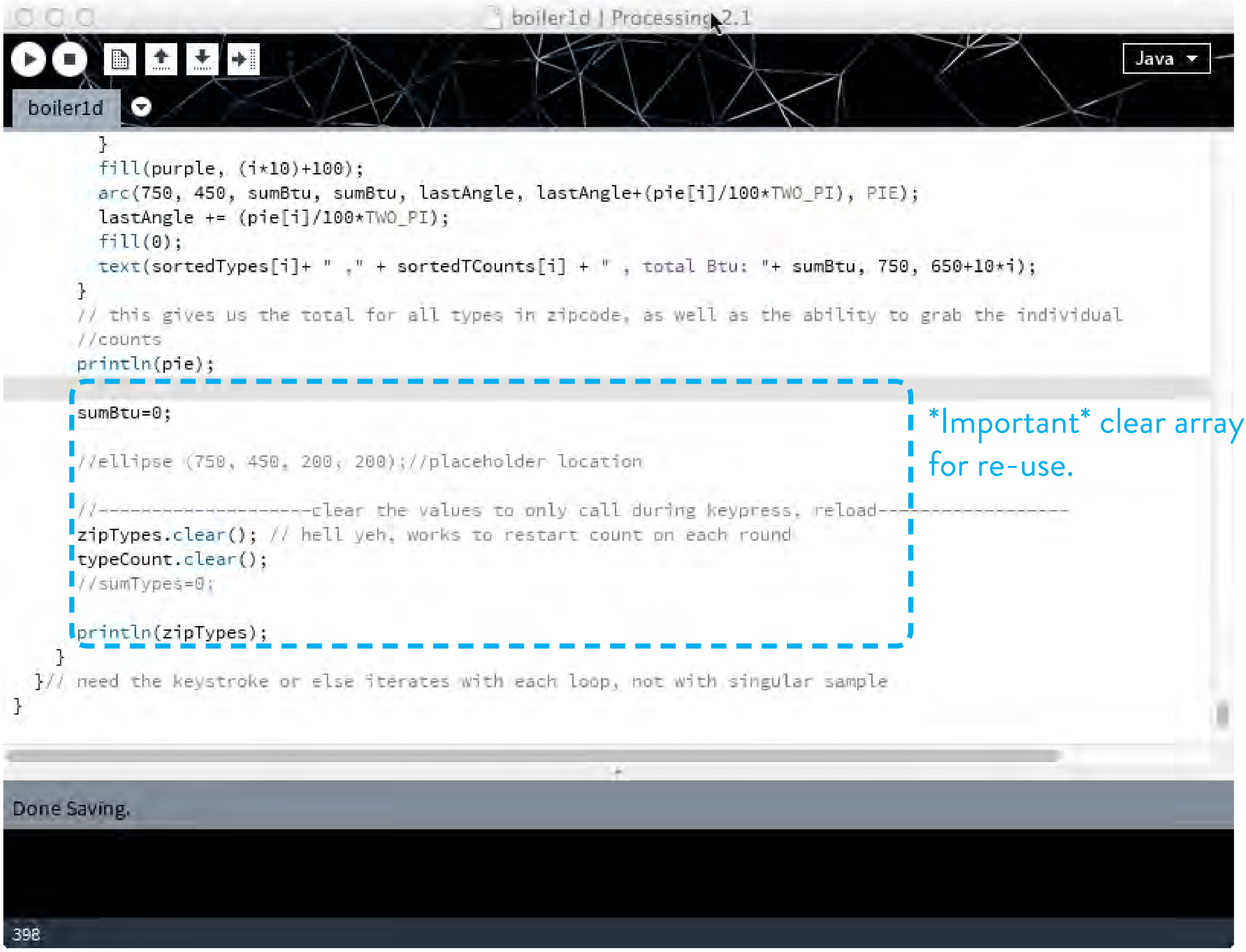
Important declare variables=0;

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 delared 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.



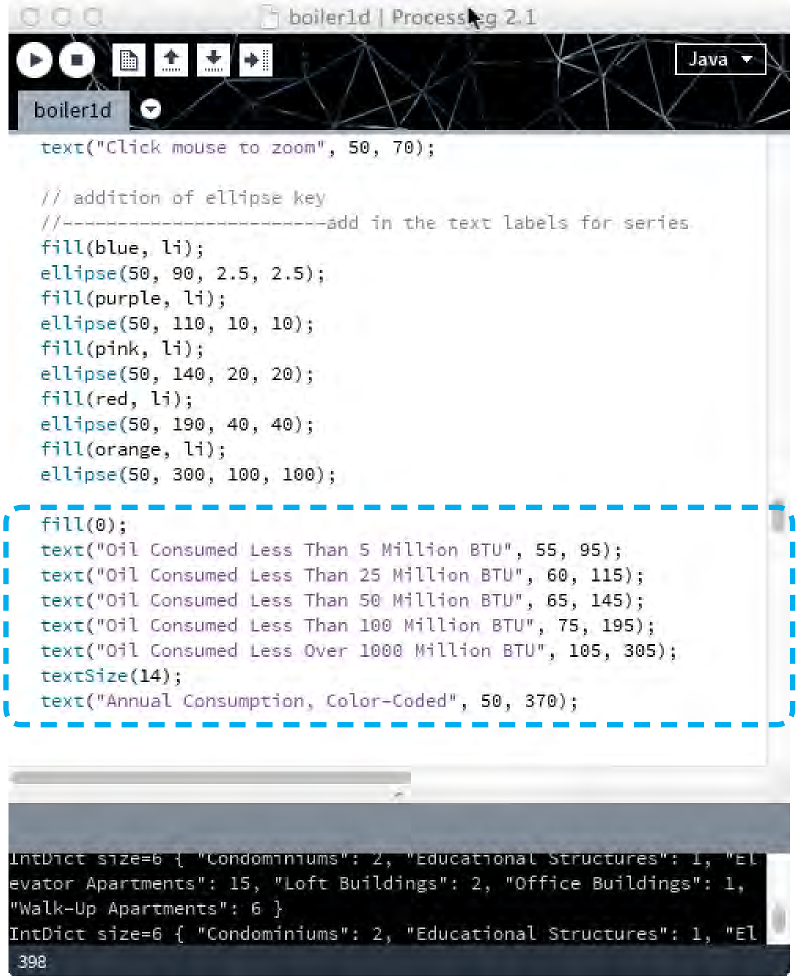
```
    }
    fill(purple, (i*10)+100);
    arc(750, 450, sumBtu, sumBtu, lastAngle, lastAngle+(pie[i]/100*TWO_PI), PIE);
    lastAngle += (pie[i]/100*TWO_PI);
    fill(0);
    text(sortedTypes[i]+ " ," + sortedTCounts[i] + " , total Btu: "+ sumBtu, 750, 650+10*i);
   }
   // this gives us the total 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
}
```

Done Saving.

Important clear arrays for re-use.

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 an not to get double, triple, quad (etc.) counts of both sums and objects in arrays.



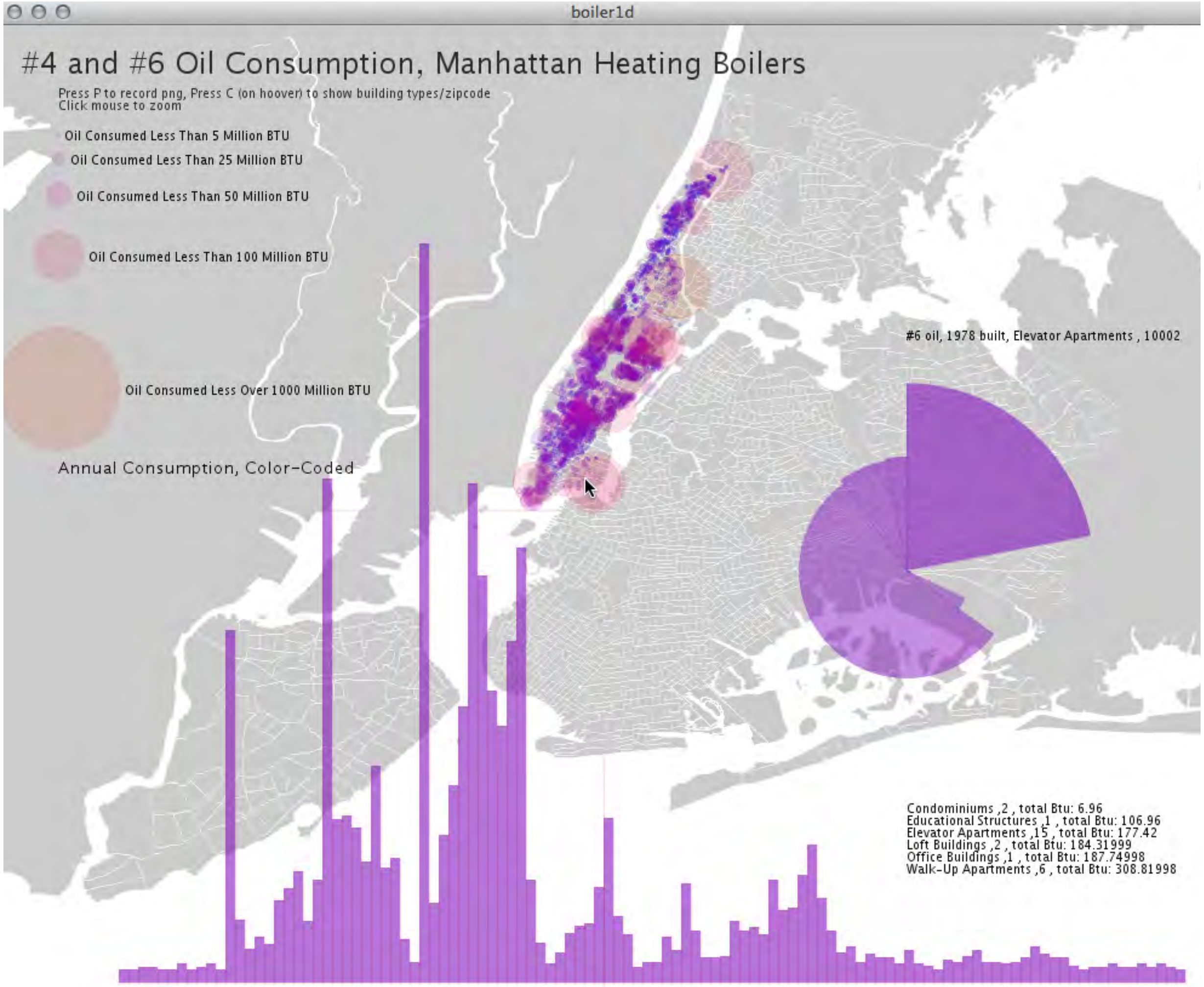
```
text("Click mouse to zoom", 50, 70);

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

Add some text labels for the keys.

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

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


boiler1d
Oil Consumed Less Than 5 Million BTU
Oil Consumed Less Than 25 Million BTU
Oil Consumed Less Than 50 Million BTU
Oil Consumed Less Than 100 Million BTU
Oil Consumed Less Over 1000 Million BTU
Annual Consumption, Color-Coded

#4 and #6 Oil Consumption, Manhattan Heating Boilers

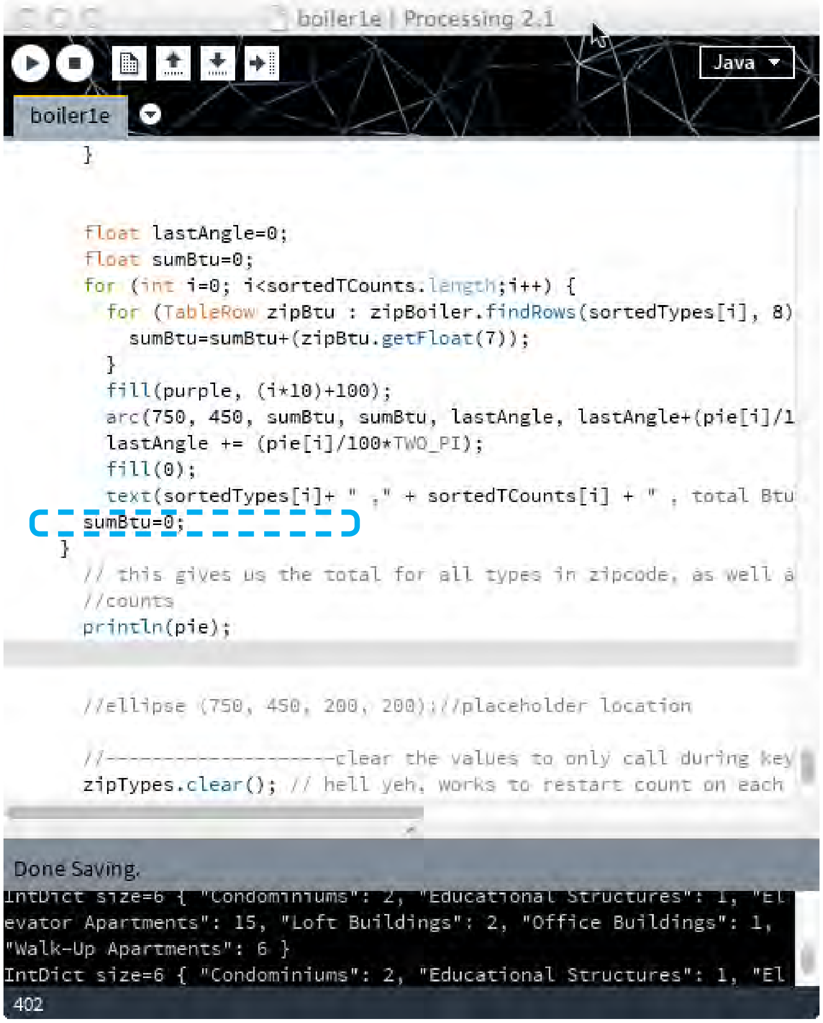
Press P to record png, Press C (on hoover) to show building types/zipcode
Click mouse to zoom

```
#6 oil, 1978 built, Elevator Apartments , 10002
```

```
Condominiums ,2 , total Btu: 6.96
Educational Structures ,1 , total Btu: 106.96
Elevator Apartments ,15 , total Btu: 177.42
Loft Buildings ,2 , total Btu: 184.31999
Office Buildings ,1 , total Btu: 187.74998
Walk-Up Apartments ,6 , total Btu: 308.81998
```

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.



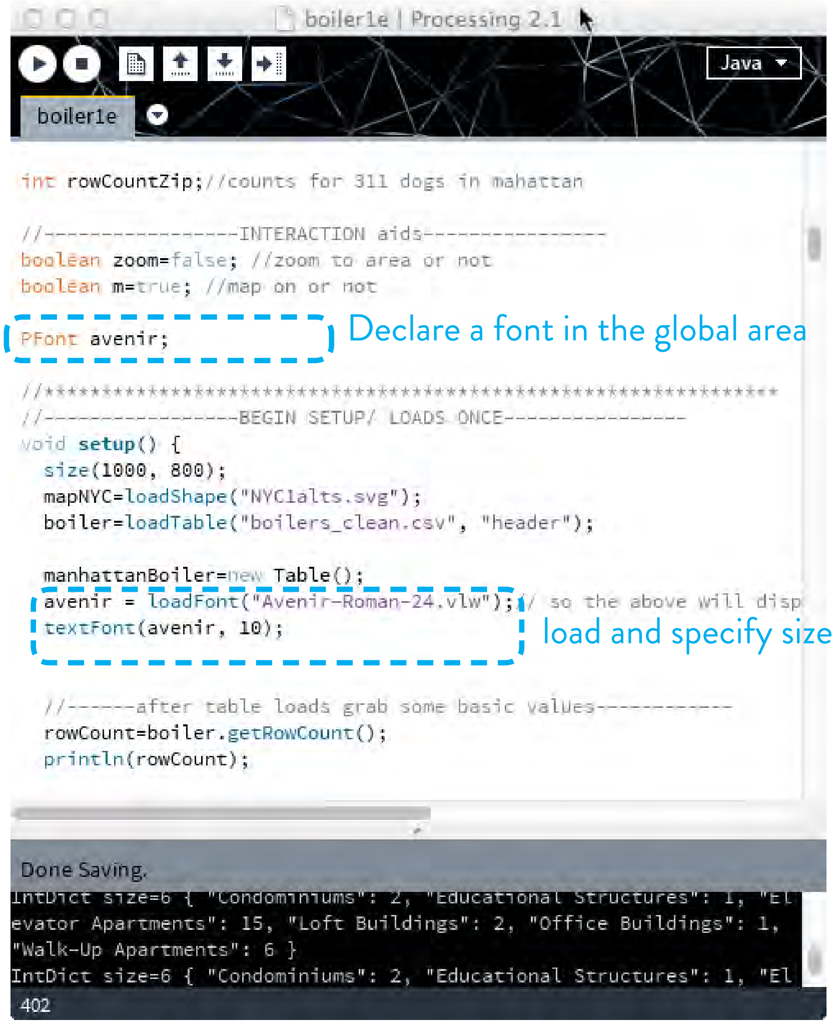
```
    }


    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 a
    //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
```

```
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
```

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 mahattan

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

PFont avenir;

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

  manhattanBoiler=new Table();
  avenir = loadFont("Avenir-Roman-24.vlw");// so the above will disp
  textFont(avenir, 10);

  //------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
```

Declare a font in the global area

load and specify size within void setup( )

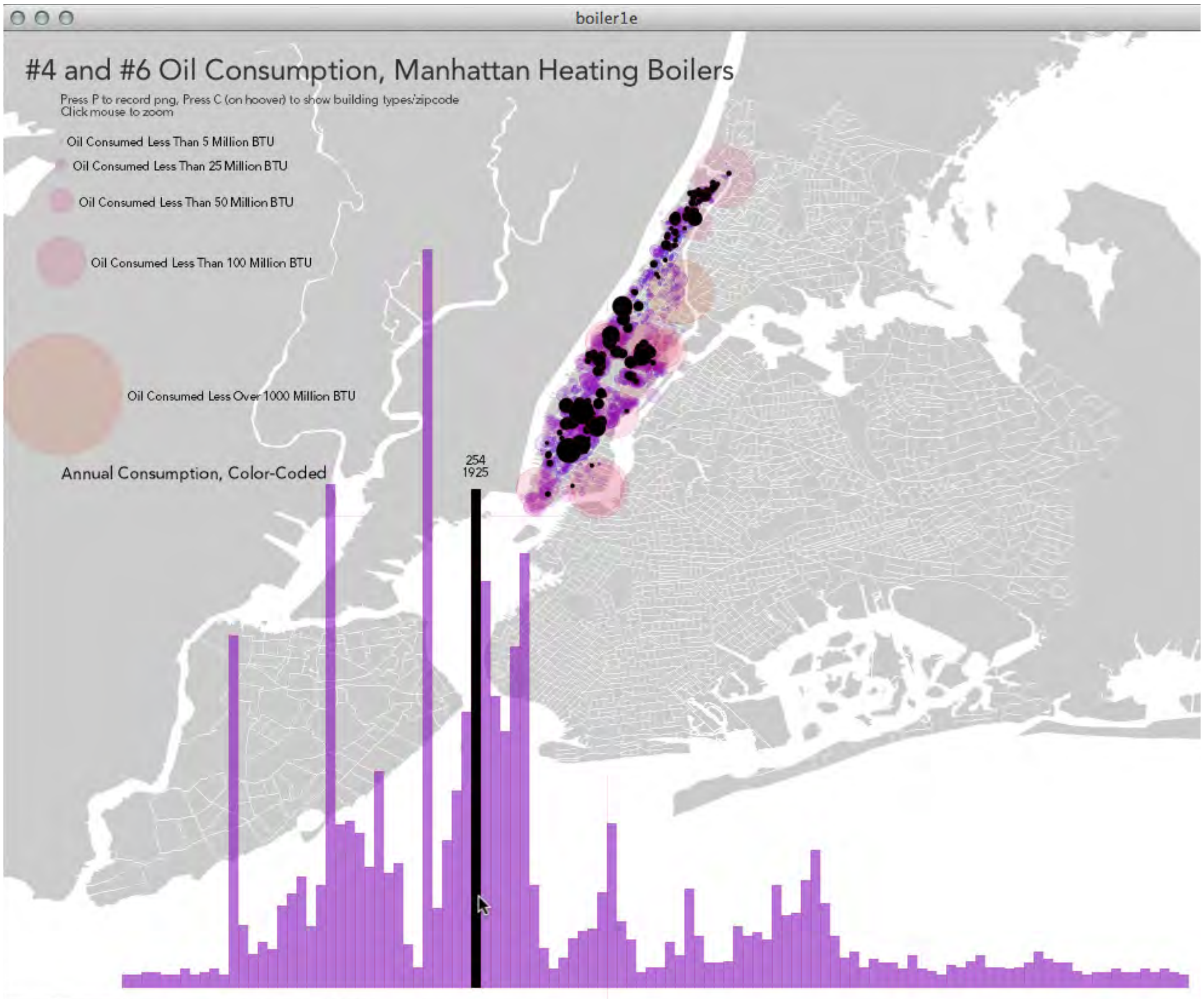
create fonts within the menu TOOLS/CREATE FONT

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.

WK 2-FINAL VIEW- BAR INTERACTIONS

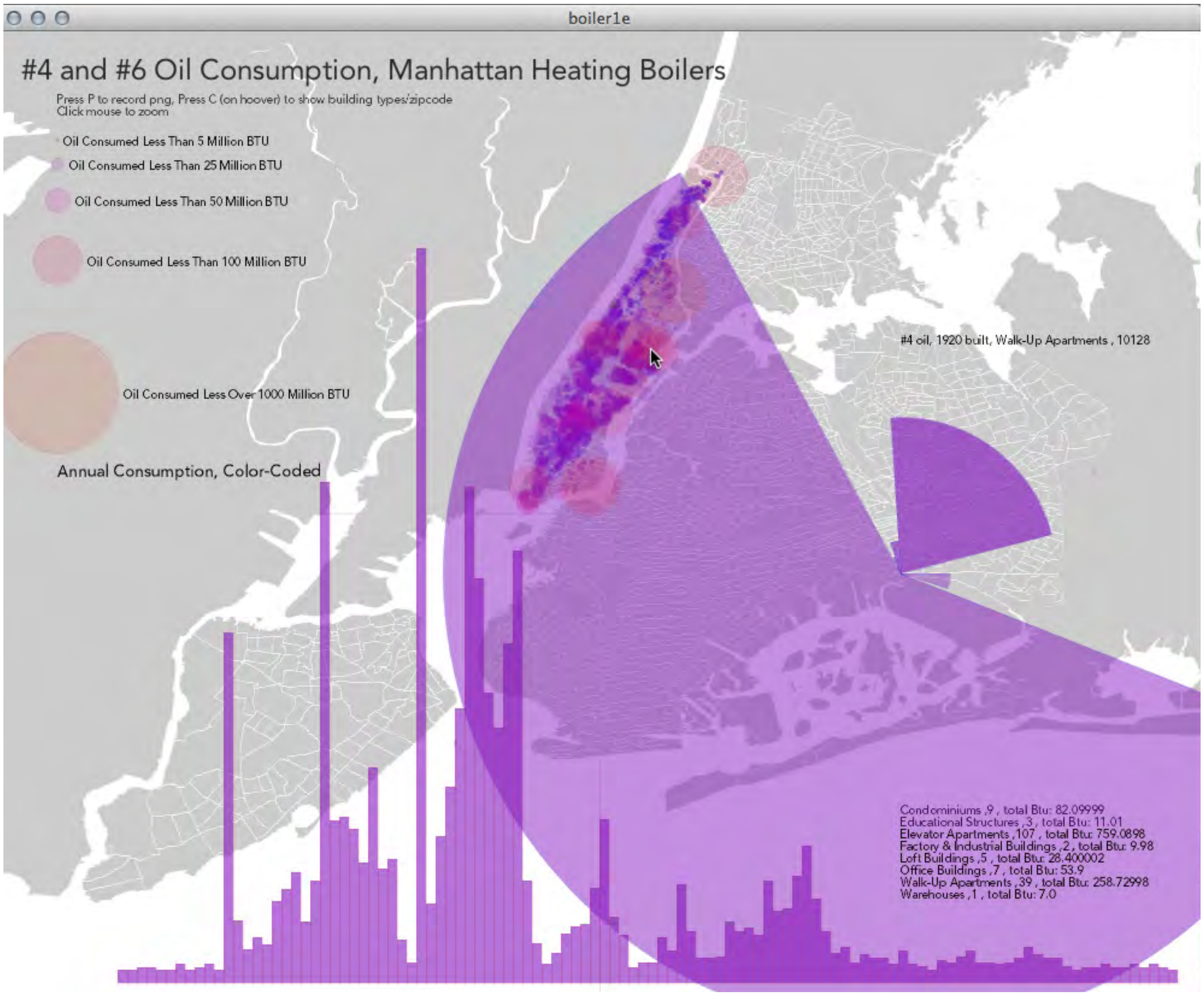


#4 and #6 Oil Consumption, Manhattan Heating Boilers

Press P to record png, Press C (on hoover) to show building types/zipcode
Click mouse to zoom

boiler1e
Oil Consumed Less Than 5 Million BTU
Oil Consumed Less Than 25 Million BTU
Oil Consumed Less Than 50 Million BTU
Oil Consumed Less Than 100 Million BTU
Oil Consumed Less Over 1000 Million BTU
Annual Consumption, Color-Coded
254
1925

SKETCH: BOILER 1E



boiler1e

#4 and #6 Oil Consumption, Manhattan Heating Boilers

Press P to record png, Press C (on hoover) to show building types/zipcode
Click mouse to zoom

Oil Consumed Less Than 5 Million BTU

Oil Consumed Less Than 25 Million BTU

Oil Consumed Less Than 50 Million BTU

Oil Consumed Less Than 100 Million BTU

Oil Consumed Less Over 1000 Million BTU

Annual Consumption, Color-Coded

#4 oil, 1920 built, Walk-Up Apartments , 10128

```
Condominiums ,9 , total Btu: 82.09999
Educational Structures ,3 , total Btu: 11.01
Elevator Apartments ,107 , total Btu: 759.0898
Factory & Industrial Buildings ,2 , total Btu: 9.98
Loft Buildings ,5 , total Btu: 28.400002
Office Buildings ,7 , total Btu: 53.9
Walk-Up Apartments ,39 , total Btu: 258.72998
Warehouses ,1 , total Btu: 7.0
```

SKETCH: BOILER 1E

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

- Moving toward correlation and arguments with data

- Introduce API and live web data, 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 pay-walls 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