WHAT DOES IT MEAN??

Why Program?

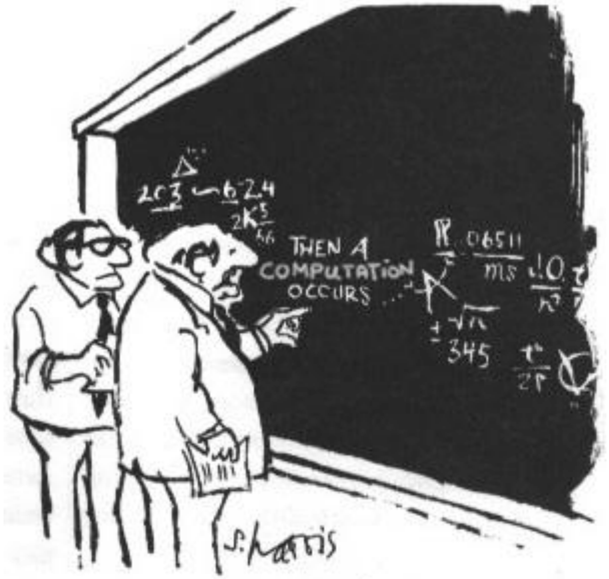
- Applications come from the needs of the present: *your needs*
- Effectively articulating needs is the first step
- Express complex logic and perform computation
- Do things that would take a human a long time to do
  - counting
  - comparing
  - repeating

Digital Humanities Programming?

What is a programming language?

An artificial language with a limited purpose

A means of expressing computations (math) and algorithms (logic)



" good call using a computer here in step two."

What is a programming language?

...like human languages in some ways!

- Syntax (form)
- Semantics (meaning)
  - signs/words (variables, symbols, numbers, strings)
  - expressions
  - "flow" (decisions, conditions, loops, narrative)
  - complex entities (methods, structures, & objects)

"when you don't create things, you become defined by your tastes rather than ability. your tastes only narrow & exclude people. so create."

why the lucky stiff (@_why)

Software Terminology

- **Operating System** talks to computer hardware
- **Application** sends `input` to the operating system and receives `output`

Language

- Code used to create applications
    - Ruby
    - PHP
    - Python
    - JavaScript
    - Java
    - C++
    - C
    - many, many more...

Language Choice

- Is it “easy” to maintain?
- Is the standard library good enough?
- Can developers learn it?
- Can you live with the syntax?

Library

A collection of reusable code to accomplish a generic activity

- Date math (three months from today)
- Logging
- Working with file systems
- Compressing files

Framework

- Collection of reusable code to facilitate development of a particular product or solution
    - Twitter Bootstrap
    - Rails
    - Susy
    - jQuery

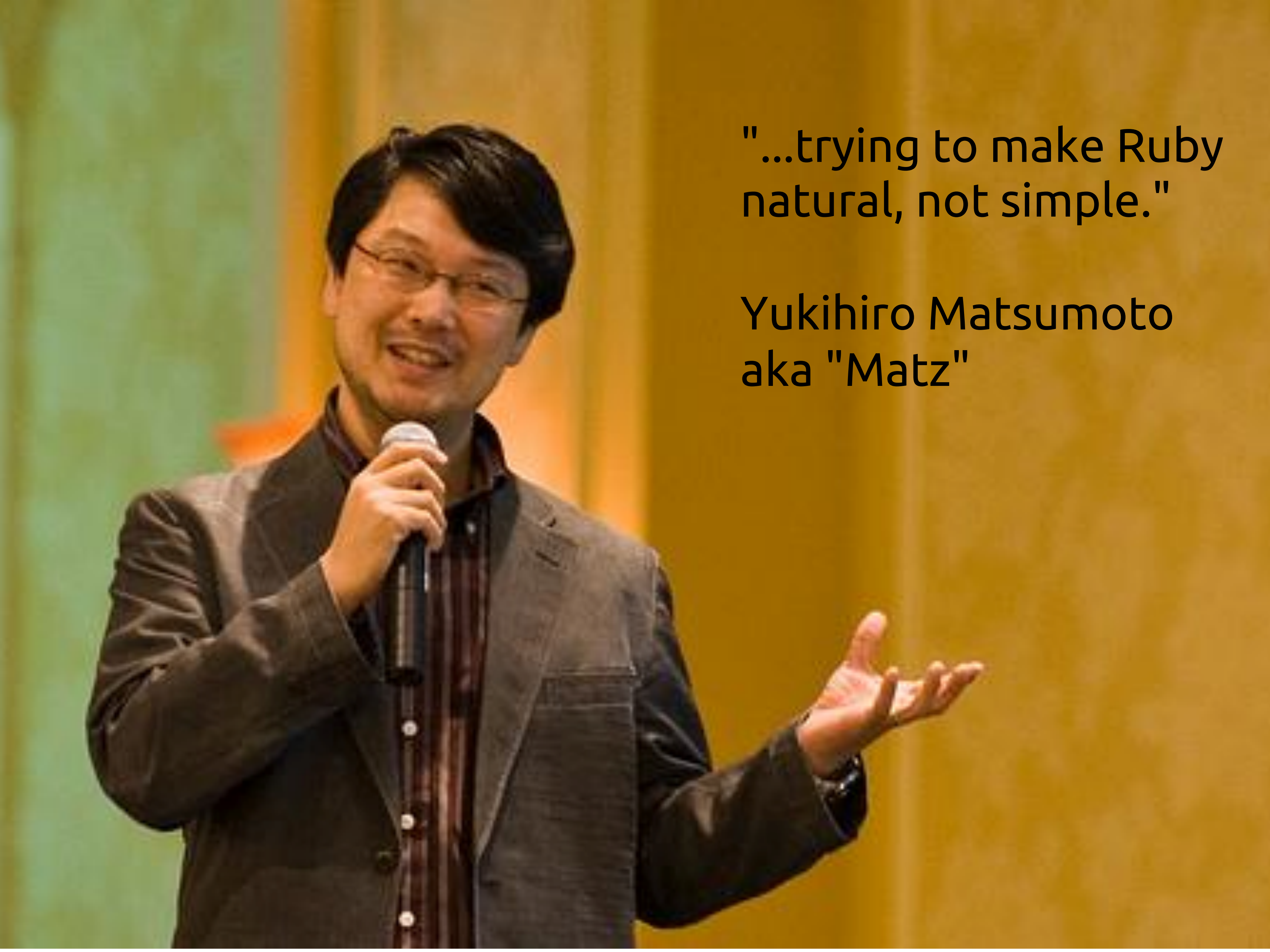
Ruby vs. Rails

- Ruby is a language
- Gems are Ruby libraries
- Rails is a framework
    - Written in Ruby
    - Contains many Ruby gems
    - Used to build web applications

Ruby Philosophy

"Principal of least surprise"

- People want to express themselves when they program
- People don't want to fight the language
- Programming languages must feel natural



"...trying to make Ruby natural, not simple."

Yukihiro Matsumoto
aka "Matz"

Ruby Philosophy: @matz

“I tried to make people enjoy programming and concentrate on the fun and creative part of programming when they use Ruby”

Ruby Philosophy: applied

- Ruby is a *humane interface* (many ways to do things)
- Favors readability and variety over concision and perfection
- Sometimes this makes code harder to understand, but usually it's easier
- Contrasts with a *minimal interface* with one (or very few) "correct" ways to do things

Many Rubies

Ruby 1.0 (1996)

Implementations

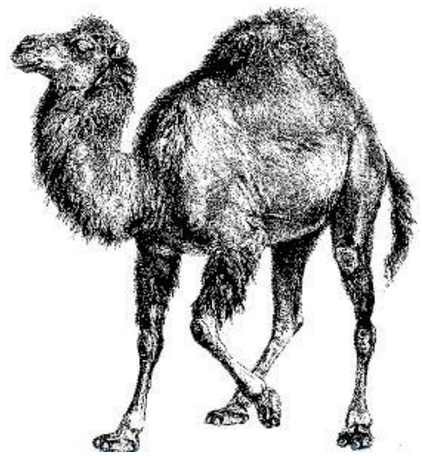
- MRI
- REE
- Jruby
- Rubinius
- MagLev
- MacRuby

Many Versions

- MRI 1.9.3
- MRI 2.1.2
- Jruby 1.7.13
- ...

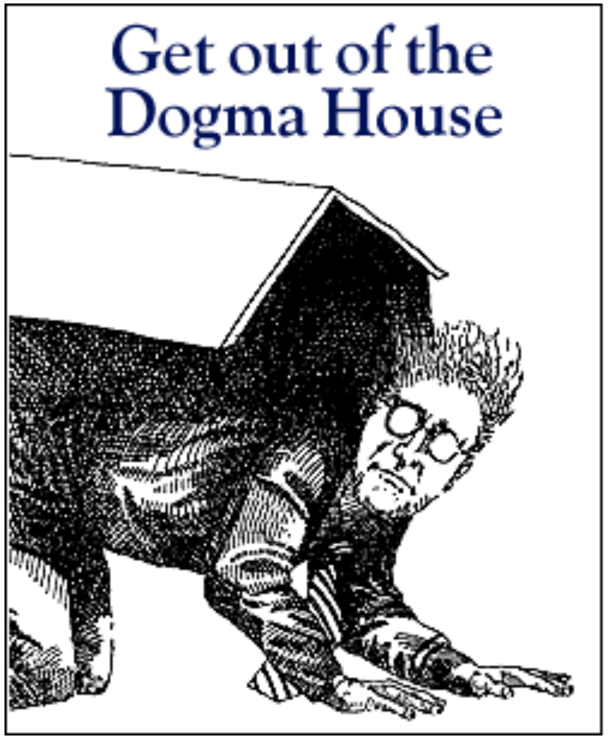
Myth

- Scripting languages don't scale
  - Facebook (PHP)
  - Twitter (Ruby)
  - Google (Python)
  - Slashdot (Perl)



Dogma

- Language x is not web-scale
- Language x is not enterprise
- Language x does not scale
- The x framework doesn't handle this weird edge case



EVERY TIME YOU CODE IN PHP
GOD KILLS A KITTEN
notmeme.net

Why Ruby?

- General purpose
- Usable on your computer or over the web
- English-like syntax and useful built-in features
- Doesn't require a compiler
- "Fun" to write
- Object-oriented

Why Not Ruby?

- Not as easy to run on the web as PHP
- Used less often than PHP, and major platforms (WordPress, Drupal, Omeka) use PHP
- Ruby isn't Rails
- Object-oriented languages are conceptually difficult to grasp

What we will cover

What is a **`data type`**?

What is a **`variable`**?

What is an **`operator`**?

What you will be able to do

create numeric and text information

store information in variables

print information to the screen

Open the Terminal

- Windows: `git bash`
- OS X: `iTerm2`

Prompt

- Terminals show a line of text after a command finishes
- Whenever instructions start with "`$ `", type the rest of the line into the terminal
- Let's give the terminal a `command` to open Interactive Ruby (IRB)

```
$ irb
```

irb: Interactive Ruby

IRB has its own prompt with ends with `>`

```
$ irb
>
```

You can use `Control + D` to exit IRB at any time or type `exit` on its own line

Variables

"words" that refer to information

Variables

Give it a name so we can refer to it
It's information can be changed

```
$ irb
>  my_variable = 5
=> 5
> my_other_variable = "Hi"
=> "Hi"
> my_variable = 10
=> 10
```

What's with => ?

- Setting a variable to a value is called "**assignment**"
- What types of information can we hold in a variable?

Variable Assignment

- Variables are assigned using a single equals sign (`=`)
- The ***right*** side of the equals sign is **evaluated first**, then assigned to the variable name on the ***left*** side of the equals

Variable Assignment

```
apples = 5
bananas = 10 + 5
fruits = 2 + apples + bananas
bananas = fruits - apples
```

Variable Naming

all letters (`folders`)

all numbers (`2000`)

with an underscore (`first_name`)

with a dash (`last-name`)

a number anywhere (`l33t`)

a number at the start (`101dalmations`)

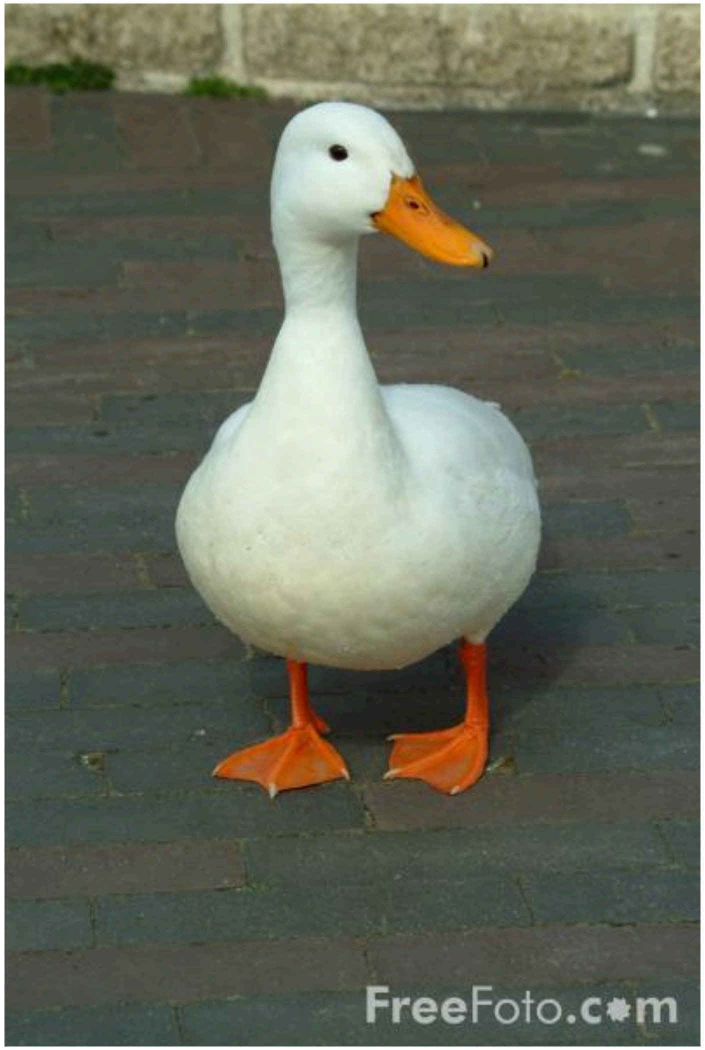
a number at the end (`starwars2`)

Variable Naming

Be descriptive of the “thing”

Ruby is a "duck-typed" language

Duck-typing



If it looks like a duck and it quacks like a duck, chances are it's a duck.

Types of ducks

standard types:

numbers & letters

Numbers & Letters

integers:

```
4, 1040, -55, 9999
```

floating-point numbers:

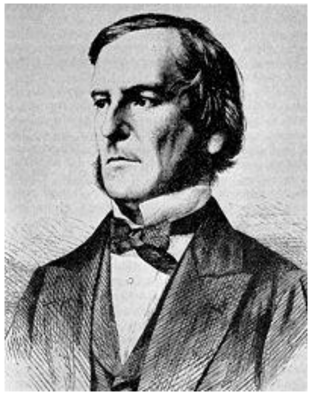
```
1.1, 0.444, 9999.0001, -3.33
```

text (strings):

```
"a", 'cat', "The quick brown fox jumped over the
lazy dogs.", '8 keys', '7'
```

boolean (yes or no?):

```
true, false, 0, 1
```



Strings

Strings are text; it must be wrapped in a **matched pair of quotation** marks.

```
$ irb
> 'Single quotes work'
=> "Single quotes work"
> "Double quotes work"
=> "Double quotes work"
> "Start and end have to match'
">
```

Exercise

Create variables named `first_name`, `last_name`, and `favorite_color`

Assign string values to the variables

Numbers

- Numbers *without a decimal* point are **integers**
  - `0`
  - `-105`
  - `898989898`
  - `2`
  - `-898989898`

Numbers

Numbers *with decimal points* are floating point numbers (**floats**)

- `0.0`
- `-105.56`
- `.33`
- `.000004`
- `3.14159265359`

Numbers

- You can perform operations on both types of numbers
    - `+`
    - `-`
    - `/`
    - `*`

Exercise

- Try dividing an integer by an integer
- Try dividing an integer by a float
- How are the results different?
- Create two integer variables named `num1` and `num2` and assign your favorite numbers
- Compute the sum, difference, quotient, and product of these two numbers and assign these values to variables named `sum`, `difference`, `quotient`, and `product`

An answer

```
num1 = 4
num2 = 5
sum = num1 + num2
difference = num1 - num2
quotient = num1 / num2
product = num1 * num2
```

Why does quotient = 0 ?

Collections

Collections

Collection Types: **Array**, **Hash**

- Define an Array
- Array syntax
- Array indexing
- Array methods
- Definition of a hash
- Hash syntax
- Hash indexing

Array

- An array is a list
- Each array is surrounded by `square braces` (aka `square brackets`) []
- Each element (`member`) is separated by a comma

```
> fruits = ["kiwi", "strawberry", "plum"]
=> ["kiwi", "strawberry", "plum"]
```

Exercise

- Make your own array named `grocery_list`
- Include at least 5 items in your grocery list in the array

Array

- Indexing
  - Members are stored in order
  - Each member can be accessed by its `index`
  - Ruby starts counting at ***zero***

```
> fruits[0]
=> "kiwi"
> fruits[1]
=> "strawberry"
> fruits[2]
=> "plum"
```

Exercise

- Still have your `grocery_list` array?
- What is at index zero in your grocery list array?
- How about index 5?
- Guess the answers and use the syntax examples to see if your guesses are correct
    - hint: `fruits[0]`

Hash

- In a hash, we can refer to a member by a keyword instead of a number
- Each member is a pair
    - ***Key***: address of the hash member
    - ***Value***: variable contained by the member, and located by the *key* name
- Other names for a hash:
    - `dictionary`
    - `associative array`
    - `map`

Hash Syntax

- Surrounded by **curly braces** (aka curly brackets) `{}`
- **Commas** separate each member pair
- A *key* uses `=>` (the rocket) to point to its *value*

```
> states = {"VA" => "Virginia",
"MD" => "Maryland"}
=> {"VA" => "Virginia", "MD" =>
"Maryland"}
```

Exercise

Define a hash named `my_info` that contains the following keys

- `first_name`
- `last_name`
- `hometown`
- `favorite_food`

Hash Indexing

- Member pairs can be accessed by their key
  - Each **key** needs to be *unique*
  - **Values** *do not* need to be unique

```
states["MD"]
=> "Maryland"
```

Exercise

- Add the key `good_food` to your `my_info` hash and give it the same value as your `favorite_food` key. What happens?
- Add a second `favorite_food` key to your `my_info` hash. What happens?

Methods

- **Things** that do **stuff**
  - Objects (like strings, integers, and hashes) are **nouns**; methods are verbs
  - Called (used) with a "`.`"
    - `5.to_s` (`to_s` is the method)
  - `5 + 5` is a shortcut way of writing `5.+(5)`
- Each data type has a set of built in methods.
  - See String's methods http://www.ruby-doc.org/core-2.1.2/String.html

Exercise

- Create a **String** variable named `old_string` and assign it the value "Ruby is cool"
- Use String methods to modify the `old_string` variable to that it is now "LOOC IS YBUR" and assign it to another variable named `new_string`
  - **Hint**: look at the String methods "`upcase`" and "`reverse`"

Booleans

A boolean can only have one of two values:
`true` or `false`

```
> 1 + 1 == 2
true
> 1 + 1 == 0
=> false
```

(`==` means "is equal to;" More on that later...)

Exercise

- Create a variable named `favorite_color` and assign it to your favorite color
- Create a variable named `not_favorite_color` and assign it to a different color
- Test if these variables are equal
    - Is equal to operator is `==`

Sometimes there is a problem...

Casting to appropriate type

- `to_s` (to string)
- `to_i` (to integer)
- `to_f` (guesses?)

Example:

```
> "3".to_f
=> 3.0
```

Operators: do stuff with objects

```
> my_variable + 2
=> 7

> my_variable * 3
=> 15

> my_other_variable + " there!"
=> "hi there!"

> fruits = fruits + ["lychee"]
=> ["kiwi", "strawberry", "plum", "lychee"]

> fruits = fruits - ["lychee"]
=> ["kiwi", "strawberry", "plum"]
```

Exercises

- Create an array named `vegetables` that contain three vegetables you like and one vegetable you don't
- Using the `vegetables` array, create an array named `my_vegetables` that contains only the vegetables you like
- **Extra**: can you use the first two arrays to create a new array named `your_vegetables` that only contains the vegetables you don't like?

More Operators

`+, -, /, *`	math operators (`+` also means concatenation)
`=`	assign a value
`+=`	addition, then assignment
`||`	or
`&&`	and
`==`	equal
`!=`	not equal

Printing things to the screen

```
puts "Doctor Who"

doctors = ['Matt Smith', 'David Tennent']
puts doctors[0]

best_episode = 'Blink'

puts "My favorite episode is " + best_episode

puts "My favorite Doctor is " + doctors[1]
```

Code Exercise 1

Store your street address, city, state, and zip code in variables (or even better, a hash!), then print them in the usual format:

```
Wayne Graham
123 My Street
Lexington, VA 22450
```

An Answer

```
address = {
    'name' => 'Wayne Graham',
    'street' => '123 My Street',
    'city' => 'Lexington',
    'state' => 'VA',
    'zip' => '24450'
}

puts address['name']
puts address['street']
puts address['city'] + ', ' + address['state']
+ ' ' + address['zip']
```

Code Exercise 1

Write a program that converts **seconds** to **years**. Test your program with `600000000` seconds, `60` seconds, and `40000.33` seconds.

An Approach

- Figure out how many seconds in a year
    - 60 seconds in a minute
    - 60 minutes in an hour
    - 24 hours in a day
    - 365 days in a year (365.242 if you're really precise)
- Do the math
- Return a result

An Answer

```
sec = 600000000.0

puts sec/60/60/24/365
```

Resources

- Rubylearning.com
- Learn to Program (http://pine.fm/LearnToProgram/)
- Why's Poignant Guide to Ruby (http://mislav.uniqpath.com/poignant-guide/)
- Ruby Documentation (http://ruby-doc.org/core/)
- "Pick-axe Book" (http://ruby-doc.org/docs/ProgrammingRuby/)