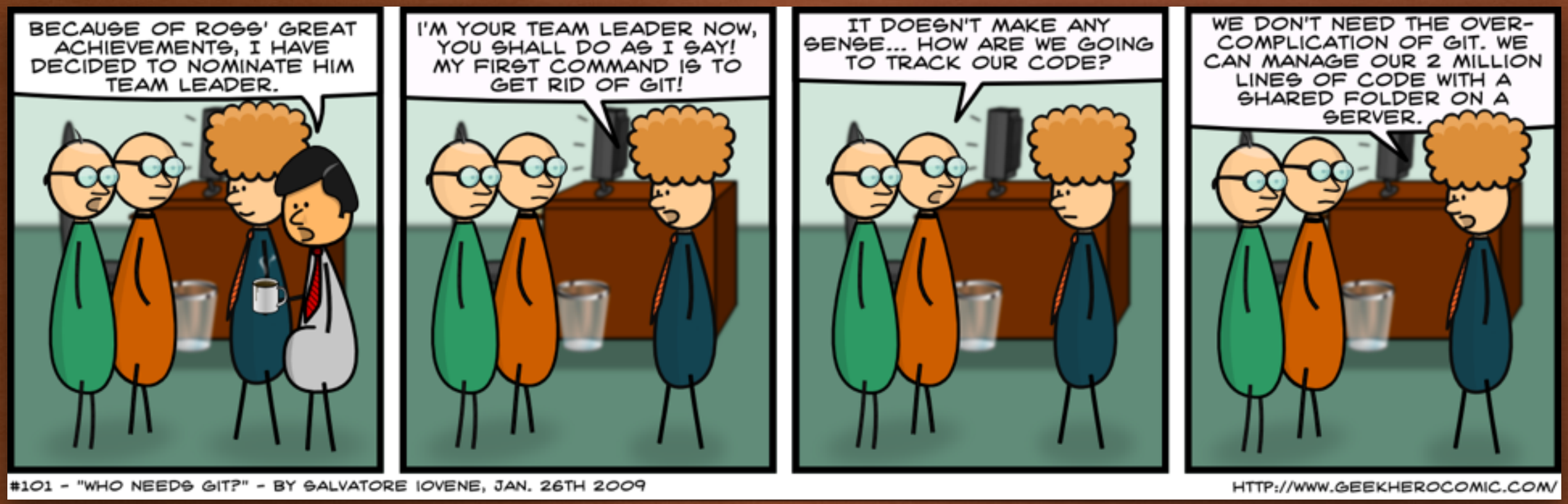
CLASS 2 - 5 NOV 2015

# INTRO TO RUBY & MORE GIT

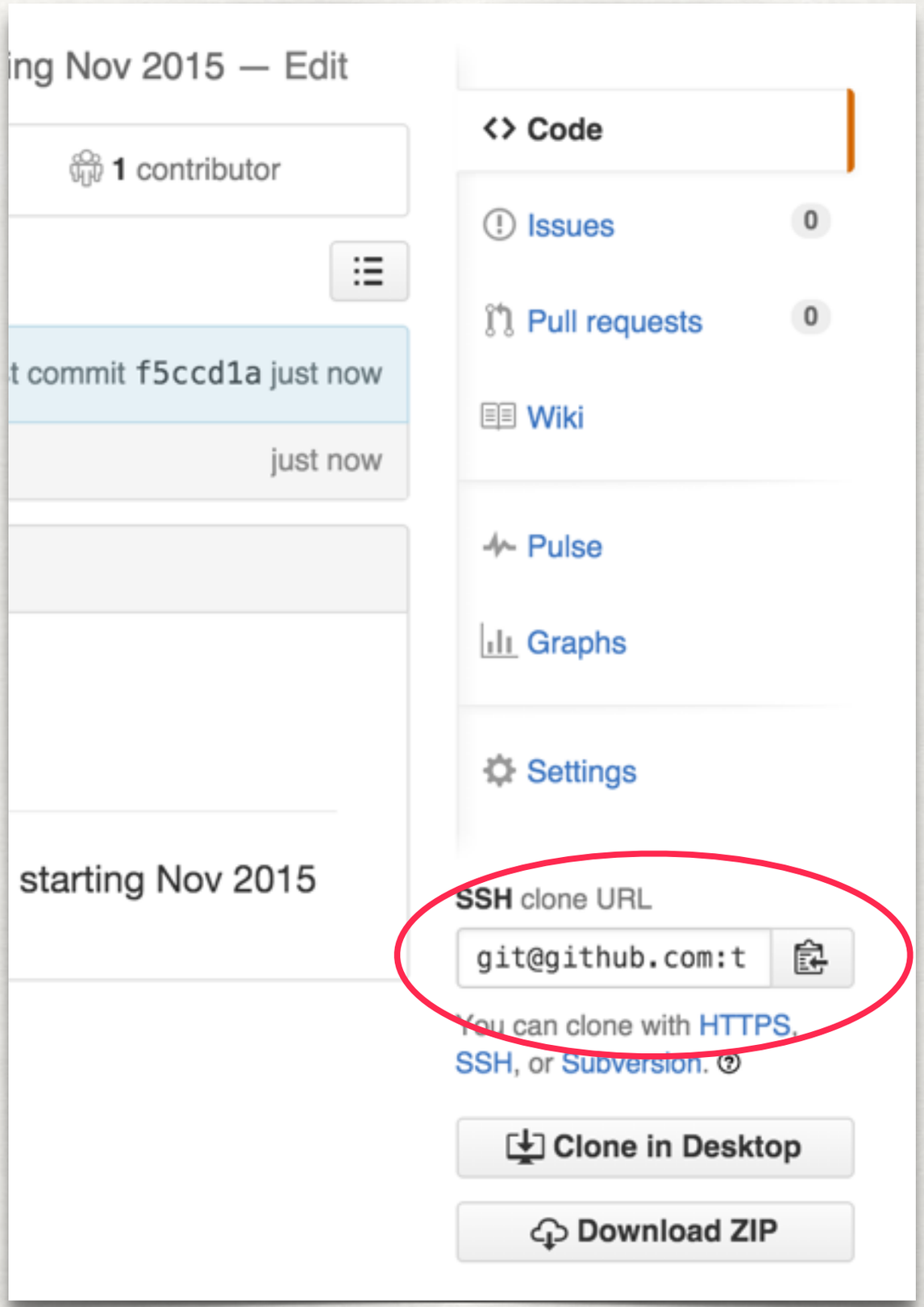


# TODAY'S OBJECTIVES

- Getting used to working with git and Github to collaborate and submit homework assignments.
- Intro to Ruby!
- Variables
- Strings
- Getting user input in command line applications.
- Conditionals

# GIT CLONE

- This is is how you “download” a git repo with all of its history to your local machine
- If you are added as a contributor on a project, you can push your changes to the repo.
- This is how coders collaborate.
- `git clone git@github.com:USER/REPO.git`



# PAIR UP!

## 15 MINUTES

- First navigate to your ‘lexicon’ folder from the other night.
- Edit your homework so one of the answers is wrong. Push up those changes.
- Find someone near you to work with and add each other as collaborators on the git repo.
- Navigate somewhere else and git clone your partner’s repo. Correct his or her error and push it up to github.

# HOW WE WILL SUBMIT HOMEWORK FROM NOW ON

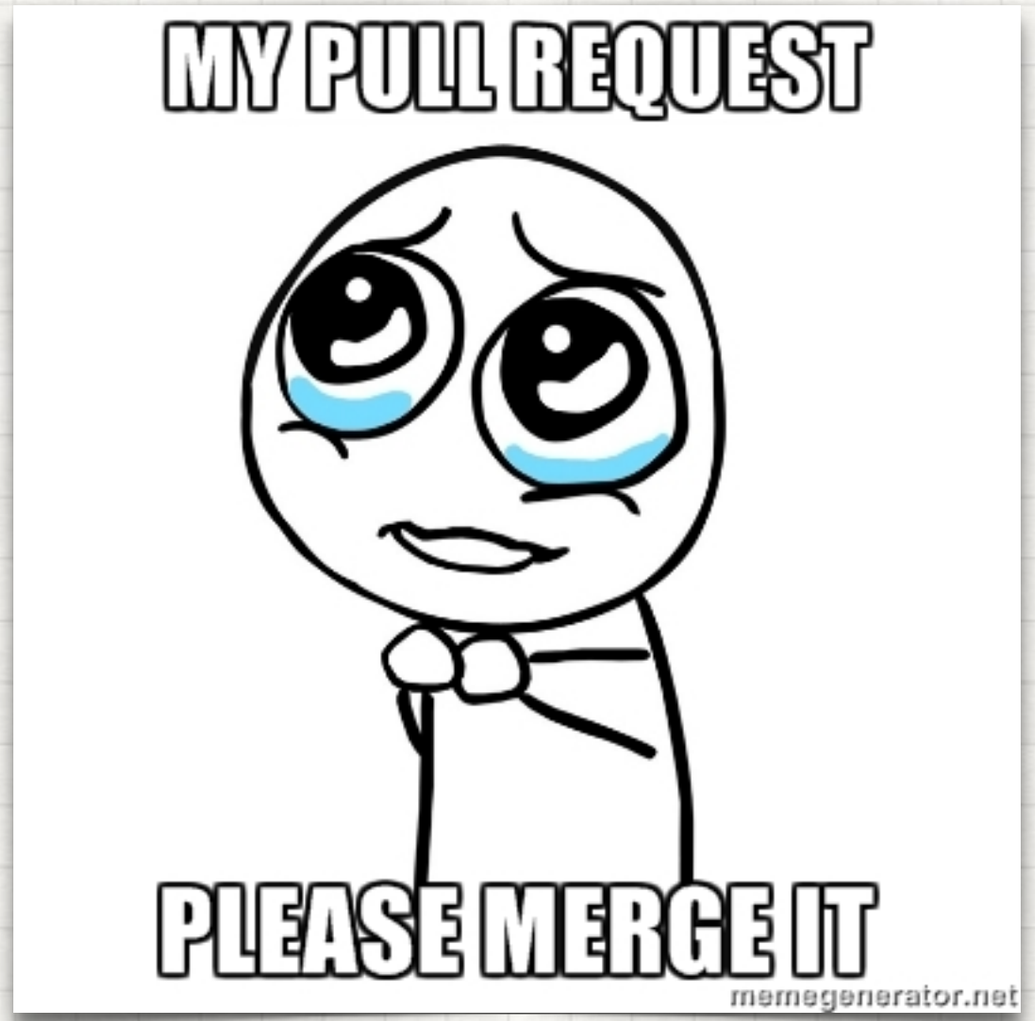
# DONEC QUIS NUNC

- Fork my homework repo, then clone your fork.
- `git remote -v`
- `git remote add upstream git@github.com:trivett/BEWD-NYC.git`
- I'll add some stuff now
- Now `git pull upstream master`
- Now open it in sublime!

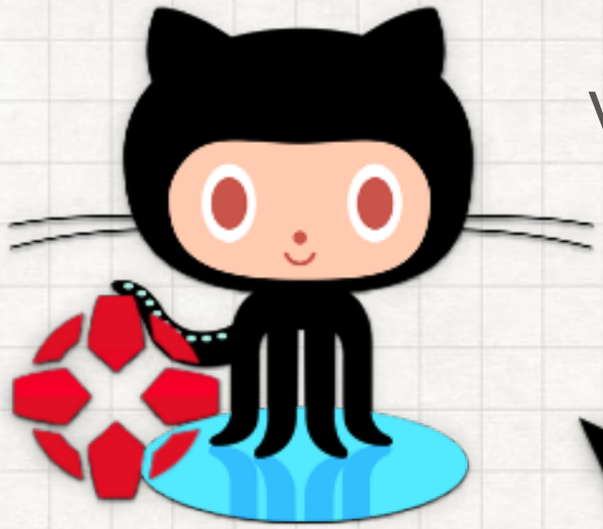
# ONE MORE TIME!

# PULL REQUESTS

- Part of Github, not git.
- After you push things up to your fork of the project (in this case, the class repo), click the pull request button.
- Press the pull request button on github.
- We review your submission
- MAKE SURE YOU DON'T COMMIT TO SOMEONE ELSE'S FOLDER!!!
- Add a message if need be.



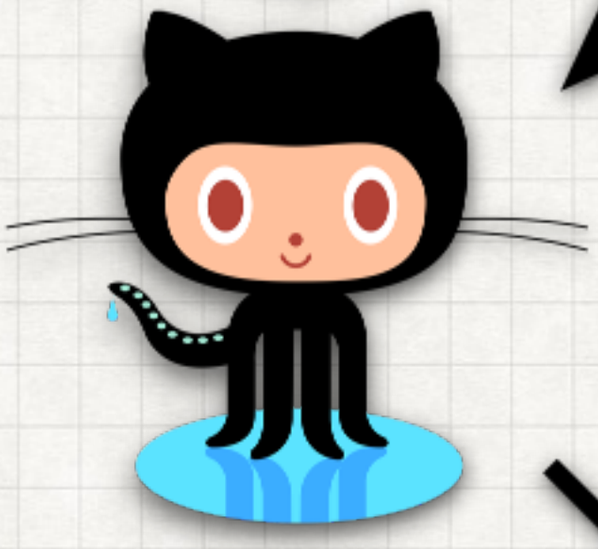
upstream



Vincent's repo

Your fork

origin



pull request

local

# DRESS REHEARSAL FOR HOMEWORK
## (AND CONTRIBUTING TO OPEN SOURCE SOMEDAY)

- Pull from upstream
- Navigate to your folder with your name on it.
- Create a new a new directory inside that called ‘class2’
- Inside that folder, create a file called ‘joke.txt’
- Write a joke in that file, use git to push it up to `origin` to push it to your fork.
- Then make a pull request and I will approve your contribution if the joke is funny.

# BREAK FOR 5

# I BET YOU ARE READY FOR RUBY

# Ruby first.

- It will be easier to navigate a Rails project once we have a basic understanding of Ruby.
- We will first teach you how to write simple Ruby programs as standalone applications.
- Once we have become familiarized with Ruby, we will start building Rails applications (which are groups of Ruby files that work together)

# Programming Fundamentals

In order to start writing our own Ruby programs, we need to learn some of the basic fundamental tools

Specifically, we need to learn:

- Variables
- Methods
- Conditions

We will first learn the basics on their own, and then try to apply our skills in a simple interactive Ruby script

# MAKE SURE YOU HAVE THE RIGHT VERSION

Try `ruby -v` in the terminal. You should get `'2.2.3'`

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
install/master/install)”

brew doctor

brew update
```

```
brew install rbenv ruby-build

# Add rbenv to bash so that it loads every time you open a terminal
echo 'if which rbenv > /dev/null; then eval "$(rbenv init -)"; fi'
>> ~/.bash_profile
source ~/.bash_profile

# Install Ruby
rbenv install 2.2.3
rbenv global 2.2.3
ruby -v
```

# WHAT IS RUBY?

- Open source programming language started by Matsumoto Yukihiro (a.k.a Matz) in 1995.
- Designed to be easy and fun to use with English-like syntax.
- Interpreted, object oriented (we'll discuss this later).
- Takes care of the more boring parts of programming.
- High level - It is itself an *abstraction* of C, which makes you handle boring stuff like reserving and relinquishing memory.

“

Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run fast. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves.

*— Matsumoto Yukihiro a.k.a. “Matz”*

”

# COMMUNITY

- MINASWAN
- Nice supportive community
- This matters a lot! There are programmers all over the world releasing tools to solve problems with Ruby. Ruby on Rails is one of those open source tools.
- If there is something you need done, there is most likely a great Ruby gem for it.
- Plenty of meetups in NYC, very welcoming to beginners.

# HELLO WORLD IN C, ASSEMBLY, AND RUBY

```
section     .text
global      _start                          ;must be declared for linker (ld)

_start:                                     ;tell linker entry point

    mov     edx,len                         ;message length
    mov     ecx,msg                         ;message to write
    mov     ebx,1                           ;file descriptor (stdout)
    mov     eax,4                           ;system call number (sys_write)
    int     0x80                            ;call kernel

    mov     eax,1                           ;system call number (sys_exit)
    int     0x80                            ;call kernel

section     .data

msg     db  'Hello, world!',0xa             ;our dear string
len     equ $ - msg                         ;length of our dear string
```

Assembly

```
#include<stdio.h>

main()
{
     printf("Hello World");


}
```

C

# HELLO WORLD IN C, ASSEMBLY, AND RUBY

```
section     .text
global      _start                          ;must be declared for linker (ld)

_start:                                     ;tell linker entry point

    mov     edx,len                         ;message length
    mov     ecx,msg                         ;message to write
    mov     ebx,1                           ;file descriptor (stdout)
    mov     eax,4                           ;system call number (sys_write)
    int     0x80                            ;call kernel

    mov     eax,1                           ;system call number (sys_exit)
    int     0x80                            ;call kernel

section     .data

msg     db  'Hello, world!',0xa             ;our dear string
len     equ $ - msg                         ;length of our dear string
```

Assembly

```
#include<stdio.h>

main()
{
     printf("Hello World");


}
```

C

```
puts “Hello world!”
```

Ruby

# THIS IS WHY RUBY IS SO AWESOME

# USING VARIABLES

- We can tell our program to remember values for us to use later on, kind of like the *x* in algebra. Remember that?
- The action of saving a value to memory is called **assignment**
- The entity we use to store the value is called a **variable**
- The action of getting the value of a variable is called **accessing** the variable
- You *cannot* access a variable that has not been defined yet
- We will use all the above techniques to store values into variables, and generate new values using existing variables

# CODE ALONG

# DATA TYPES

## DONEC QUIS NUNC

- Ruby (along with many languages) has several types of data. The basic ones are
  - String (contains text data)
  - Fixnum (integers only, 1, 2, 42)
  - Float (numbers with a decimal point, floating point)
  - Boolean (true or false)
    - For the individuals that have prior software experience, 0 does not evaluate to false.

# STRINGS

- Strings start and end with either a single quote (‘) or double quote (“)
  - For example: “hello world” is a String

# PRINTING TEXT TO THE SCREEN

- It's common to print information out to your terminal screen. Ruby allows you to easily do this by using a command called "puts". For example:

  - puts "Hello World"

- This will print the words "Hello World" to your terminal screen when you execute this ruby code.

- The puts command can take *any* data type and print it to the screen. (Strings, numbers, you name it)

# GETTING USER INPUT

## IN COMMAND LINE RUBY APPS AT LEAST

- **set a variable to** `gets.chomp`
- For example: `input = gets.chomp`
- This creates a prompt, waiting for the user to type and press enter.

# STRING INTERPOLATION & CONCATENATION

- *Concatenation* sounds like something that should never be attempted near an open flame, but it's pretty simple. Just a matter of mashing one thing into another.

- `"String " + "String"` gives you a string.

- Don't forget spaces!

- You can insert variables with this syntax, which is  nicer:

- `puts "Hey everyone, did anyone see #{ somevariable } anywhere?"`

# LAB

- Navigate to your folder in the class repo, then cd into the `class2` folder
- Create a folder called `classwork`, and use `touch` to create `why_hello.rb`.
- Use `subl` to open the file in sublime text.
- Make a Ruby program that asks for a first name, then a last name and saves them as variables.
- Then print to the screen a nice greeting like in my example.
- You have 5 minutes!

# BREAK

# NUMBERS

- Numbers are just the number
- 3 is not the same as "3"
  - For example: 123 or 42.12
  - An integer mixed with a float gives you a float
  - Funsies: 1_000_000 becomes 1 million (underscores are allowed to help make larger numbers more legible)
- Need to change to string to concatenate with strings.

# NUMBERS

| Operator | Meaning | Example |
|---|---|---|
| + | Addition | 8 + 10 |
| - | Subtraction | 10 – 8 |
| * | Multiplication | 12 * 2 |
| / | Division | 10 / 5 |
| % | Modulus | 10 % 6 |

# REUSING CODE

## METHODS

- The same way we can store VALUES in memory by using variables…
- We can store CODE in memory by using methods.
- In other words, we can train the program to 'remember' a set of commands, and give that set of tasks a command name
- Then, we can call that command by name and the program will perform those tasks

# DEF JAM

- In ruby, we define methods with a keyword, “def”
- It’s used like this:

```
def omaha
  puts “run up the middle”
end
```

- def “omaha” creates a method called omaha
- Everything in-between the “def” and “end” is the procedure to be run.

# CODE ALONG

# CONDITIONAL LOGIC

## Booleans

- Besides strings and integers, Ruby also has a Boolean data type
- A boolean is a simple value that is either true or false
- When different data types are compared to each other, the result of that comparison is a boolean result
- (e.g. 5 < 7 would result in “true”)

# OPERATORS

| Operator | Description | Example (a =4 and b= 2) |
|---|---|---|
| == | Equal | a == b<br>*false* |
| != | Not Equal | a != b<br>*true* |
| > | Greater than | a > b<br>*true* |
| < | Less than | a < b<br>*true* |
| >= | Greater than or equal to | a <= b<br>*false* |
| < = | Less than or equal to | a <= b<br>*false* |
| ⇔ | same value? return 0<br>less than? return -1<br>greater than? return 1 | a <=> b<br>*1* |
| .eql? | same value and same type? | 1.eql?(1.0)<br>*false* |

# TRUTHY VS FALSEY

- Some values in Ruby are “falsey”. Unlike some programming languages, 0 and “” (empty string) are not falsey in Ruby.

- False and nil are falsey

# CONDITIONALS

- Conditionals let our program make decisions.
- They usually start with the if keyword and end with end.
- If there are multiple conditions, the elsif keyword is used. else is the default fallback if none of the above are true. The first condition to be true runs, and the rest don’t even evaluate.

# BELIEVE IT OR NOT, I WAS ONCE A BOUNCER

# CREATE A FILE TITLED BOUNCER.RB AND OPEN IT IN SUBLIME TEXT

# TAKEAWAYS

- Data Types
  - Integers
  - Float (number with decimals)
  - String
  - Booleans
- Variables
  - Store values
  - Can be passed to methods as parameters

- Methods let us train the program to 'remember' a set of code to perform later
- Making a new method is called declaring a method
- Declaring a method does NOT run the method immediately
- If the method takes in variables to use while it is doing its tasks, those are called parameters

- Conditionals let your program make decisions by evaluating true / false (boolean) statements.
  - if
  - elsif
  - else
  - end