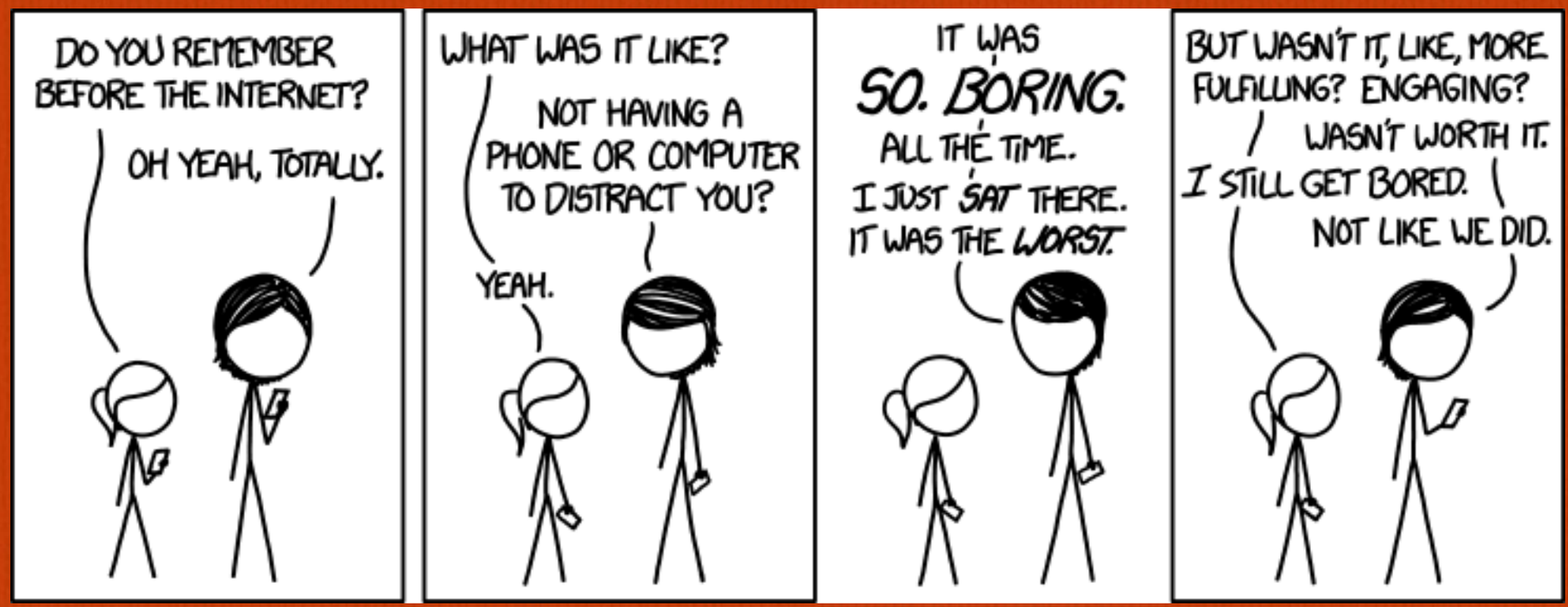


Hello World!

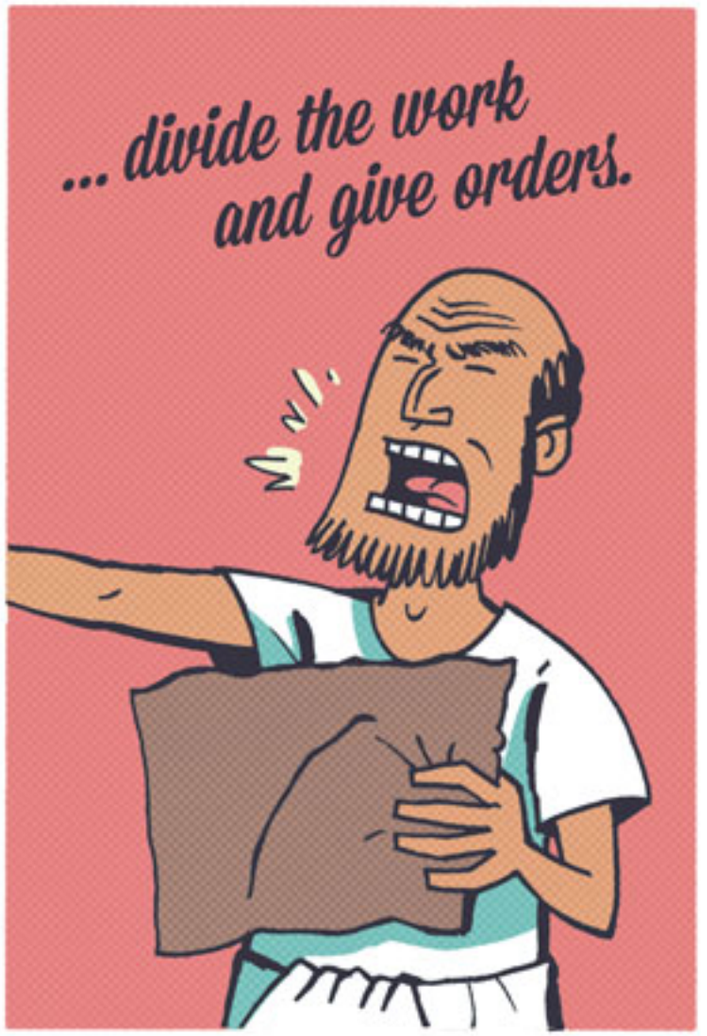
Intro to Web Development with Rails

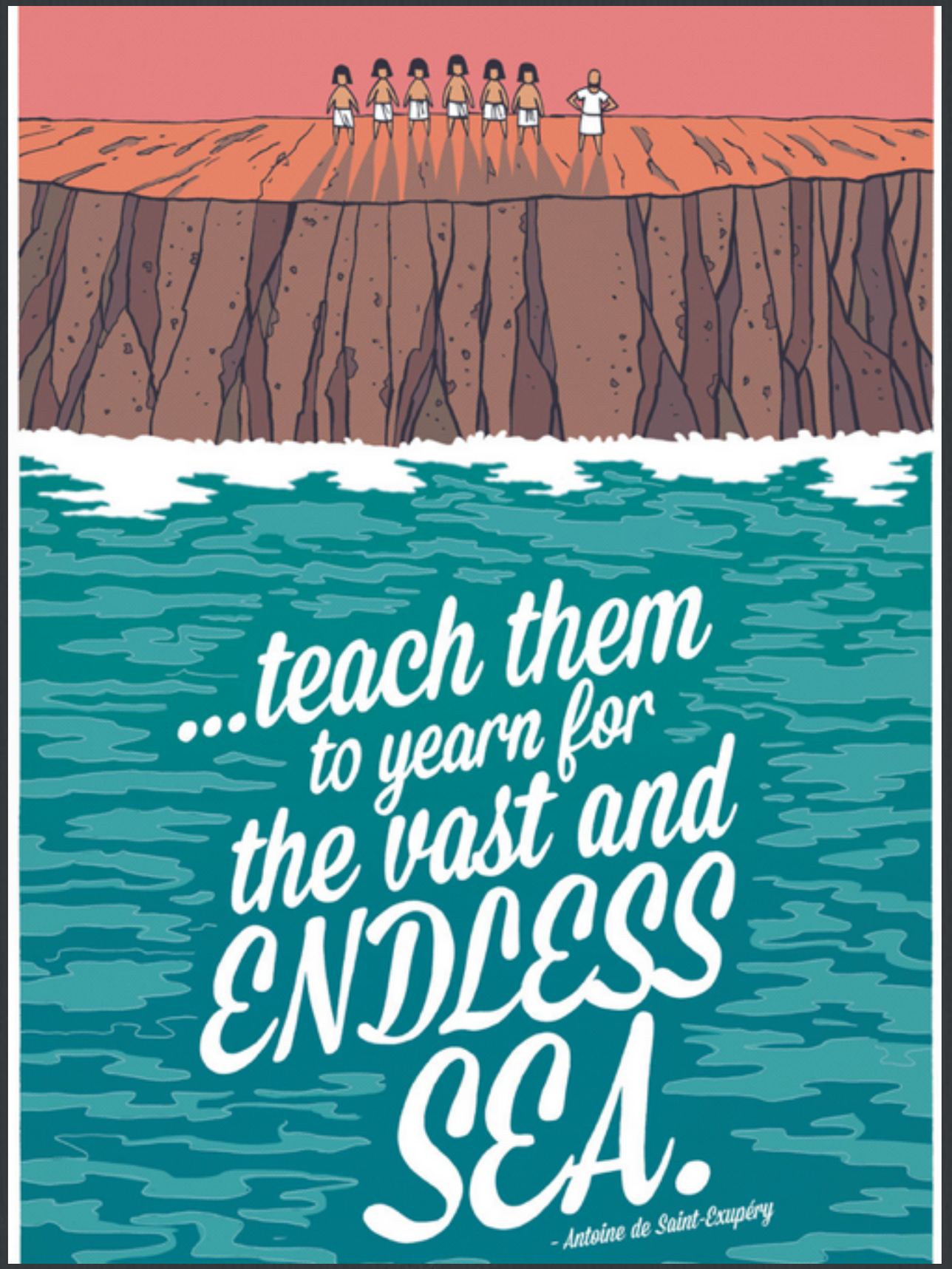
Today's objectives

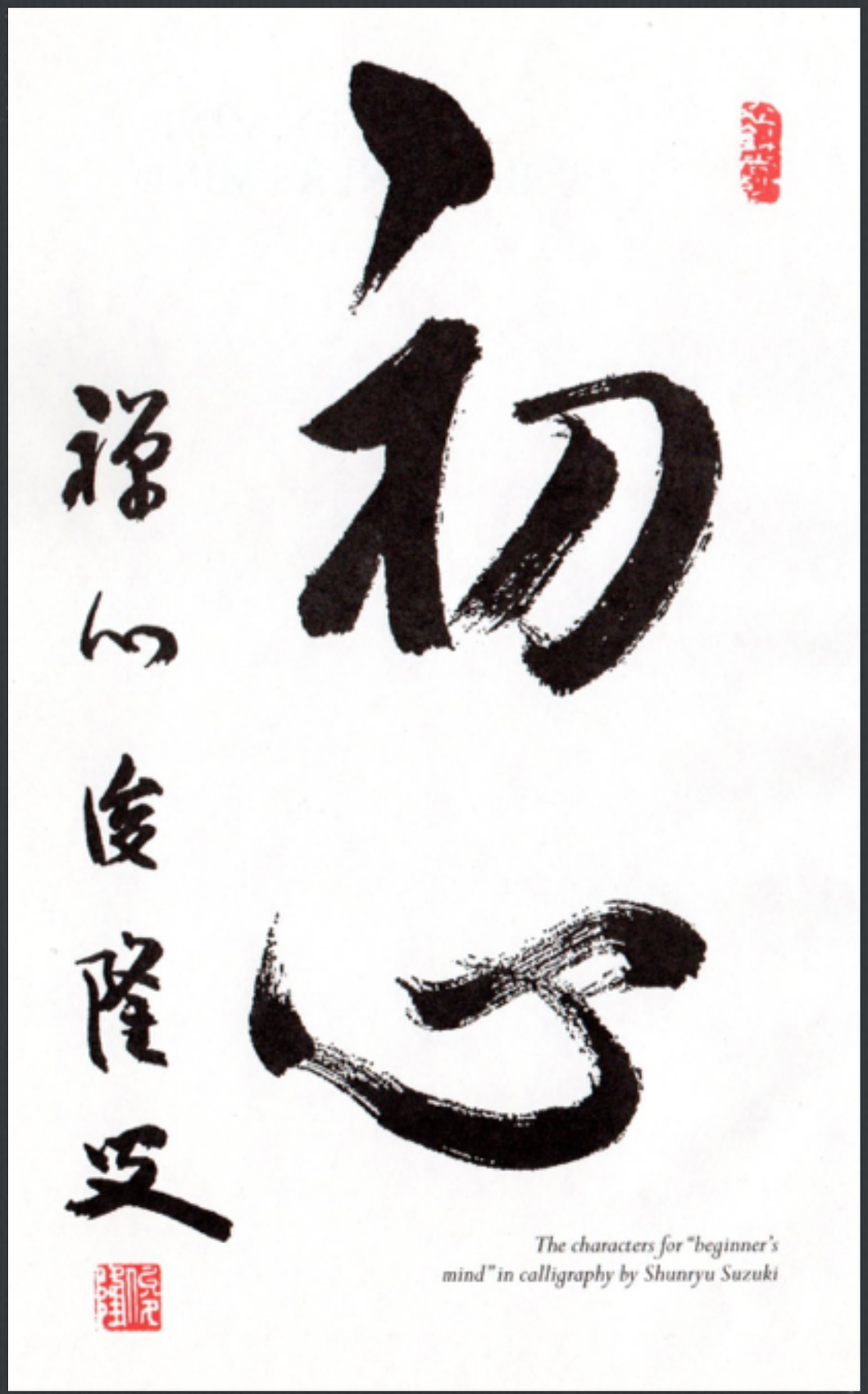
- Set expectations for the workshop
- Learn about who we are
- Get our hands dirty with the command line, Ruby and Git
- Make our first program!


If you want
to build a ship...


... don't drum up the
men to gather wood ...


... divide the work
and give orders.


...teach them
to yearn for
the vast and
ENDLESS
SEA.
- Antoine de Saint-Exupéry



The characters for "beginner's mind" in calligraphy by Shunryu Suzuki

In the beginner's mind there are many possibilities, in the expert's mind there are few.

-Suzuki Shunryu

Hi, I'm Vincent

- From Jersey
- Made my first website when I was in 7th grade at the library computer, titled 'Get outta my site!'
- Majored in English at Rutgers
- Tutored freshmen in Writing, became a 'professor' at a terrible ripoff school in Queens soon after graduation.
- Moved to Yokohama, Japan and taught English in public Junior High schools, did that for 3.5 years.

Hi, I'm Vincent

- While I was at it, I taught myself Japanese, started doing some translation work
- Thanks to a fortuitous chance meeting in a bar, met a man with whom I helped start a magazine business
- That led me to want to become a better journalist, so I went to CUNY.
- While producing packages for the web, I realized that I enjoyed coding WAY more than reporting, writing, editing, or producing stories

Hi, I'm Vincent

- Started learning Ruby in like 2012
- Took a part time class on Rails at GA
- In early 2014 I decided to take WDI in this very building
- 3 weeks later, started working at 2U, an education technology company
- Taught Back End Web Development at GA just recently
- When not coding, I bike and read a lot and I like taking pictures with old-school technology.

Your turn

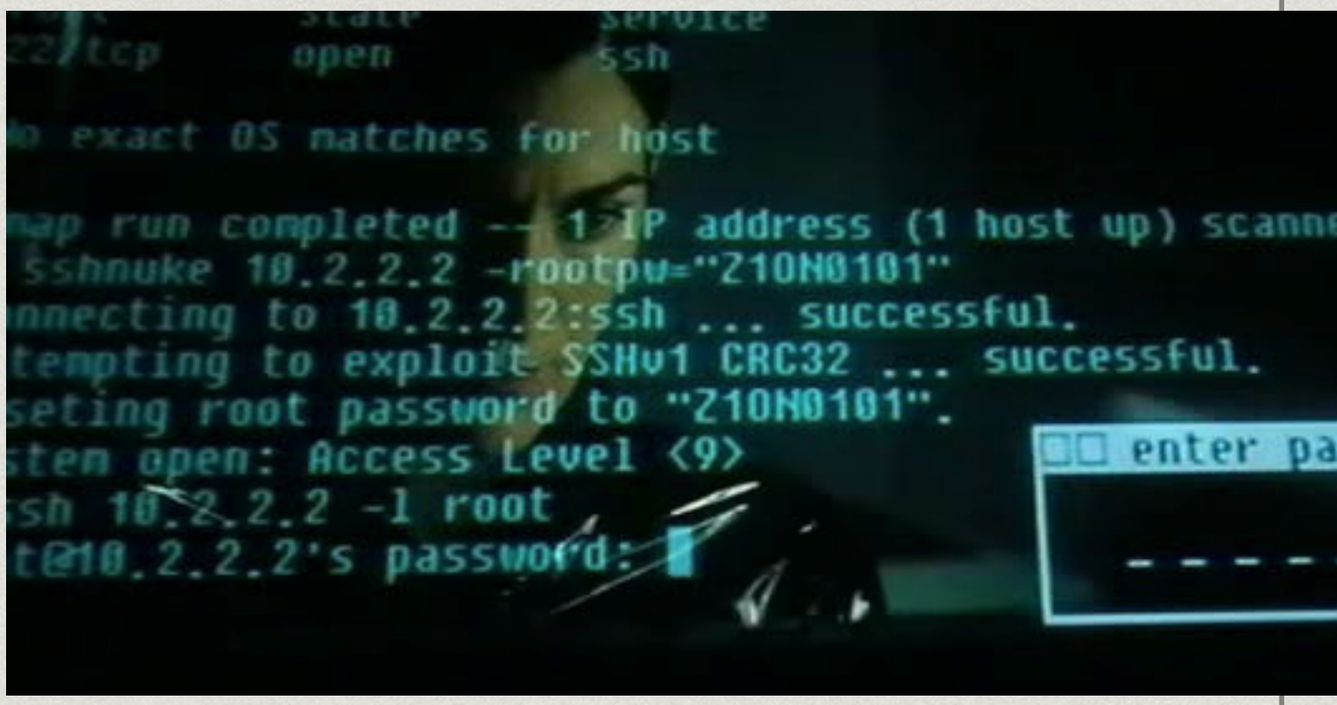
- What's your story? What are you doing now?
- What's your programming background if any?
- What you hope to get achieve in this class?

COMMAND LINE + GIT

DATE

Why bother with the command line?

- Godmode for using your computer
- It's how we will run standalone Ruby applications.
- It's the best way to issue commands to a remote server, view logs
- Faster than GUIs when you know what you are doing!



Downsides

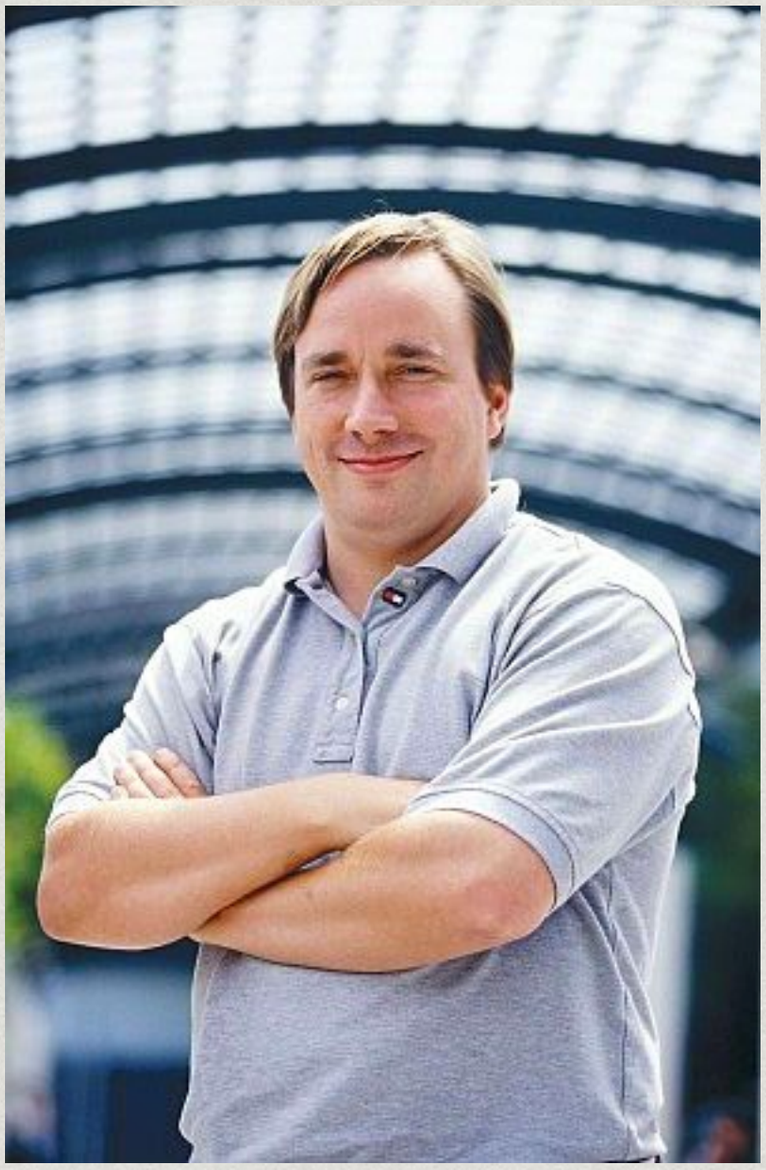
- Easily destroy your entire computer if you aren't careful
- When you delete something, it's GONE.
- Takes some getting used to, like typing or learning to swim.

GIT


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

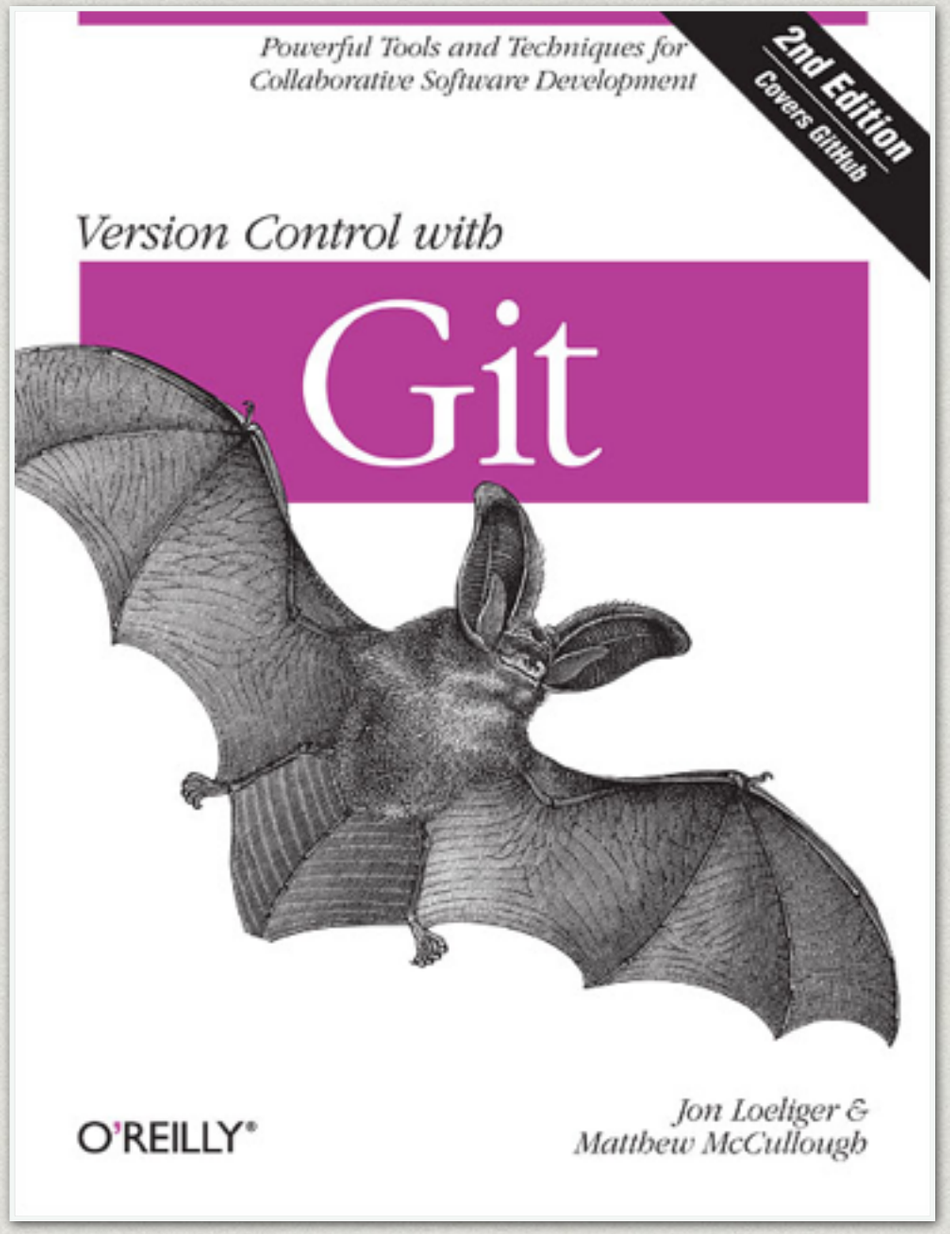
What is Git?

- Git is a source control management tool started by Linus Torvalds, the same dude who gave us Linux.
- Git allows you to store and update your code in a structured way.
- Git includes history of changes you make, so you can create "snapshots" and track your work better over time.



Git can be super tricky

- I mess something up with git roughly twice a week, probably.
- It takes a lot of getting used to. We'll cover the absolute basics today



456 pages!!!

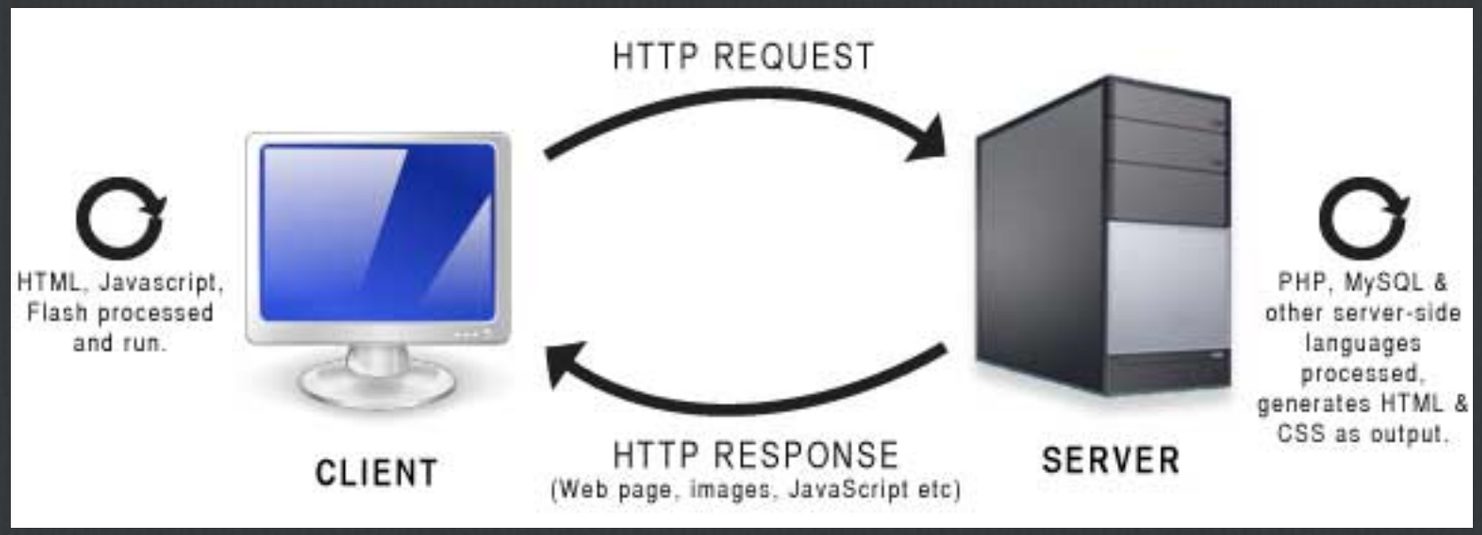
Let's practice

Break time

Ruby first.

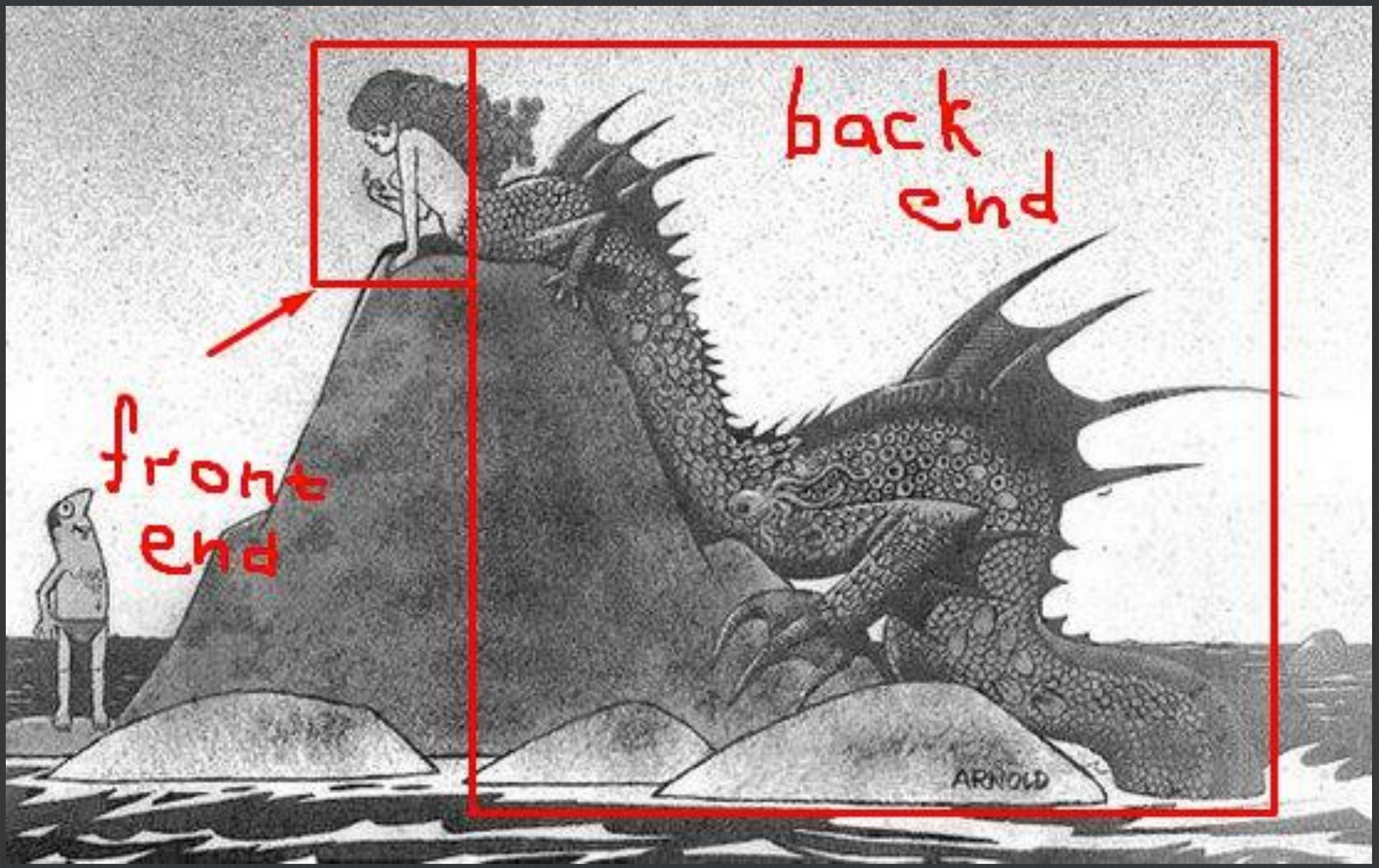
- It will be easier to navigate a Rails project once we have a basic understanding of Ruby.
- We will first teach you how to write simple Ruby programs as standalone applications.
- Once we have become familiarized with Ruby, we will start building Rails applications (which are groups of Ruby files that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

Check your Ruby and Rails installations



Front end

Back end


back end
front end
ARNOLD

- The back end of a web application is the part that happens on the server.
- Users interact with the front end, and the back end code decides what to do with it.
- Almost invariably, this involves interacting with a database for persistent storage.
- The *client* is whatever consumes the data sent from the back end. This might be a browser or a mobile app.
- We will be focused on just getting the data to a browser, and we don't care about how it looks, really.

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

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.
- Takes care of the more boring parts of programming.
- High level - Ruby is itself an *abstraction* of C, which makes you handle boring stuff like reserving and relinquishing memory.

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

LET'S 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

- Create a folder called `classwork`, and use `touch` to create `why_hello.rb`.
- 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 just 5 minutes!

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

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

DEF JAM

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

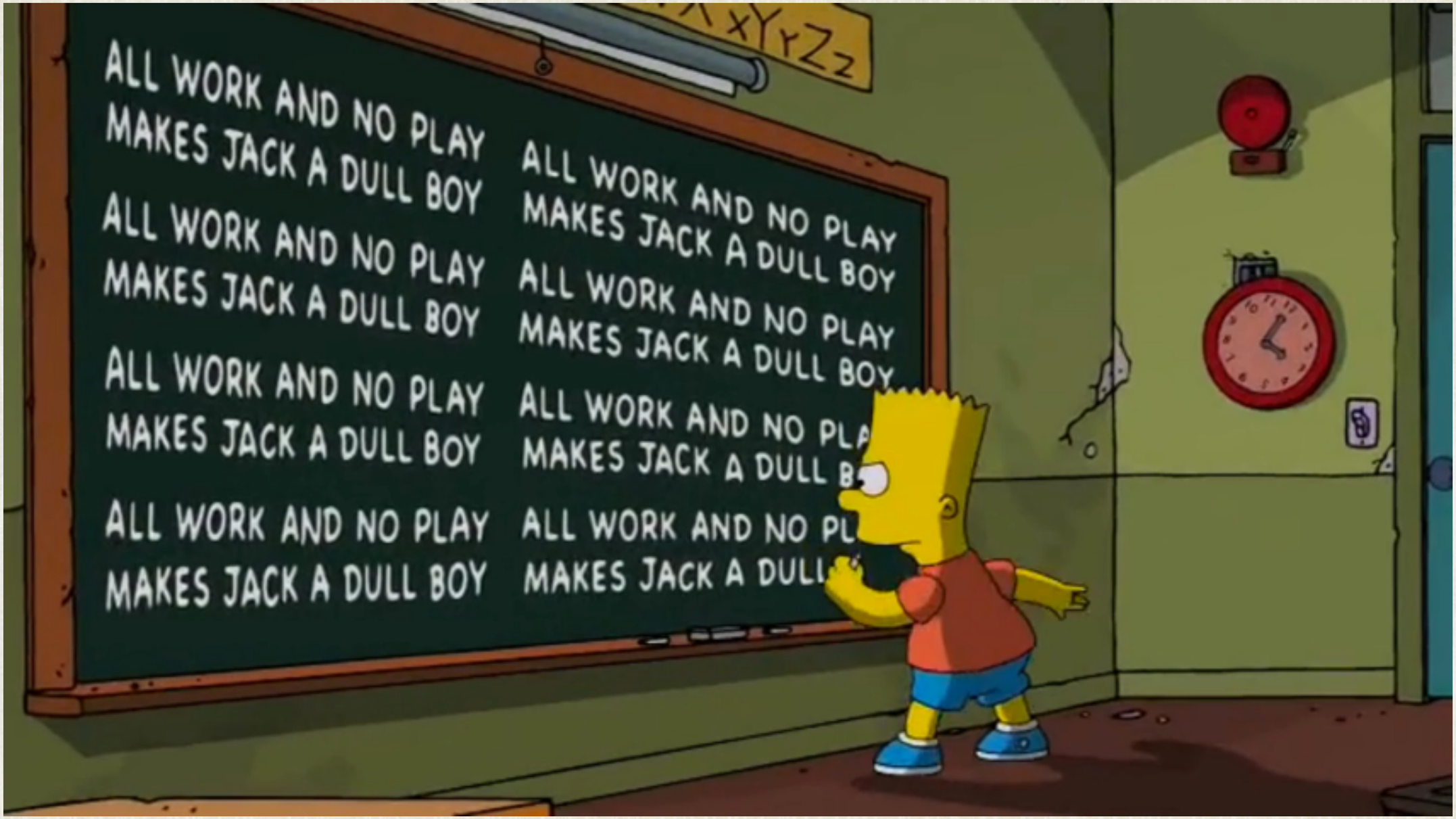
```
def omaha
  puts “run up the middle”
end
```

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

- 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

BELIEVE IT OR NOT, I WAS ONCE A BOUNCER

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


ALL WORK AND NO PLAY
MAKES JACK A DULL BOY
ALL WORK AND NO PLAY
MAKES JACK A DULL BOY
ALL WORK AND NO PLAY
MAKES JACK A DULL BOY
ALL WORK AND NO PLAY
MAKES JACK A DULL BOY
ALL WORK AND NO PLAY
MAKES JACK A DULL BOY
ALL WORK AND NO PLAY
MAKES JACK A DULL BOY

REPETITION

LOOPS AND ITERATION

- Computers' tireless repetition and iteration of operations is what makes them so powerful.
- This way, we don't have to be like Bart Simpson, repeating ourselves. Let the computer do that.

An Example of a loop

```
# bart.rb

100.times do
  puts “I will not xerox my butt”
end
```

Will result in…

```
$ ruby bart.rb
I will not xerox my butt
I will not xerox my butt
I will not xerox my butt…
```

100 times!

A loop is a type of block

Start of block

```
100.times do
  puts “I will not xerox my butt”
end
```

End of block

The code that will run 100 times

BLOCKS

```
# loops.rb

3.times do
  puts “going...”
end
```

- The purple “do” and “end” indicate the start and end of something called a “block”. A block can be inside a method.
- A block is similar to a method, in that it contains a routine of code to be executed.
- The largest difference between blocks and methods are that they don’t have a “method definition”
  - i.e. You can’t execute a block via “do_something()”

UP TO X

```
# upto_loop.rb

1.upto(10) do |num|
  puts “#{num}. going…”
end
```

- num starts at 1 and ends at 10
- It’s essentially a throw-away variable that changes at every cycle of the loop.
- num is only scoped within the block. It is meaningless outside of it.

ARRAYS

ARRAYS

- Arrays are an ordered collection of objects
- Arrays can contain any type of data (string, int, other arrays ...)
- Arrays can be extremely long (based on how you have ruby / your computer setup)
- Arrays can be iterated over to get every object individually

ARRAY SYNTAX

- Arrays start with [and end with], for example:
  - `[1, 2, 3]` generates an Array with 3 fixnums
  - `["a", "b", "c"]` generates an Array with 3 strings
  - `["1", 2, "abc"]` generates an Array with a mixed object collection
  - `["1", 2, [1,2,3], "abc"]` generates an Array with a mixed object collection

ARRAY SYNTAX

- Accessing an Array uses [and] as well, but on the array itself.
- `people = [“Vincent”, “Eddie”, “Ali”]`
  - `me = people[0]`
  - `eddie = people[1]`
- All array’s start at 0, getting the first element must be 0
- You can use -1 to retrieve the last element
  - troll = people[-1]

ARRAYS RECAP

- Arrays are ordered collections of data
- Each element can be accessed with an index
- The index is its *offset* from the first. The first one is 0, second is 1, and so on. You can use -1, -2 etc to count from the back.
- Arrays have lots of methods such as `.class`, `.size`, `.push(x)`, `<<` `.unshift`, `.pop`, `.shift`, `.include?`, `.sort`, `.uniq`, `.shuffle`, `.join`, `.to_s`

RUBY'S .EACH METHOD

- A way to perform an operation on "each" item in a collection.
- An item in a collection is a thing in an array, or a value in a hash
- The each method allows us to perform a routine on every item in the collection

THE .EACH METHOD OUTPUTS THE ORIGINAL ARRAY

```ruby
awesome_band = [ "sporty", "scary", "ginger", "posh", "baby" ]

awesome_band.each do |sg|
  puts sg
end
```

```
sporty
scary
ginger
posh
baby
=> ["sporty", "scary", "ginger", "posh", "baby"]
```

Class =

the abstract idea of 'house'
There are many like it.

Instances of that class =

This house in particular.
There are many like it but
this one is mine.

- Classes start with:
  - `class MyClassName`
- And end with:
  - end

```
class MyNewClass
end
```

- Classes MUST start with a capital letter
- Ruby comes with data type classes.
- You can find out the class of an object with .class
- you can find out what a class can do with .methods

CODE ALONG IN IRB

HASHES

- Often referred to as dictionaries
- Each entry in a hash needs a key and a value. The value can be any data type, including Hash.
- If you access a hash at a specific key, it will return the value at that key
- Think of a paper dictionary:
  - You know the word, but not what it means. It’s position in the
  - You look up the definition by going to where the word is in the book
  - You then read the definition

ACCESSING VALUES BY KEY

```
1  user = {"name" => "Vincent", "age" => 31, "home" =>
   "Brooklyn", "email" => "trivett@gmail.com"}
2
3  user["name"]
4  => "Vincent"
```

SETTING VALUES

```
1  user = {"name" => "Vincent", "age" => 31, "home" =>
   "Brooklyn", "email" => "trivett@gmail.com"}
2
3  user["job"] = "developer"
4
5  user
6  => {"name"=>"Vincent", "age"=>31, "home"=>"Brooklyn",
   "email"=>"trivett@gmail.com", "job"=>"developer"}
7
8
```

METHODS

```
1  user = {"name" => "Vincent", "age" => 31, "home" =>
   "Brooklyn", "email" => "trivett@gmail.com"}
2
3  user.has_key? "email"
4  =>true
5  user.keys
6  => ["name", "age", "home", "email"]
7
```

SYNTAX DIFFERENCES

Ruby 1.9+ Alternate Syntax

```
user = {:user_name => "SalmanAnsari", :email => "salman.ansari@gmail.com"}

# becomes

user = {user: "SalmanAnsari", email: "salman.ansari@gmail.com"}

# a little bit more concise
# more closely matches JSON format
# considered an 'alternate' syntax, not a replacement
```

Only works when the keys are symbols

Don’t get too hung up on syntax at the moment.

AWESOME