计算原理导论

Introduction to Computing Principles

天津大学　计算机科学与技术学院　刘志磊

Data

Data is a set of values of qualitative or quantitative variables.
-- Wikipedia

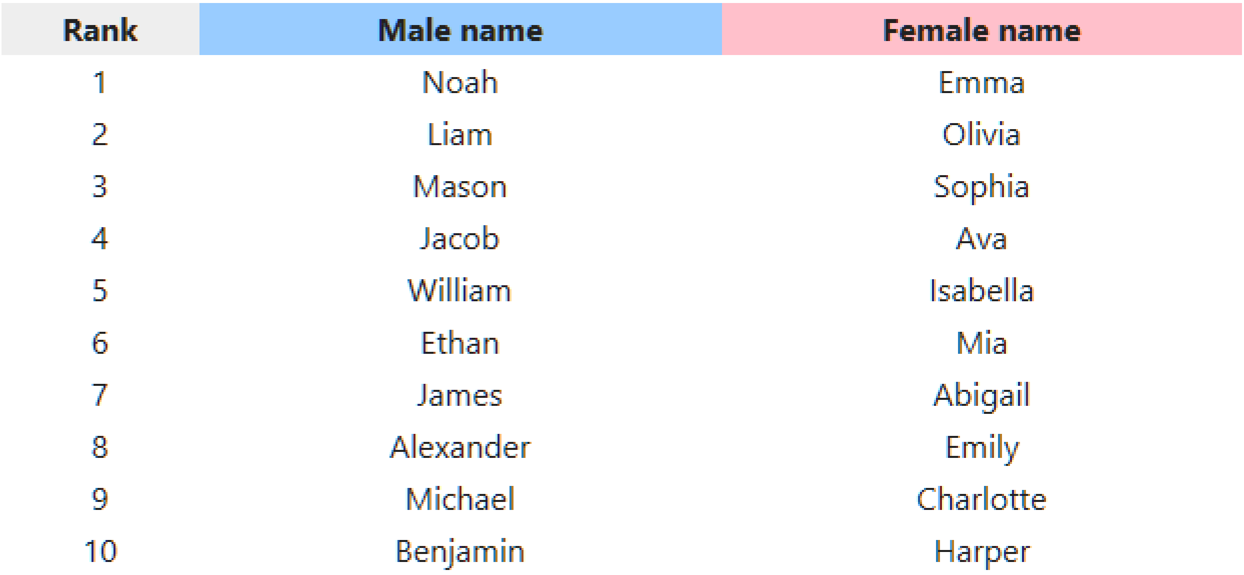
Structured data refers to any data that resides in a fixed field within a record or file. This includes data contained in relational databases and spreadsheets.

Table

A table is a means of arranging data in rows and columns. The use of tables is pervasive throughout all communication, research, and data analysis. A table consists of an ordered arrangement of rows and columns.

-- Wikipedia

Table Example - Social Security Baby Name



Rank	Male name	Female name
1	Noah	Emma
2	Liam	Olivia
3	Mason	Sophia
4	Jacob	Ava
5	William	Isabella
6	Ethan	Mia
7	James	Abigail
8	Alexander	Emily
9	Michael	Charlotte
10	Benjamin	Harper

On the Social Security Baby Name site:

- Names for babies born each year in the USA
- Top 1000 boy and girl names, 2000 names total

Table Example - Social Security Baby Name

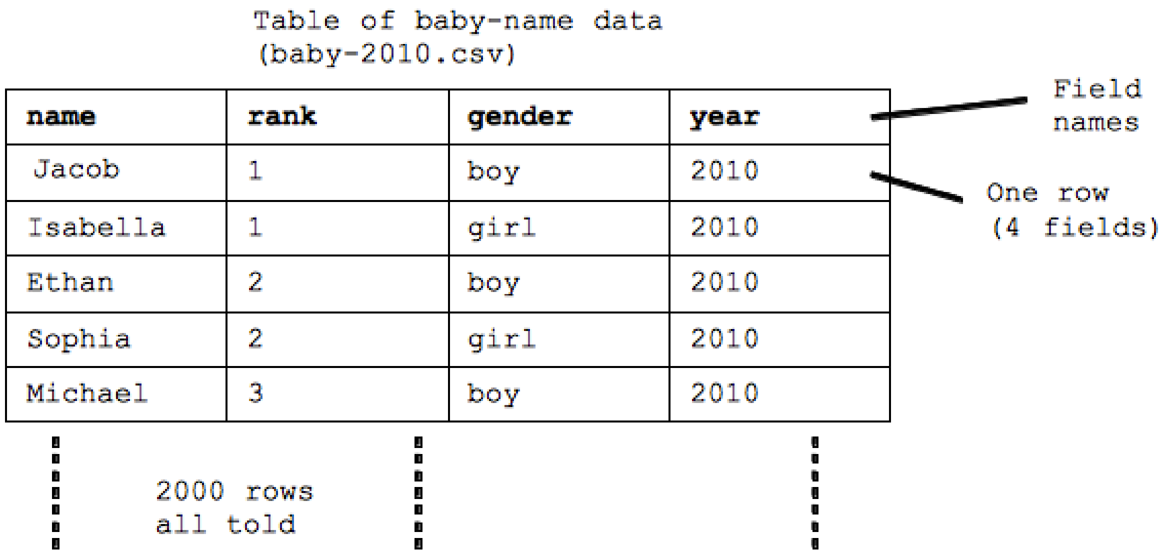


Table of baby-name data (baby-2010.csv)

name	rank	gender	year
Jacob	1	boy	2010
Isabella	1	girl	2010
Ethan	2	boy	2010
Sophia	2	girl	2010
Michael	3	boy	2010

Field names

One row (4 fields)

2000 rows all told

- Table terminology (术语):
  - **table** the whole rectangle of data
  - **row** data for one name
  - **field** individual items (columns) in a row
- Each field has a name: name, rank, gender, year

Tables Are Extremely Common

- Rectangular table format is very common
- Database: extension of this basic table idea
- Number of fields is small (categories)
- Number of rows can be millions or billions

Examples

- email inbox
  - one row is one message
  - fields: date, subject, from, ...
- craigslist
  - one row is one thing for sale
  - fields: description, price, seller, date, ...

Code and Practice - SimpleTable

- Baby data stored in "baby-2010.csv"

  - ".csv" stands for "comma separated values“

  - csv is a simple and widely used standard format to store a table as text in a file.

- For images: *for(pixel: image) { code }*
- For tables: *for (row: table) { code }*
- For print: *print(row)*

  - prints out the fields of a row on one line

For-loop

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
     print(row);
}
```

Example

```
Jacob, 1, boy, 2010
Isabella, 1, girl, 2010
Ethan, 2, boy, 2010
Sophia, 2, girl, 2010
Michael, 3, boy, 2010
Emma, 3, girl, 2010
Jayden, 4, boy, 2010
Olivia, 4, girl, 2010
William, 5, boy, 2010
Ava, 5, girl, 2010
Alexander, 6, boy, 2010
Emily, 6, girl, 2010
Noah, 7, boy, 2010
Abigail, 7, girl, 2010
Daniel, 8, boy, 2010
Madison, 8, girl, 2010
Aiden, 9, boy, 2010
Chloe, 9, girl, 2010
Anthony, 10, boy, 2010
Mia, 10, girl, 2010
Joshua, 11, boy, 2010
Addison, 11, girl, 2010
Mason, 12, boy, 2010
Elizabeth, 12, girl, 2010
Christopher, 13, boy, 2010
```

If-statement

- For select: **if-statement**

  - *if (condition) { code }*

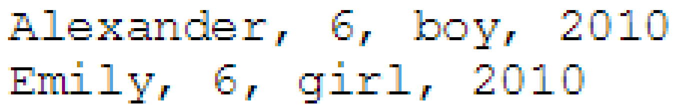
- Terminology: “query” (查询)

  - for: runs for every row (2000 rows)

  - if: picks out some

If-statement

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
    if (row.getField("rank") == 6) {
        print(row);
    }
}
```



```
Alexander, 6, boy, 2010
Emily, 6, girl, 2010
```

Example

Query Logic

- Field names for the baby table: name, rank, gender, year
- Pick field out of row: *getField*

  - *row.getField("field-name")*
- two equal signs and single equal sign

  - = : variable assignment

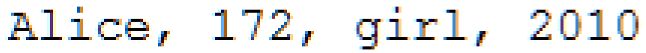
  - == : variable comparison

  - use == inside if-test
- Other comparisons

  < > <= >=

Query Logic

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
    if (row.getField("name") == "Alice") {
      print(row);
    }
}
```



```
Alice, 172, girl, 2010
```

Query Logic

- Baby table fields: name, rank, gender, year
- name field is "Robert", "Bob", "Abby", "Abigail" (try each in turn)
- rank field is 1
- rank field is < 10
- rank field is <= 10
- rank field is > 990
- gender field is "girl"
- rank field is less than 15
- gender field is "boy"

Query Logic

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name") == "Alice") {
    print(row);
  }
} // Change string to "Robert", "Bob", etc.
```

```
if (row.getField("gender") == "girl"){
 print(row);
}
if (row.getField("gender") == “boy"){
 print(row);
}
```

```
if (row.getField("rank") == 1){
 print(row);
}
if (row.getField("rank") < 10){
 print(row);
}
if (row.getField("rank") <= 10){
 print(row);
}
if (row.getField("rank") > 990){
 print(row);
}
if (row.getField("rank") < 15){
 print(row);
}
```

startsWith & endsWith

- == : very handy for baby names
- Test if the name field in a row starts with "Ab":

  - *if (row.getField("name").startsWith("Ab")) { ...*

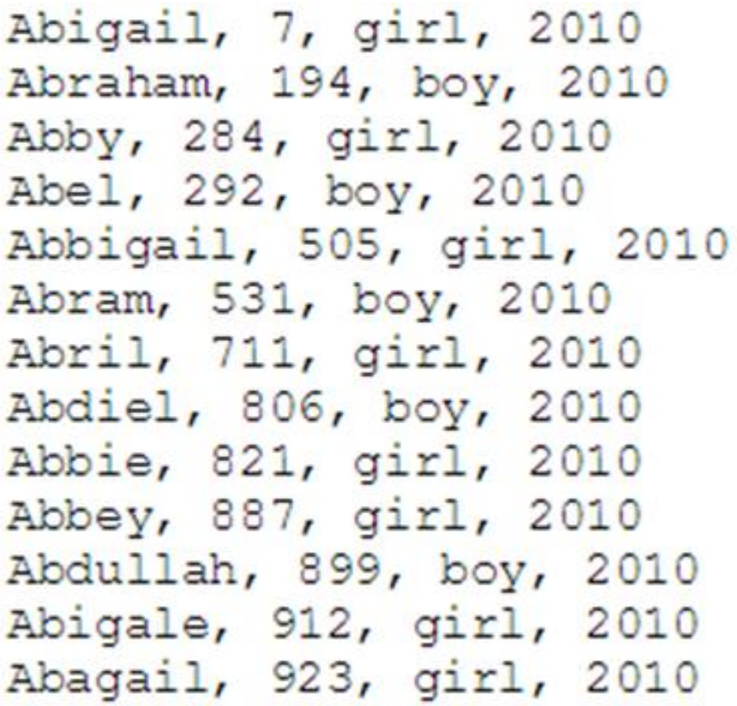
- Test if the name field in a row ends with "zy":

  - *if (row.getField("name").endsWith("zy")) { ...*

startsWith & endsWith

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name").startsWith("Ab")) {
    print(row);
  }
}
```

Example



```
Abigail, 7, girl, 2010
Abraham, 194, boy, 2010
Abby, 284, girl, 2010
Abel, 292, boy, 2010
Abbigail, 505, girl, 2010
Abram, 531, boy, 2010
Abril, 711, girl, 2010
Abdiel, 806, boy, 2010
Abbie, 821, girl, 2010
Abbey, 887, girl, 2010
Abdullah, 899, boy, 2010
Abigale, 912, girl, 2010
Abagail, 923, girl, 2010
```

startsWith & endsWith

name field starts with "Ab", "A", "a"(lower case)

name field starts with "Z", "Za" (each in turn)

name field ends with "z", "ly", "la" (each in turn)

```
if (row.getField("name").startsWith("Ab")) {
 print(row);
} // Change string to "A", "a", "Z", .. each in turn
```

```
if (row.getField("name").endsWith("z")) {
 print(row);
}// Change string to “z", “ly", “la", .. each in turn
```

Boolean Logic (布尔逻辑)

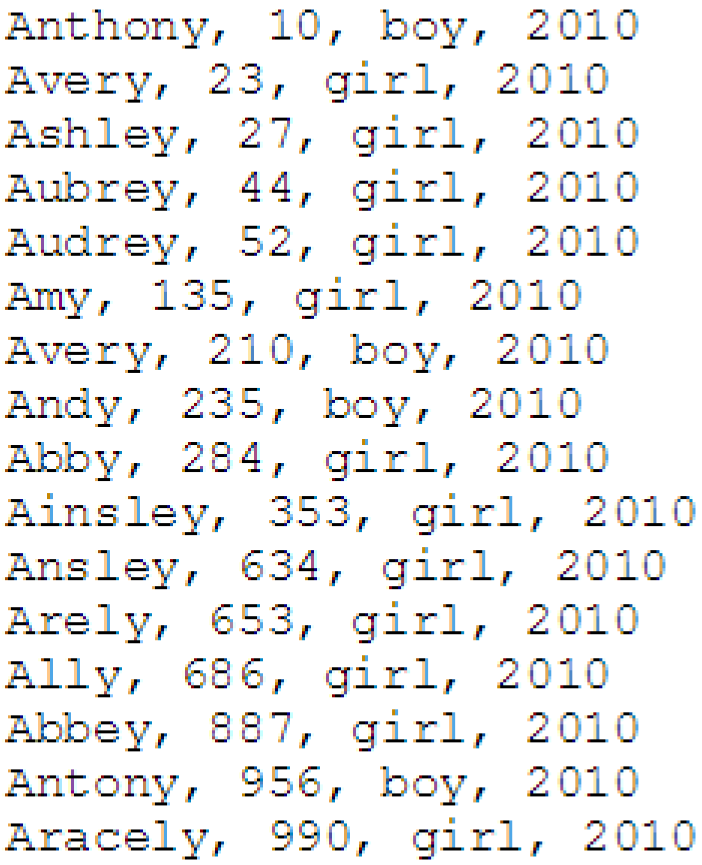
In mathematics and mathematical logic, boolean logic is the branch of algebra (代数) in which the values of the variables are the truth values true and false, usually denoted 1 and 0 respectively. Instead of elementary algebra where the values of the variables are numbers, and the main operations are addition and multiplication, the main operations of Boolean algebra are the conjunction and denoted as ∧, the disjunction or denoted as ∨, and the negation not denoted as ¬. It is thus a formalism for describing logical relations in the same way that ordinary algebra describes numeric relations.

Boolean Logic

- Boolean Logic: && || !
- In English, combine tests like this:
  - Name starts with "A" and ends with "y"
- In code, combine tests use "boolean logic“:
  - **and** is && (two ampersands)
  - **or** is || (two vertical bars)
  - **not** is ! (exclamation mark)

Boolean Logic

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name").startsWith("A") &&
      row.getField("name").endsWith("y")) {
    print(row);
  }
}
```



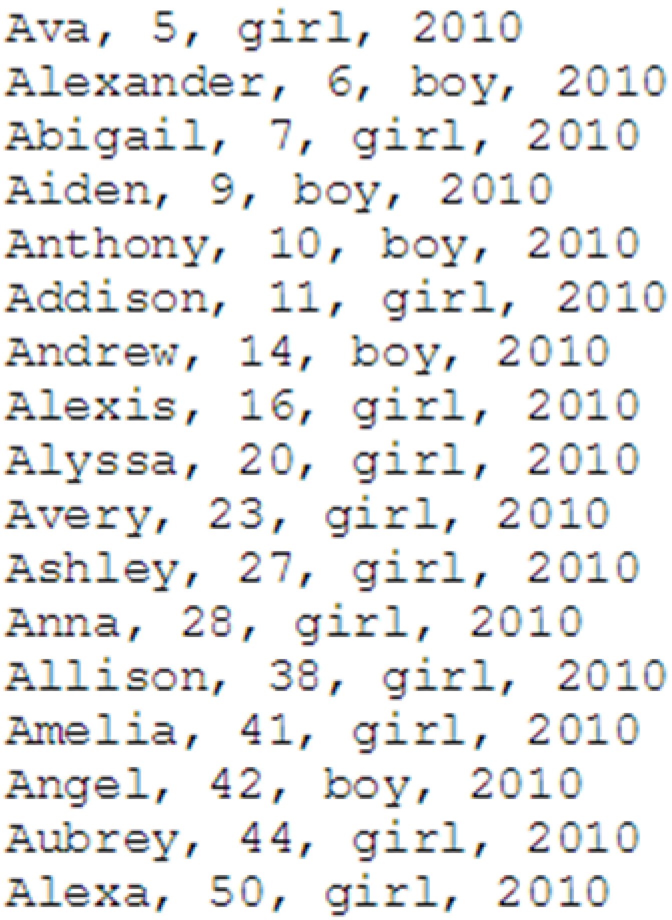
```
Anthony, 10, boy, 2010
Avery, 23, girl, 2010
Ashley, 27, girl, 2010
Aubrey, 44, girl, 2010
Audrey, 52, girl, 2010
Amy, 135, girl, 2010
Avery, 210, boy, 2010
Andy, 235, boy, 2010
Abby, 284, girl, 2010
Ainsley, 353, girl, 2010
Ansley, 634, girl, 2010
Arely, 653, girl, 2010
Ally, 686, girl, 2010
Abbey, 887, girl, 2010
Antony, 956, boy, 2010
Aracely, 990, girl, 2010
```

Boolean Logic - && ||

- The && joins a startsWith test and an endsWith test
- The whole test is written across two lines because it is kind of long (optional)
- **Standalone rule:**
  - the tests are syntactically complete tests on their own, then joined with && or ||
  - row.getField("name).startsWith("A") && endsWith("y") Incorrect, not Standalone
- **Common error**
  - too few right parenthesis around the test
  - Run tries to detect certain common errors
  - omitting the {, or typing & instead of && will give an error message

Boolean Logic - && ||

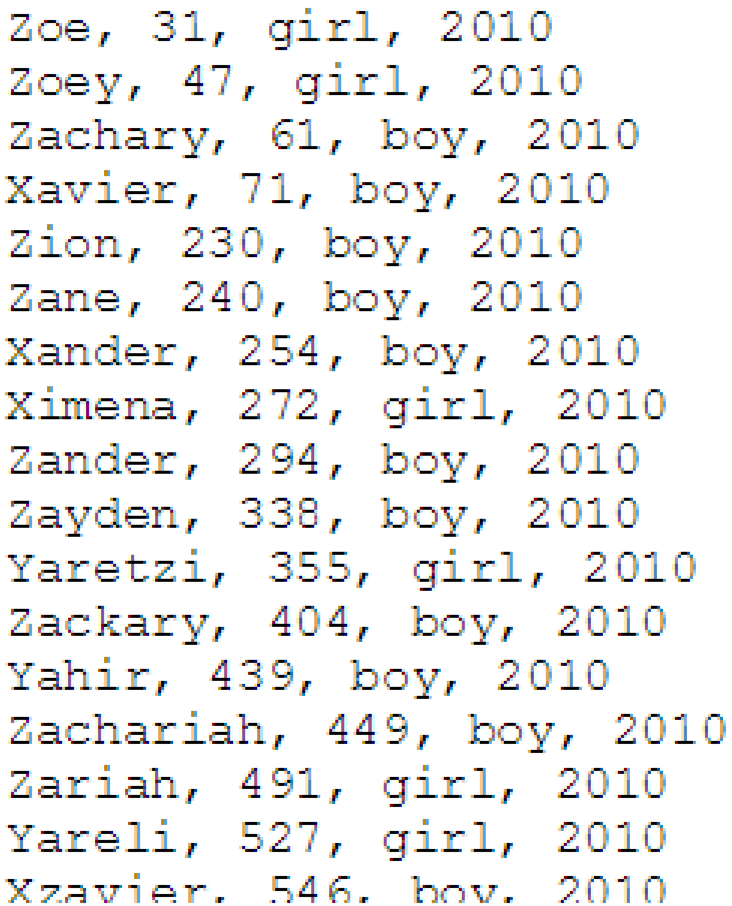
```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name").startsWith("A") &&
      row.getField("rank") <= 50) {
    print(row);
  }
}
```



```
Ava, 5, girl, 2010
Alexander, 6, boy, 2010
Abigail, 7, girl, 2010
Aiden, 9, boy, 2010
Anthony, 10, boy, 2010
Addison, 11, girl, 2010
Andrew, 14, boy, 2010
Alexis, 16, girl, 2010
Alyssa, 20, girl, 2010
Avery, 23, girl, 2010
Ashley, 27, girl, 2010
Anna, 28, girl, 2010
Allison, 38, girl, 2010
Amelia, 41, girl, 2010
Angel, 42, boy, 2010
Aubrey, 44, girl, 2010
Alexa, 50, girl, 2010
```

Boolean Logic - && ||

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  // your code here
  if (row.getField("name").startsWith("X") ||
      row.getField("name").startsWith("Y") ||
      row.getField("name").startsWith("Z")) {
    print(row);
  }
}
```



```
Zoe, 31, girl, 2010
Zoey, 47, girl, 2010
Zachary, 61, boy, 2010
Xavier, 71, boy, 2010
Zion, 230, boy, 2010
Zane, 240, boy, 2010
Xander, 254, boy, 2010
Ximena, 272, girl, 2010
Zander, 294, boy, 2010
Zayden, 338, boy, 2010
Yaretzi, 355, girl, 2010
Zackary, 404, boy, 2010
Yahir, 439, boy, 2010
Zachariah, 449, boy, 2010
Zariah, 491, girl, 2010
Yareli, 527, girl, 2010
Xzavier, 546, boy, 2010
```

Boolean Logic - && ||

- name starts with "Ab" or name starts with "Ac"
- name starts with "Ab" or name starts with "Ac" or name starts with "Al"
- name starts with "O" and name ends with "a"
- name starts with "O" and gender is "girl"
- name ends with "a" and gender is "boy"
- rank is <= 10 and gender is "girl" ("top 10 girl names")
- rank is <= 10 or gender is "girl"
- name ends with "ia" and gender is "boy" (also try with gender is "girl")
- name ends with "io" and gender is "girl" (then try "boy")
- name ends with "o" and gender is boy and rank is >= 900

Boolean Logic - && ||

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name").startsWith("Ab") ||
      row.getField("name").startsWith("Ac")) {

}
```

```
if (row.getField("name").startsWith("Ab") ||
    row.getField("name").startsWith("Ac") ||
    row.getField("name").startsWith("Al")) {

}
```

```
if (row.getField("name").startsWith(“O") ||
    row.getField("name").endsWith("A")) {

}
```

```
if (row.getField("name").startsWith(“O") ||
    row.getField(“gender") == “girl”) {
 print(row);
}
```

```
if (row.getField(“rank") <= 10 &&
    row.getField(“gender") == “girl”) {

}
```

```
if (row.getField("name").endsWith("a") &&
    row.getField("gender") == "boy") {

}
```

```
if (row.getField(“rank") <= 10 ||
    row.getField(“gender") == “girl”) {

}
```

```
if (row.getField("name").endsWith("ia") &&
    row.getField("gender") == "boy") {

}
```

```
if (row.getField("name").endsWith("io") &&
    row.getField("gender") == "girl") {

}
```

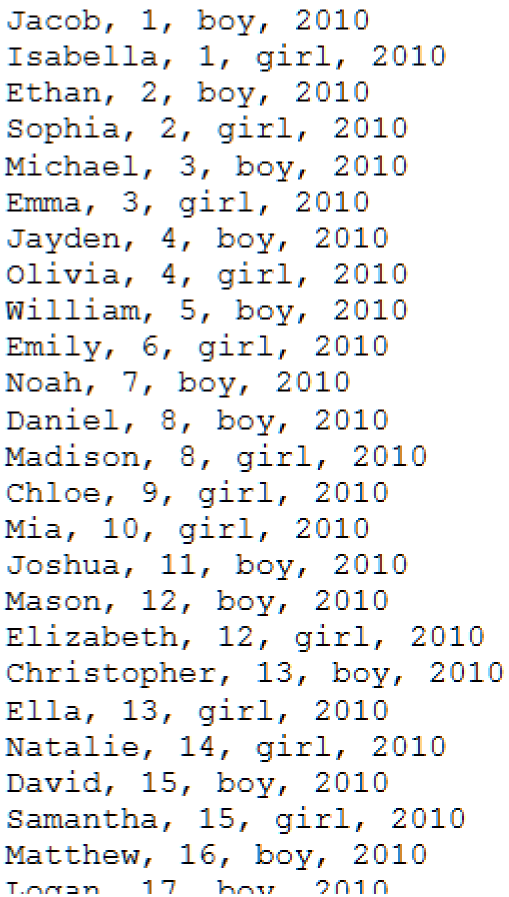
```
if (row.getField("name").endsWith("o") &&
    row.getField("gender") == "boy" &&
    row.getField("rank") >= 900) {
  print(row);
}
```

Boolean Logic - !

- “not” operation: !
- The boolean "not" operation inverts true and false
- Two forms of not:
- ! (exclamation mark) can go in front of an s.startsWith() s.endsWith() expression
  - !row.getField("name").startsWith("A")
  - names not starting with "A“, starting with any letter other than "A"
- ! = variant of = =, meaning "not equal to“
  - row.getField("name") != "Alice“
  - names which are not equal to "Alice“, any name other than "Alice“

Boolean Logic - !

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (!row.getField("name").startsWith("A")) {
    print(row);
  }
}
```



```
Jacob, 1, boy, 2010
Isabella, 1, girl, 2010
Ethan, 2, boy, 2010
Sophia, 2, girl, 2010
Michael, 3, boy, 2010
Emma, 3, girl, 2010
Jayden, 4, boy, 2010
Olivia, 4, girl, 2010
William, 5, boy, 2010
Emily, 6, girl, 2010
Noah, 7, boy, 2010
Daniel, 8, boy, 2010
Madison, 8, girl, 2010
Chloe, 9, girl, 2010
Mia, 10, girl, 2010
Joshua, 11, boy, 2010
Mason, 12, boy, 2010
Elizabeth, 12, girl, 2010
Christopher, 13, boy, 2010
Ella, 13, girl, 2010
Natalie, 14, girl, 2010
David, 15, boy, 2010
Samantha, 15, girl, 2010
Matthew, 16, boy, 2010
Logan, 17, boy, 2010
```

Boolean Logic - !

- girl names starting with "A" (no "nots" in this one)
- girl names not starting with "A"
- names starting with "A" and not ending with "y"
- names starting with "A" and ending with "y" and not equal to "Abbey"

Boolean Logic - !

```
table = new SimpleTable("baby-2010.csv");
for (row: table) {
  if (row.getField("name").startsWith("A") &&
      row.getField("gender") == "girl") {
     print(row);
  }
}
```

```
if (row.getField("name").startsWith("A") &&
    !row.getField("name").endsWith("y")) {
   print(row);
}
```

```
if (!row.getField("name").startsWith("A") &&
    row.getField("gender") == "girl") {
   print(row);
}
```

```
if (row.getField("name").startsWith("A") &&
    row.getField("name").endsWith("y") &&
    row.getField("name") != "Abbey") {
  print(row);
}
```

Inside/Outside Loop

- Loop: power technique to do something a zillion times
- Inside vs. outside the loop is a huge difference.

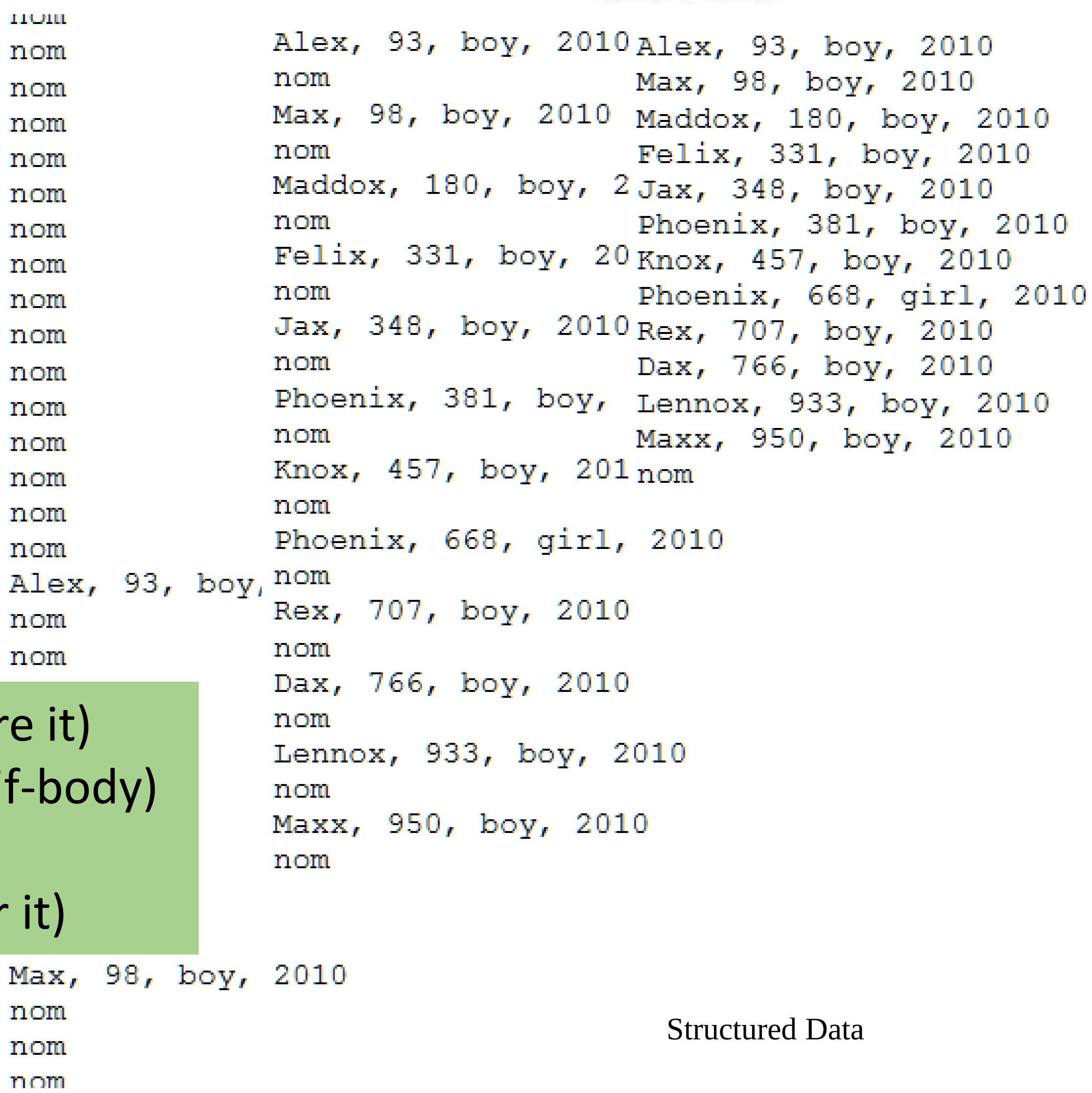
Experiment:

- 2000 names (rows) total, 12 names end with "x"
- We have the line: print("nom"); and we'll move it around

Inside/Outside Loop

```
table = new SimpleTable("baby-2010.csv");
// Location 1
for (row: table) {
  print("nom");
  // Location 2
  if (row.getField("name").endsWith("x")) {
    print(row);
    // Location 3
  }
}
// Location 4
```

Location 1: 1 (outside the loop, before it)
Location 2: 2000 (in the loop, not in the if-body)
Location 3: 12 (in the if-body)
Location 4: 1 (outside the loop, after it)



```
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
nom
Alex, 93, boy,
nom
nom
Max, 98, boy, 2010
nom
nom
nom
```

```
Alex, 93, boy, 2010
nom
Max, 98, boy, 2010
nom
Maddox, 180, boy, 2
nom
Felix, 331, boy, 20
nom
Jax, 348, boy, 2010
nom
Phoenix, 381, boy,
nom
Knox, 457, boy, 201
nom
Phoenix, 668, girl, 2010
nom
Rex, 707, boy, 2010
nom
Dax, 766, boy, 2010
nom
Lennox, 933, boy, 2010
nom
Maxx, 950, boy, 2010
nom
```

```
Alex, 93, boy, 2010
Max, 98, boy, 2010
Maddox, 180, boy, 2010
Felix, 331, boy, 2010
Jax, 348, boy, 2010
Phoenix, 381, boy, 2010
Knox, 457, boy, 2010
Phoenix, 668, girl, 2010
Rex, 707, boy, 2010
Dax, 766, boy, 2010
Lennox, 933, boy, 2010
Maxx, 950, boy, 2010
nom
```

Count

How to Count: three things to do counting

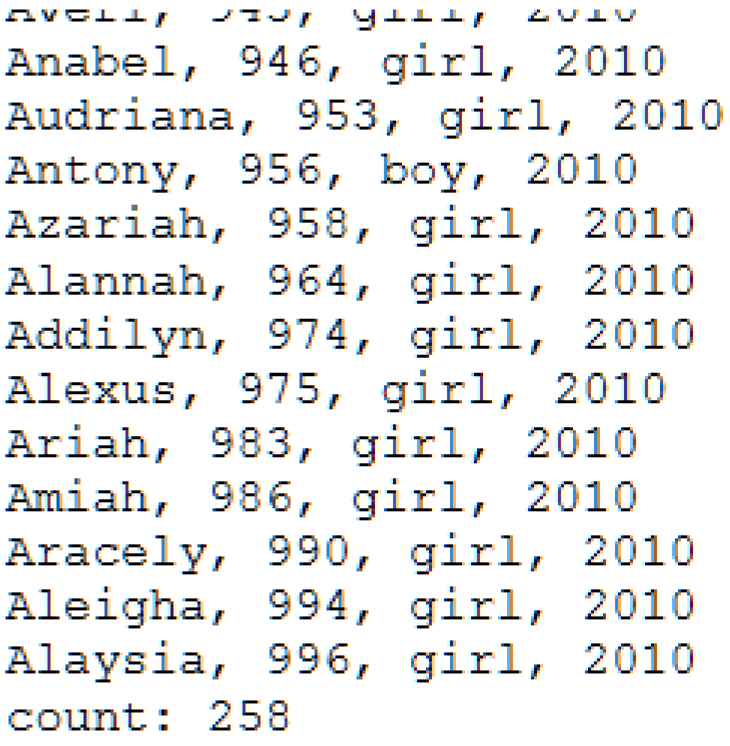
1. Create a count variable and set it to 0 before the loop
   - count = 0;
2. Inside the if-statement, increase count by 1
   - count = count + 1;
3. Print the final value stored in count after the loop
   - print("count:", count);

Pattern: init, increment, print

x = x + 1; increments the value stored in a variable

Count

```
table = new SimpleTable("baby-2010.csv");
count = 0;
for (row: table) {
  if (row.getField("name").startsWith("A")) {
    print(row);  // Could comment this line out
    count = count + 1;  // increases the value
  }
}
print("count:", count);
```



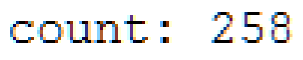
```
Averil, 945, girl, 2010
Anabel, 946, girl, 2010
Audriana, 953, girl, 2010
Antony, 956, boy, 2010
Azariah, 958, girl, 2010
Alannah, 964, girl, 2010
Addilyn, 974, girl, 2010
Alexus, 975, girl, 2010
Ariah, 983, girl, 2010
Amiah, 986, girl, 2010
Aracely, 990, girl, 2010
Aleigha, 994, girl, 2010
Alaysia, 996, girl, 2010
count: 258
```

Count

- Try commenting out or removing the print(row); line inside the { .. } then-code. What is the output now?
- How many names start with "X"? Then change to count starting with "Y"?
- How many girl names begin with "A"? Then change to count how many boy names begin with "A"?

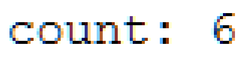
Count

```
table = new SimpleTable("baby-2010.csv");
count = 0;
for (row: table) {
  if (row.getField("name").startsWith("A")) {
      count = count + 1;
      // increases the value in count by 1
  }
}
print("count:", count);
```



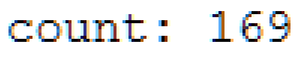
```
count: 258
```

```
if (row.getField("name").startsWith("X")) {
 count = count + 1;
}
```



```
count: 6
```

```
if (row.getField("name").startsWith("A") &&
    row.getField("gender") == "girl") {
  count = count + 1;
}
```



```
count: 169
```

Count

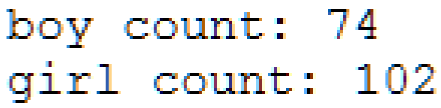
How to Count Multiple Things: Counting multiple things in the loop

1. Have multiple counters:
    - count1 = 0; // boy counter
    - count2 = 0; // girl counter
2. Series of if-statements inside the loop
    - count1 = count1 + 1;
    - count2 = count2 + 1;
    - if-statements are not nested (more complex)
3. After the loop, print both counters
    - print(“count1”, count1);
    - print(“count2”, count2);

More mnemonic variable names, like countBoy and countGirl

Count

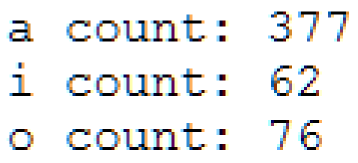
```
table = new SimpleTable("baby-2010.csv");
count1 = 0;  // boy counter
count2 = 0;  // girl counter
for (row: table) {
  if (row.getField("name").endsWith("y") &&
      row.getField("gender") == "boy") {
    count1 = count1 + 1;
  }
  if (row.getField("name").endsWith("y") &&
      row.getField("gender") == "girl") {
    count2 = count2 + 1;
  }
}
print("boy count:", count1);
print("girl count:", count2);
```



```
boy count: 74
girl count: 102
```

Count

```
table = new SimpleTable("baby-2010.csv");
count1 = 0;
count2 = 0;
count3 = 0;
for (row: table) {
  if (row.getField("name").endsWith("a")) {
    count1 = count1 + 1;
  }
  if (row.getField("name").endsWith("i")) {
    count2 = count2 + 1;
  }
  if (row.getField("name").endsWith("o")) {
    count3 = count3 + 1;
  }
}
print("a count:", count1);
print("i count:", count2);
print("o count:", count3);
```



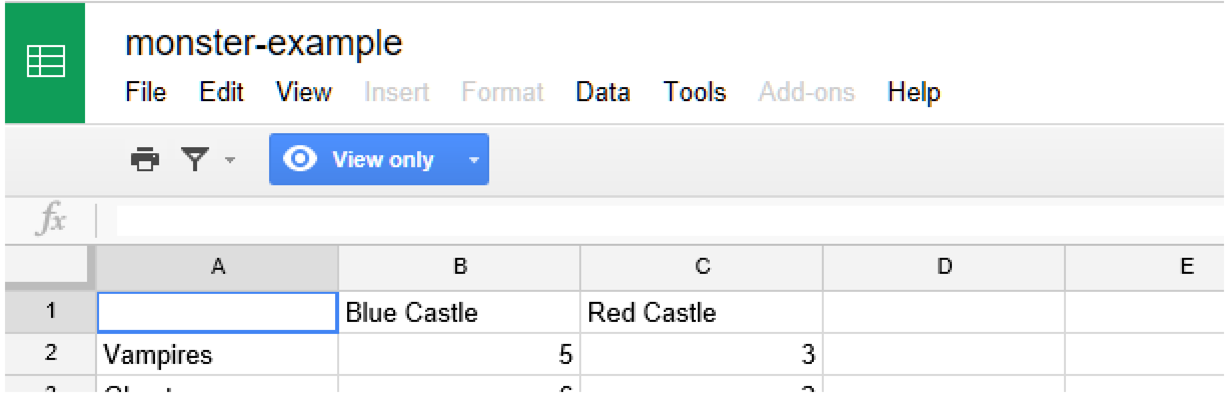
```
a count: 377
i count: 62
o count: 76
```

Spreadsheet

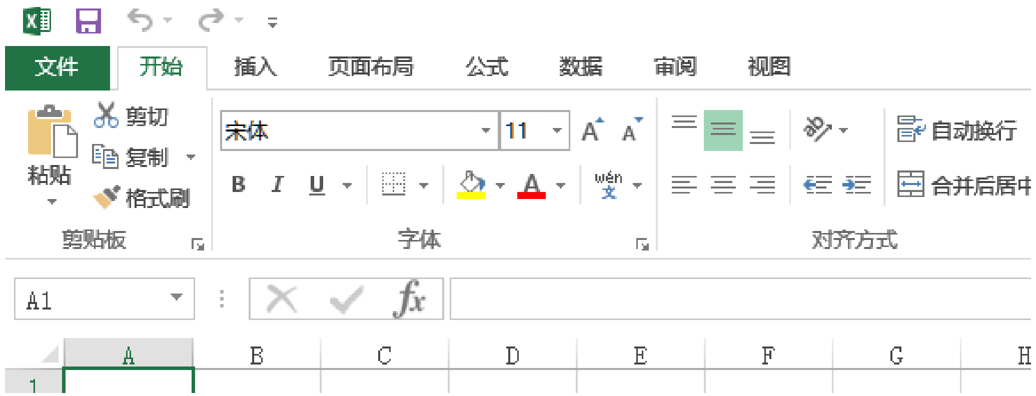
A spreadsheet is an interactive computer application for organization, analysis and storage of data intabular form. Spreadsheets are developed as computerized simulations of paper accounting worksheets. The program operates on data entered in cells of a table. Each cell may contain either numeric or text data, or the results of formulas that automatically calculate and display a value based on the contents of other cells. A spreadsheet or worksheet may also refer to one such electronic document.

Spreadsheet

- A “spreadsheet” is an easy way to do simple computations
- Everyone should be able to make a basic spreadsheet
- Numbers and formulas on paper is a paradigm
- Numbers and variables in computer code is a paradigm, like x = x + 1
- Spreadsheet paradigm is rows and columns of numbers (visual)
- Spreadsheets enable math without programming, a great invention
- History: Visicalc, then Lotus, then Excel
- Spreadsheets energized the “personal computer” revolution



	A	B	C	D	E
1		Blue Castle	Red Castle		
2	Vampires	5	3		



Spreadsheet

Who Makes Great Software

- The creators of the spreadsheet knew finance math and paper spreadsheets, and they had some computer knowledge
- Theory: knowing the problem domain creates great software more than knowing CS
  - What problem to solve
  - How users look at the problem
  - The user's priorities
- Hidden agenda: everyone should know a little CS

Spreadsheet - Cells and Naming

- A spreadsheet is a rectangle of individual cells
- Each cell can contain number, date, text, .. whatever
- Addressing: columns are named: A, B, C, D, ...
- Addressing: rows are numbered: 1, 2, 3, 4, 5, ...
- So one cell can be identified like: B3, C12, A1, ..

Spreadsheet - sum( )

How to add a range of cells

- Compute the total number of monsters in the blue castle
- Click on the B8 cell, a couple rows below the last blue castle number
- Type in the following "formula" (with the equal sign): =sum(B1:B6)
- The equal sign = at the start means this cell is computed from other cells
- The sum() adds up all the numbers in a range of cells
- The B1:B6 means the whole vertical group of cells from B1 down through B6 (lowercase letters like b1:b6 work too)
- Type in "Total Count" in the cell to the left (A8) to serve as a label
- Famous Reinhart/Rogoff bug - wrong cells in formula

Spreadsheet - sum( )

- When you change a number up above, the sum is automatically updated
- Once you type in the =**sum(...)** in the cell, it is replaced with the computed sum number (28 in this case)
- Click the cell, edit up above
- Double click the cell to edit
- Color shows cell-dependency
- Type in "b1:b6" vs. click-drag
- Hit the esc key to cancel out of editing, a life saver
- Using =**sum()** to add up a bunch of numbers is super common

Spreadsheet - + - * /

- Suppose every monster pays $100 per night and we want to compute the $ income per night, i.e. **count*100**
- We can write an arithmetic formula like **=B1*B2** in a cell to compute a number based on the values of other cells
- Click the B9 cell just below the sum
- Type in the formula (with the equal sign): **=B8*100**
- Probably the easiest way to edit an existing formula such as in B8 and B9 is double clicking the cell
- Trick: while typing in the formula, instead of typing "B8", just click the cell you mean
- Type in "Total $/night" as a label to the left
- This is similar to the earlier **sum()** computation, but with basic **+ - * /** type arithmetic

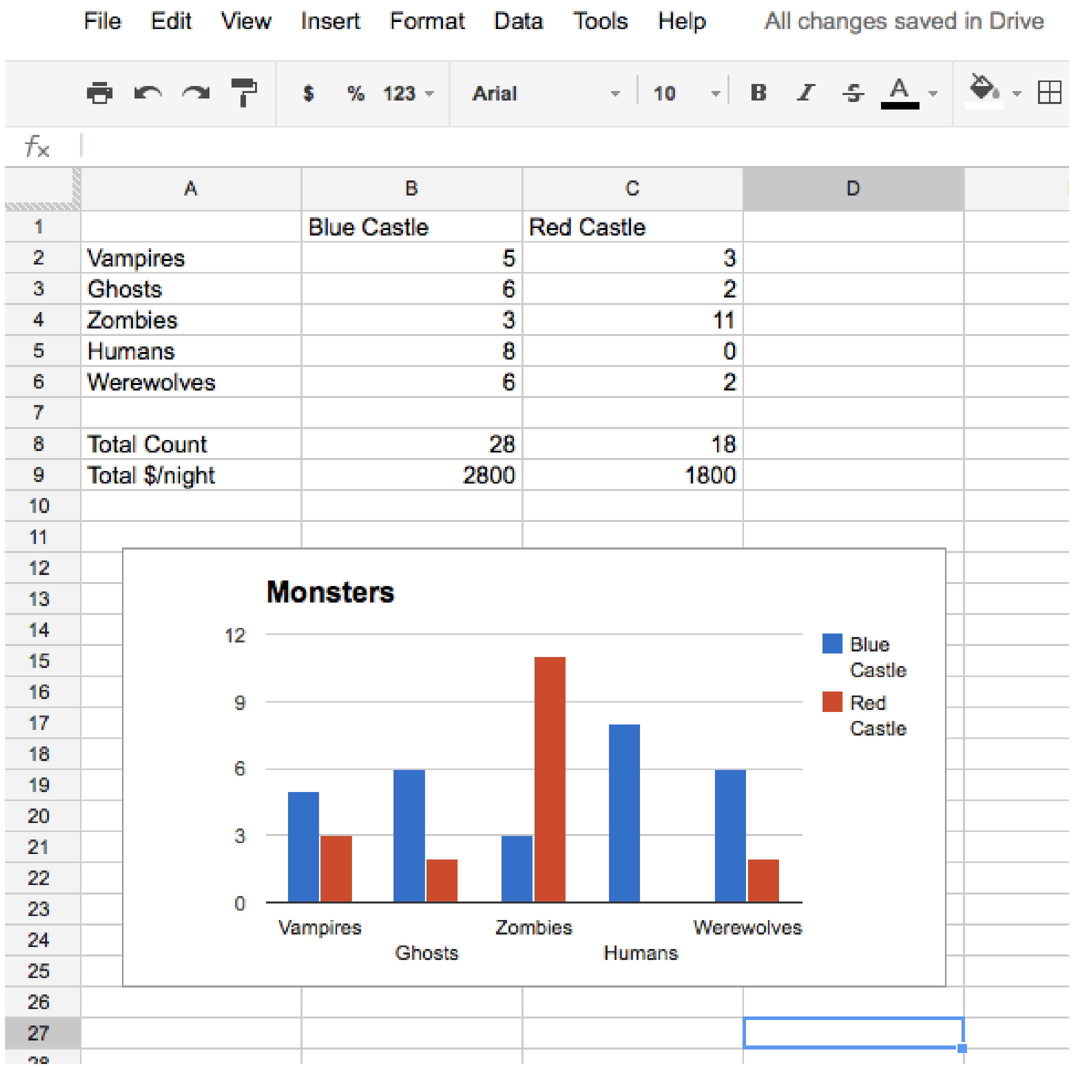
Spreadsheet - average( )

- Above **sum(a1:a10)** computes the total sum of range of of numbers
- Similarly, **average(a1:a10)** compute the average of range of numbers
- **sum()** and **average()** are probably the two most commonly used functions

Spreadsheet - Chart

- Click on A1 (the upper left of the data) and drag down to the lower right of the data (C6)
- Don't include the totals, just the raw numbers at the top
- Select Insert Chart
- There are many types of chart available
- Experiment with bar vs. line chart, or maybe add a title, resize it a bit
- Position the chart below all the numbers
- Notice: changing a number updates the chart
- Making pretty charts with your data is pretty easy

Spreadsheet - Chart



	A	B	C	D
1		Blue Castle	Red Castle	
2	Vampires	5	3	
3	Ghosts	6	2	
4	Zombies	3	11	
5	Humans	8	0	
6	Werewolves	6	2	
7				
8	Total Count	28	18	
9	Total $/night	2800	1800	

Thank You!

Q&A