

# Data Storage Formats

# How do you get the size of your SVG?

```
PShape square; // The PShape object

void setup() {
  size(100, 100);
  // Creating the PShape as a square. The corner
  // is 0,0 so that the center is at 40,40
  square = createShape(RECT, 0, 0, 80, 80);
}

void draw() {
  shape(square, 10, 10);
}
```

## Constructors

`PShape(g, kind, params)`

## Fields

`width` Shape document width

`height` Shape document height


Will not be 100% accurate, but good enough for a broad-phase collision check.

# How can we define an AABB?

The AABB should be the smallest axis-aligned box enclosing the shape.

Suppose we have points  $(x_1, y_1), (x_2, y_2), \dots$  in our shape.  
What should the xMin and xMax of the AABB be?

**What is the ideal class structure for  
giving objects AABBs?**



Box 1 centered at  $(x_1, y_1, z_1)$  with axial sizes of  $h_1, w_1, d_1$ .

Box 2 centered at  $(x_2, y_2, z_2)$  with axial sizes of  $h_2, w_2, d_2$ .

Boxes are **not** colliding if **any** of the following are true

$$|x_1 - x_2| > \frac{w_1 + w_2}{2}$$

$$|y_1 - y_2| > \frac{h_1 + h_2}{2}$$

$$|z_1 - z_2| > \frac{d_1 + d_2}{2}$$

So boxes **are** colliding if **all** of the following are true

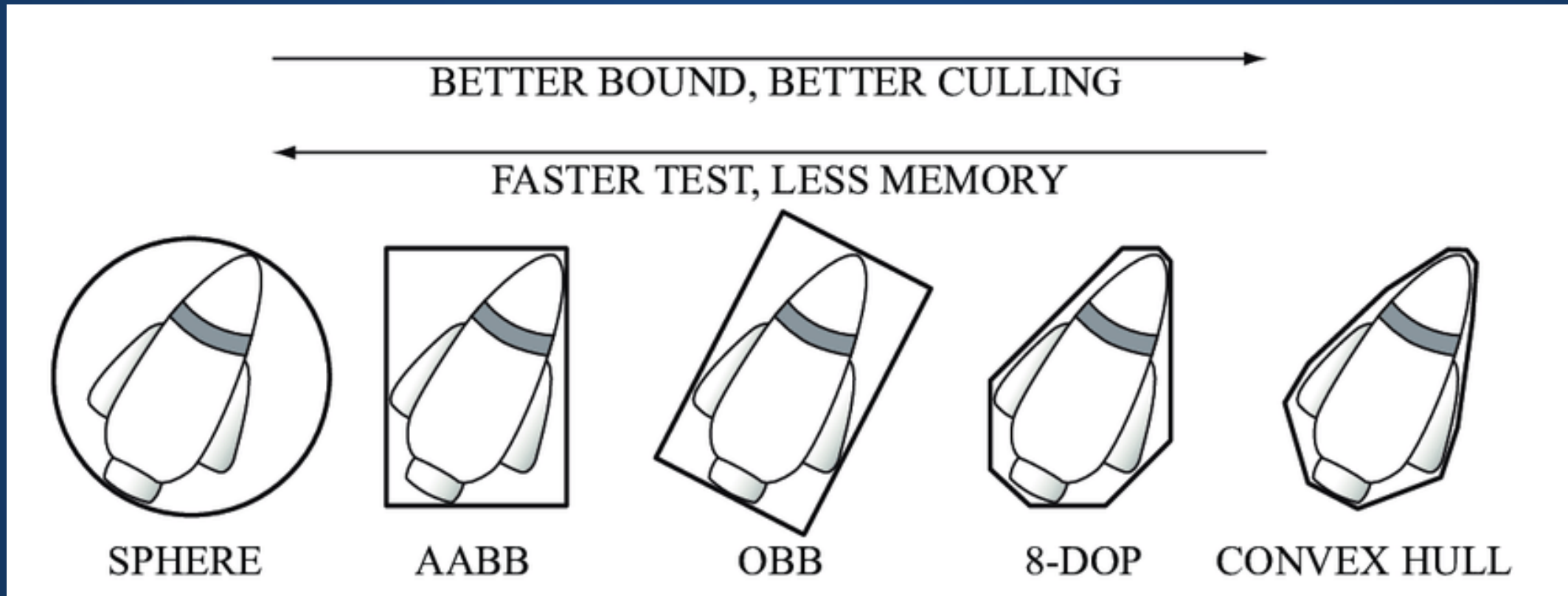
$$|x_1 - x_2| \leq \frac{w_1 + w_2}{2}$$

$$|y_1 - y_2| \leq \frac{h_1 + h_2}{2}$$

$$|z_1 - z_2| \leq \frac{d_1 + d_2}{2}$$

# Are circular hitboxes often used in bullet hell games?

If your hitbox is a circle, you'll get hit in circle, which may be a frustrating player experience.

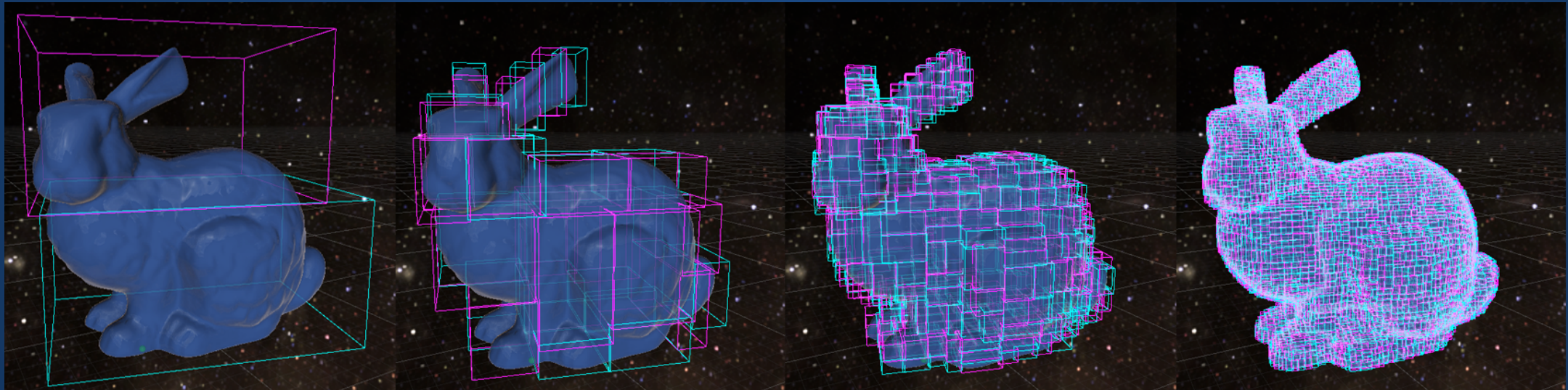


# How can we simulate the collision of a liquid with an object?

<https://www.youtube.com/embed/2gp7-ejkwBQ?enablejsapi=1>

**Suppose we have a cloth with lots of points. Once we bypass the broad-phase, do we have to check every single point in the cloth?**

No, we can segment the shape into smaller bounding volumes.



# If we want to learn about sprite animation, do you recommend any resources?

The old Lecture 18 had some information in it, along with the materials you need to make a simple sprite animation yourself.

Slides available on Canvas under Class Slides > cs324e-18-old

Hands-On materials (with sprite sources) available under Hands-On Materials > sprites-inclass.zip

Processing website has examples of sprites and image animation:  
<https://processing.org/examples#topics-animation>

If you prefer videos, there do exist YouTube tutorials on the subject, e.g <https://www.youtube.com/watch?v=oiWmBVlu350>.



# Data Storage Types

# Ordered Collection

A row of boxes which can store objects. Accessed by specifying the number of the box you want.

## Python

```
1 my_lst = [1,2,3,5]
2 x = my_lst[3]
3
4 my_lst.append(7)
5 y = my_lst[4]
```

# Arrays

Arrays are *fixed-length* and store values of a given type.

```
int[] values = new int[10];
```

Index to access data stored within the array.

```
int x = values[4];
```

Array has fixed length upon creation which cannot be changed.

```
int l = values.length;
```

# ArrayLists

Structure of *mutable* length that can hold any object (data blob) of declared type.

```
ArrayList<int> v = new ArrayList<int>();
```

Method	Action
v.add(obj)	Add obj to the back of the ArrayList
v.remove(i)	Remove the object at index i, shifting tail left
v.get(i)	Get the object at index i
v.set(i, obj)	Set the object at index i to obj
size()	Return the size of the ArrayList

# Convenience Lists

ArrayLists are flexible and allow for us to create arrays whose length we don't know ahead of time.

Unfortunately, due to restrictions in Java\*, there is a minor performance penalty for accessing them.

Processing offers us specialized versions of ArrayList that only work with a particular type, but do not pay these overheads.

`FloatList`, `IntList`, `StringList`

\*if you're interested, you can look for information on type erasure and generics in Java

# Unordered Collections

(These turn out not to be that important for what we're about to do, so I'm mostly putting them here for completeness).

Like ordered collections, but there is no order. ̄\\_(\ツ)\\_̄

You can't ask for a specific element of the collection. You can ask for an arbitrary element or to iterate over it.

## Python

```
1 s = set([1,2,3,4])
2 for elem in s:
3     print(elem)
```

## Java

```
1 HashSet<String> set = new HashSet<>();
2 set.add("Apple");
3 set.add("Banana");
4 set.remove("Apple");
5 System.out.println(set);
```

# Key-Value Stores

# Key-Value Stores

Instead of storing things in order, we give each object we want to store a "key", a small piece of data associated with it.

Later, when we want to get the whole object, we can look it up by the key.

Key

Value

999-11-2968



```
1 {
2   "name": "John Doe",
3   "age": 30,
4   "email": "johndoe@example.com",
5   "address": {
6     "street": "123 Main Street",
7     "city": "Cityville",
8     "state": "State",
9     "postalCode": "12345"
10  },
11  "phoneNumbers": [
12    "+1 555-123-4567",
13    "+1 555-987-6543"
14  ],
15  "hobbies": ["Reading", "Hiking", "Cooking"]
16 }
```



# Processing Dictionaries

Dictionary where key is a `String` and value is of the specified type.

- `FloatDict`
- `IntDict`
- `StringDict`

<b>Method</b>	<b>Action</b>
<code>d.set("key", value)</code>	Adds the key-value pair to the dict
<code>d.get("key")</code>	Retrieves the value assigned to key
<code>d.hasKey("key")</code>	Returns a boolean--whether "key" is in the dictionary or not

# Java HashMaps

A more flexible key-value store than a Processing dictionary: keys and values can be of any (fixed) type.

```
1 import java.util.HashMap;
2
3 // Create a new HashMap
4 HashMap<Integer, Float> map = new HashMap<>();
5
6 // Add key-value pairs to the HashMap
7 map.put(1000, 3.5f);
8 map.put(2274, 2.8f);
9 map.put(3896, 4.2f);
10
11 // Get the value associated with a key
12 float value = map.get(2274);
13
14 // Check if a key is present in the HashMap
15 boolean containsKey = map.containsKey(1);
```

**What's another data type we've learned about that can be a key-value store?**

```
1 import java.util.HashMap;
2
3 // Create a new HashMap
4 HashMap<String, Float> particle1 = new HashMap<>();
5
6 particle1.put("x", 0.0);
7 particle1.put("vx", 0.0);
8 particle1.put("ax", 1.0);
```

## versus

```
1 Particle particle1 = new Particle();
2
3 particle1.x = 0.0;
4 particle1.vx = 0.0;
5 particle1.ax = 1.0;
```

When should we create a class for our data versus store data in  
HashMaps/Dictionaryes?

# File Storage

Data is stored in files like OBJ, SVG, etc.

We need some way to turn this data into something that can be represented in-memory in the programming language.

Examples:

- SVG File → PShape (2D)
- OBJ File → PShape (3D)
- PNG File → PImage

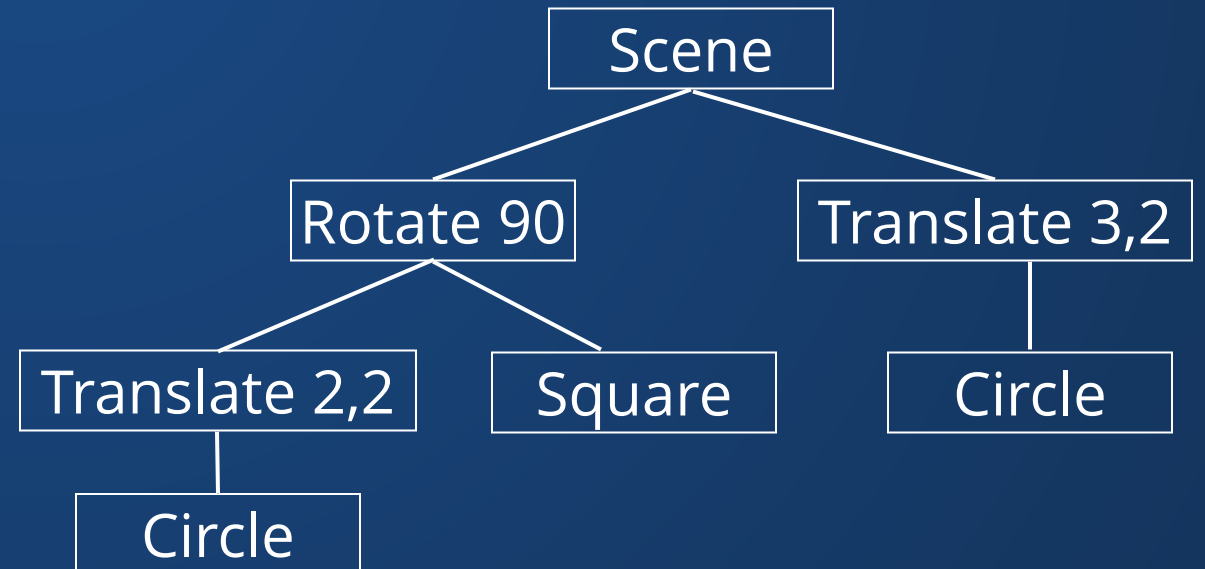
Input data is stored in files as a single linear stream of data.  
It needs to be *parsed* to be turned into a thing with structure.

```
1 Scene(  
2   rotate(90,  
3     translate(2,2, Circle()),  
4     Square(),  
5   ),  
6   translate(3,2, Circle()),  
7 )
```

Before parse

```
Scene(rotate(90,translate(2,2,  
Circle()),Square()),translate(3,  
2, Circle()),)
```

After Parse



# Document Formats

Specific document formats provide a consistent style of storage between programs. This aids parsing both for machines and humans (human parsing often referred to as "reading").

Processing provides support for reading/writing of three document types:

- CSV
- XML
- JSON

We're going to cover CSV and JSON because I hate XML with a passion 😡

# CSV



1 -159.3963,21.97066,"Starbucks: Lihue","Kukui Grove Shopping Center 3-2600 Kaunualii Hwy; Lihue, HI 96766"  
2 -158.0274,21.34273,"Starbucks: Honolulu","91-1401 Fort Weaver Rd, Ewa Beach; Honolulu, HI 96706; (808) 685- 4594"  
3 -158.012,21.46285,"Starbucks: Mililani","95-221 Kipapa Drive; Mililani, HI 96789; (808) 625- 5401"  
4 -158.0061,21.40369,"Starbucks: Waipahu","94-799 Lumiaina St; Waipahu, HI 96797"  
5 -157.9313,21.37066,"Starbucks: Aiea","4561 Salt Lake Blvd; Aiea, HI"  
6 -157.9148,21.33524,"Starbucks: Honolulu","550 Paiea Street Airport Trade Center; Honolulu, HI 96819"  
7 -157.8615,21.30846,"Starbucks: Honolulu","1000 Bishop Street #104; Honolulu, HI 96813; (808) 599- 4833"  
8 -157.8609,21.30759,"Starbucks: Honolulu","220 South King Street; Honolulu, HI 96813"  
9 -157.8471,21.29308,"Starbucks: Honolulu","Ala Moana Center 1450 Ala Moana Blvd Sears & Macys; Honolulu, HI 96814"  
10 -157.8326,21.27913,"Starbucks: Honolulu","2169 Kalia Road Waikiki; Honolulu, HI 96815"  
11 -157.8294,21.27954,"Starbucks: Honolulu","2222 Kalakaua Avenue; Honolulu, HI 96815"  
12 -157.827,21.28014,"Starbucks: Honolulu","2255 Kuhio Avenue, Suite S-1; Honolulu, HI 96815"  
13 -157.7876,21.27852,"Starbucks: Honolulu","4211 Waiialae Avenue; Honolulu, HI 96816; (808) 737- 0283"  
14 -157.7855,21.40587,"Starbucks: Honolulu","45-480 Kaneohe Bay Drive AlE; Honolulu, HI 96744; (808) 234- 6900"  
15 -157.7403,21.39357,"Starbucks: Kailua","539 Kailua Road; Kailua, HI 96734; (808) 263- 9548"  
16 -157.713,21.28201,"Starbucks: Honolulu","6700 Kalaniana'ole Highway; Honolulu, HI 96825"  
17 -156.6818,20.8832,"Starbucks: Lahaina","1221 Honoapiilani Highway Space A-1; Lahaina, HI 96761"  
18 -156.4742,20.88891,"Starbucks: Kahului","275 West Kaahuman Ave Suite 1200 Space # F-5; Kahului, HI 96732"  
19 -156.4556,20.88118,"Starbucks: Kahului","270 Dairy Road; Kahului, HI 96732; (808) 871- 7884"  
20 -156.4536,20.73723,"Starbucks: Kihei","1819 S. Kihei Rd. Unit D-112; Kihei, HI 96738"  
21 -155.9889,19.64723,"Starbucks: Kailua-Kona","75-1015 Henry St.; Kailua-Kona, HI 96740"  
22 -151.5312,59.64204,"Starbucks: Homer","90 Sterling Hwy; Homer, AK 99603"  
23 -151.0418,60.49444,"Starbucks: Soldotna","44428 Sterling Hwy S; Soldotna, AK 99669"  
24 -149.9093,61.13995,"Starbucks: Anchorage","2000 W Dimond Blvd; Anchorage, AK 99515-1400; (907) 267- 6700"  
25 -149.9055,61.19534,"Starbucks: Anchorage","1650 W Northern Lights Blvd; Anchorage, AK 99503; (907) 339- 0500"  
26 -149.8936,61.21759,"Starbucks: Anchorage","601 West 5th Avenue; Anchorage, AK 99501; (907) 277- 2477"  
27 -149.8643,61.19525,"Starbucks: Anchorage","1000 E Northern Lights Blvd; Anchorage, AK 99508-4283; (907) 264- 9600"  
28 -149.8638,61.14473,"Starbucks: Anchorage","1005 E Dimond Blvd; Anchorage, AK 99515; (907) 344- 4160"  
29 -149.8446,61.13806,"Starbucks: Anchorage","1725 Abbott Rd; Anchorage, AK 99507; (907) 339- 2800"  
30 -149.838,61.13751,"Starbucks: Anchorage","2300 Abbott Road; Anchorage, AK 99507; (907) 365- 2000"  
31 -149.8359,61.18082,"Starbucks: Anchorage","2421 East Tudor Road Suite 8; Anchorage, AK 99507; (907) 561- 1644"  
32 -149.7365,61.19533,"Starbucks: Anchorage","7731 E Northern Lights Blvd; Anchorage, AK 99504; (907) 331- 1700"  
33 -149.5708,61.32361,"Starbucks: Eagle River","11409 Business Blvd; Eagle River, AK 99577; (907) 696- 6400"  
34 -149.5525,61.34592,"Starbucks: Eagle River","13401 Old Glenn Hwy; Eagle River, AK 99577"  
35 -149.1183,61.60094,"Starbucks: Palmer","650 N Cobb St; Palmer, AK 99645; (907) 761- 4200"  
36 -147.8223,64.83696,"Starbucks: Fairbanks","3755 Airport Way; Fairbanks, AK 99709-4610; (907) 474- 1403"

```
1 surname,percent,cumulative percent,rank
2 SMITH,1.006,1.006,1
3 JOHNSON,0.810,1.816,2
4 WILLIAMS,0.699,2.515,3
5 JONES,0.621,3.136,4
6 BROWN,0.621,3.757,5
7 DAVIS,0.480,4.237,6
8 MILLER,0.424,4.660,7
9 WILSON,0.339,5.000,8
10 MOORE,0.312,5.312,9
11 TAYLOR,0.311,5.623,10
12 ANDERSON,0.311,5.934,11
13 THOMAS,0.311,6.245,12
14 JACKSON,0.310,6.554,13
15 WHITE,0.279,6.834,14
16 HARRIS,0.275,7.109,15
17 MARTIN,0.273,7.382,16
18 THOMPSON,0.269,7.651,17
19 GARCIA,0.254,7.905,18
```

# CSV

- Stores *tabular* data, i.e. data in a table/grid
- Rows provide record information, i.e. a single "object's" data
- Fields within the record (i.e. columns) store information
- A flat structure
- Tabs or commas separate fields
- Newline characters separate records

*“ Data comes in rectangles!*

–Rob Tibshirani

# Tables in Processing

```
Table tbl = loadTable(filename, options);
```

Generates a `Table` object.

Options are strings separated by commas.

- "header" automatically interprets a header row
- "csv" tells Processing to interpret commas as separators
- "tsv" tells processing to interpret tabs as separators
- "newlines" tells Processing that there may be newline characters within a cell.

# Processing Tables

Tables are made of `TableRows`. We can get the number of rows/columns with `getRowCount()` and `getColumnCount()`.

We can get the *i*-th row with `getRow(i)` or all rows with `rows()`.

I highly recommend scanning the [Table docs](#) in Processing.

# TableRows

TableRows support several methods: `getInt()`, `getFloat()`, and `getString()`.

These support two types of arguments:

- `getThing(int)`: gets the appropriate index
- `getThing("name")`: gets the appropriate column name (based on the CSV header)

If the data to be retrieved does not exist, or if it's of the wrong type, you'll get an error.

```
1 for (TableRow r : t.rows()) {
2     String id = r.getString("object");
3     float x = r.getFloat("x");
4     float y = r.getFloat("y");
5 }
```

# Hands-On: CSV

1. Download one of the CSVs from [https://domo-support.domo.com/s/article/360043931814?language=en\\_US](https://domo-support.domo.com/s/article/360043931814?language=en_US). You **must** use one of the CSVs from this site.
2. Load the CSV into Processing.
3. Read the values from the CSV into some sort of non-Table data structure. An example might be an `ArrayList` or a `FloatList`.
4. Visualize the values on the screen in some way. You do not necessarily have to make a plot or a map, but there should be some visual representation of the data.

Please include your CSV with the submission.

# JSON



# Hierarchical Data

CSV is great for when data is flat and uniform.

```
1 Scene(  
2   rotate(90,  
3     translate(2,2, Circle()),  
4     Square(),  
5   ),  
6   translate(3,2, Circle()),  
7 )
```

**How can we represent this as a CSV?**

# JSON

A notation for attribute-value pairs. Designed to represent the JavaScript object model (hence the name).

Four basic types for data:

- Number
- Boolean
- String
- Null

Two aggregate types of data:

- Object (key-value)
- Array (ordered)

## Simple JSON Object

```
{  
  key1: value1,  
  key2: value2,  
  ...  
}
```

```
1 {  
2   "name": "Max",  
3   "breed": "Labrador Retriever",  
4   "age": 5,  
5   "weight": 25.7,  
6   "isFriendly": true,  
7 }
```

## Simple JSONArray

```
[  
  value1,  
  value2,  
  ...  
]
```

```
1 [  
2   "a",  
3   true,  
4   3,  
5   4,  
6   false,  
7 ]
```

# JSON

## JSON Object

{key: value}

```
1 {
2   "name": "Max",
3   "breed": "Labrador Retriever",
4   "age": 5,
5   "weight": 25.7,
6   "isFriendly": true,
7   "owner": {
8     "name": "John Smith",
9     "email": "johnsmith@example.com"
10  },
11  "vaccinations": [
12    "Rabies",
13    "Distemper",
14    "Parvovirus"
15  ]
16 }
```

## JSON Array

[value1, value2, ...]

```
1 [
2   {
3     "name": "Max",
4     "breed": "Labrador Retriever",
5     "age": 5,
6     "weight": 25.7,
7     "isFriendly": true
8   },
9   {
10    "name": "Charlie",
11    "breed": "German Shepherd",
12    "age": 4,
13    "weight": 30.2,
14    "isFriendly": false
15  }
16 ]
```

```
1 Scene(  
2   rotate(90,  
3     translate(2,2, Circle()),  
4     Square()),  
5 ),  
6 translate(3,2, Circle()),  
7 )
```

```
1 [  
2   "rotate": {  
3     "degrees": 90,  
4     "elements": [  
5       {  
6         "translate": {  
7           "pos": [2, 2],  
8           "elements": [  
9             {"shape": "circle"}  
10          ],  
11         }  
12      },  
13      {"shape": "square"}  
14    ]  
15  },  
16  "translate": {  
17    "pos": [3, 2],  
18    "elements": [  
19      {"shape": "circle"}  
20    ]  
21  }  
22 ]
```

```
1 { "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA", "_id" : "01001"
2 { "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" : 36963, "state" : "MA", "_id" :
3 { "city" : "BARRE", "loc" : [ -72.108354000000001, 42.409698 ], "pop" : 4546, "state" : "MA", "_id" : "0
4 { "city" : "BELCHERTOWN", "loc" : [ -72.410953000000001, 42.275103 ], "pop" : 10579, "state" : "MA", "_i
5 { "city" : "BLANDFORD", "loc" : [ -72.936114, 42.182949 ], "pop" : 1240, "state" : "MA", "_id" : "01008
6 { "city" : "BRIMFIELD", "loc" : [ -72.188455, 42.116543 ], "pop" : 3706, "state" : "MA", "_id" : "01010
7 { "city" : "CHESTER", "loc" : [ -72.988761, 42.279421 ], "pop" : 1688, "state" : "MA", "_id" : "01011"
8 { "city" : "CHESTERFIELD", "loc" : [ -72.833309, 42.38167 ], "pop" : 177, "state" : "MA", "_id" : "0101
9 { "city" : "CHICOPEE", "loc" : [ -72.607962, 42.162046 ], "pop" : 23396, "state" : "MA", "_id" : "01013
10 { "city" : "CHICOPEE", "loc" : [ -72.576142, 42.176443 ], "pop" : 31495, "state" : "MA", "_id" : "01020
11 { "city" : "WESTOVER AFB", "loc" : [ -72.558657, 42.196672 ], "pop" : 1764, "state" : "MA", "_id" : "01
12 { "city" : "CUMMINGTON", "loc" : [ -72.905767, 42.435296 ], "pop" : 1484, "state" : "MA", "_id" : "0102
13 { "city" : "MOUNT TOM", "loc" : [ -72.67992099999999, 42.264319 ], "pop" : 16864, "state" : "MA", "_id"
14 { "city" : "EAST LONGMEADOW", "loc" : [ -72.505565, 42.067203 ], "pop" : 13367, "state" : "MA", "_id" :
15 { "city" : "FEEDING HILLS", "loc" : [ -72.675077, 42.07182 ], "pop" : 11985, "state" : "MA", "_id" : "0
16 { "city" : "GILBERTVILLE", "loc" : [ -72.19858499999999, 42.332194 ], "pop" : 2385, "state" : "MA", "_i
17 { "city" : "GOSHEN", "loc" : [ -72.844092, 42.466234 ], "pop" : 122, "state" : "MA", "_id" : "01032" }
18 { "city" : "GRANBY", "loc" : [ -72.52000099999999, 42.255704 ], "pop" : 5526, "state" : "MA", "_id" : "
19 { "city" : "TOLLAND", "loc" : [ -72.908793, 42.070234 ], "pop" : 1652, "state" : "MA", "_id" : "01034"
20 { "city" : "HADLEY", "loc" : [ -72.571499, 42.36062 ], "pop" : 4231, "state" : "MA", "_id" : "01035" }
21 { "city" : "HAMPDEN", "loc" : [ -72.43182299999999, 42.064756 ], "pop" : 4709, "state" : "MA", "_id" :
22 { "city" : "HATFIELD", "loc" : [ -72.616735000000001, 42.38439 ], "pop" : 3184, "state" : "MA", "_id" :
23 { "city" : "HAYDENVILLE", "loc" : [ -72.70317799999999, 42.381799 ], "pop" : 1387, "state" : "MA", "_id
24 { "city" : "HOLYOKE", "loc" : [ -72.626193, 42.202007 ], "pop" : 43704, "state" : "MA", "_id" : "01040"
25 { "city" : "HUNTINGTON", "loc" : [ -72.873341, 42.265301 ], "pop" : 2084, "state" : "MA", "_id" : "0105
26 { "city" : "LEEDS", "loc" : [ -72.70340299999999, 42.354292 ], "pop" : 1350, "state" : "MA", "_id" : "0
27 { "city" : "LEVERETT", "loc" : [ -72.499334, 42.46823 ], "pop" : 1748, "state" : "MA", "_id" : "01054"
28 { "city" : "LUDLOW", "loc" : [ -72.471012, 42.172823 ], "pop" : 18820, "state" : "MA", "_id" : "01056"
29 { "city" : "MONSON", "loc" : [ -72.31963399999999, 42.101017 ], "pop" : 8194, "state" : "MA", "_id" : "01058" }
```

```
1 { "_id" : { "$oid" : "52cdef7c4bab8bd675297d8a" }, "name" : "Wetpaint", "permalink" : "abc2",
  "crunchbase_url" : "http://www.crunchbase.com/company/wetpaint", "homepage_url" : "http://wetpaint-
  inc.com", "blog_url" : "http://digitalquarters.net/", "blog_feed_url" :
  "http://digitalquarters.net/feed/", "twitter_username" : "BachelrWetpaint", "category_code" : "web",
  "number_of_employees" : 47, "founded_year" : 2005, "founded_month" : 10, "founded_day" : 17,
  "deadpooled_year" : 1, "tag_list" : "wiki, seattle, elowitz, media-industry, media-platform, social-
  distribution-system", "alias_list" : "", "email_address" : "info@wetpaint.com", "phone_number" :
  "206.859.6300", "description" : "Technology Platform Company", "created_at" : { "$date" : 1180075887000
  }, "updated_at" : "Sun Dec 08 07:15:44 UTC 2013", "overview" : "<p>Wetpaint is a technology platform
  company that uses its proprietary state-of-the-art technology and expertise in social media to build
  and monetize audiences for digital publishers. Wetpaint's own online property, Wetpaint Entertainment,
  an entertainment news site that attracts more than 12 million unique visitors monthly and has over 2
  million Facebook fans, is a proof point to the company's success in building and engaging audiences.
  Media companies can license Wetpaint's platform which includes a dynamic playbook tailored to their
  individual needs and comprehensive training. Founded by Internet pioneer Ben Elowitz, and with offices
  in New York and Seattle, Wetpaint is backed by Accel Partners, the investors behind Facebook.</p>",
  "image" : { "available_sizes" : [ [ [ 150, 75 ], "assets/images/resized/0000/3604/3604v14-max-
  150x150.jpg" ], [ [ 250, 125 ], "assets/images/resized/0000/3604/3604v14-max-250x250.jpg" ], [ [ 450,
  225 ], "assets/images/resized/0000/3604/3604v14-max-450x450.jpg" ] ] }, "products" : [ { "name" :
  "Wikison Wetpaint", "permalink" : "wetpaint-wiki" }, { "name" : "Wetpaint Social Distribution System",
  "permalink" : "wetpaint-social-distribution-system" } ], "relationships" : [ { "is_past" : false,
  "title" : "Co-Founder and VP, Social and Audience Development", "person" : { "first_name" : "Michael",
  "last_name" : "Howell", "permalink" : "michael-howell" } }, { "is_past" : false, "title" : "Co-
  Founder/CEO/Board of Directors", "person" : { "first_name" : "Ben", "last_name" : "Elowitz",
  "permalink" : "ben-elowitz" } }, { "is_past" : false, "title" : "COO/Board of Directors", "person" : {
  "first_name" : "Rob", "last_name" : "Grady", "permalink" : "rob-grady" } }, { "is_past" : false,
  "title" : "SVP, Strategy and Business Development", "person" : { "first_name" : "Chris", "last_name" :
  "Kollas", "permalink" : "chris-kollas" } }, { "is_past" : false, "title" : "Board", "person" : {
  "first_name" : "Theresia", "last_name" : "Ranzetta", "permalink" : "theresia-ranzetta" } }, { "is_past"
```

# JSON in Processing

Need to know whether top-level entity is an Array or Object.

- use `loadJSONArray(filename)` to get a `JSONArray` object.
- use `loadJSONObject(filename)` to get a `JSONObject` object.

To access Arrays, use `getJSONObject(index)` to get the object at the index.

Use `getInt(attr)`, `getFloat(attr)`, and `getString(attr)` on a `JSONObject` to get the value associated with that attr.

Similar functions `setInt()`, `setString()`, and `setFloat()` exist on both `JSONObject`s and `JSONArray`s to set value. See Processing reference for details.



# Hands-On: JSON

1. Grab a JSON dataset. If you do not have a better idea, you may check out <https://github.com/jdorfman/awesome-json-datasets>
2. Load the JSON into Processing using one of the load functions.
3. Visualize some part of the JSON to the screen. You do not necessarily have to make a plot or a map, but there should be some visual representation of the data.

# Index Cards!

1. Your name and EID.
2. One thing that you learned from class today. You are allowed to say "nothing" if you didn't learn anything.
3. One question you have about something covered in class today. You *may not* respond "nothing".
4. (Optional) Any other comments/questions/thoughts about today's class.