Hello Ruby

Part I

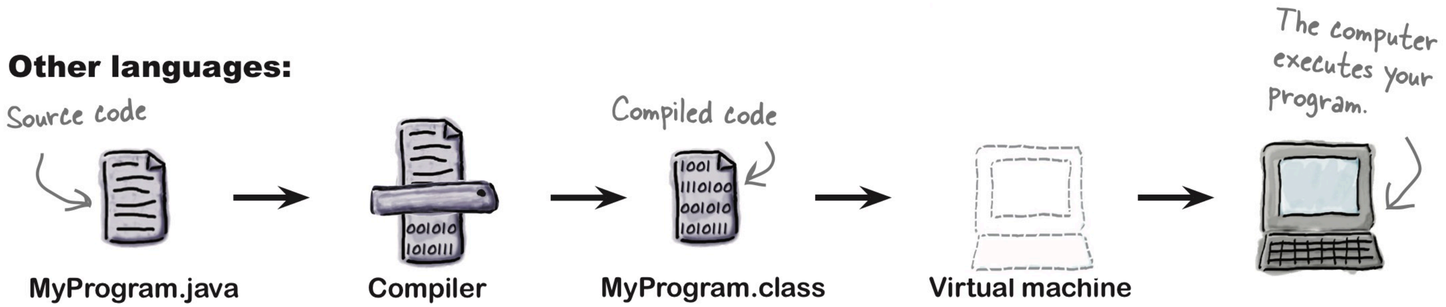
Outline

- **Basics I** - Math, strings, variables, printing and getting input
- **Basics II** - Everything in Ruby is an object
- **Basics III** - Conditionals
- **Basics IV** - Arrays and looping over things
- **Basics V** - Packing things up into functions
- **Basics VI** - Reading and writing text files

Before we start

Compiled vs Interpreted

Other languages:

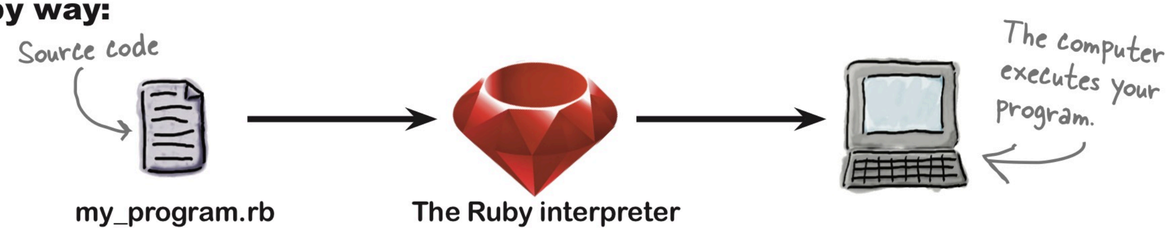


C++

Java

C#

The Ruby way:



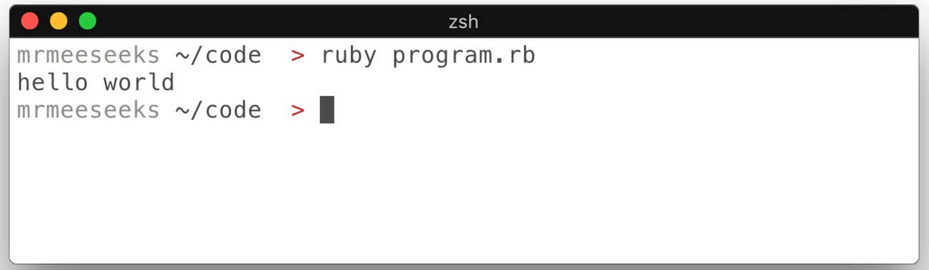
JavaScript

php

Test your programs!

```
puts "hello world"
```

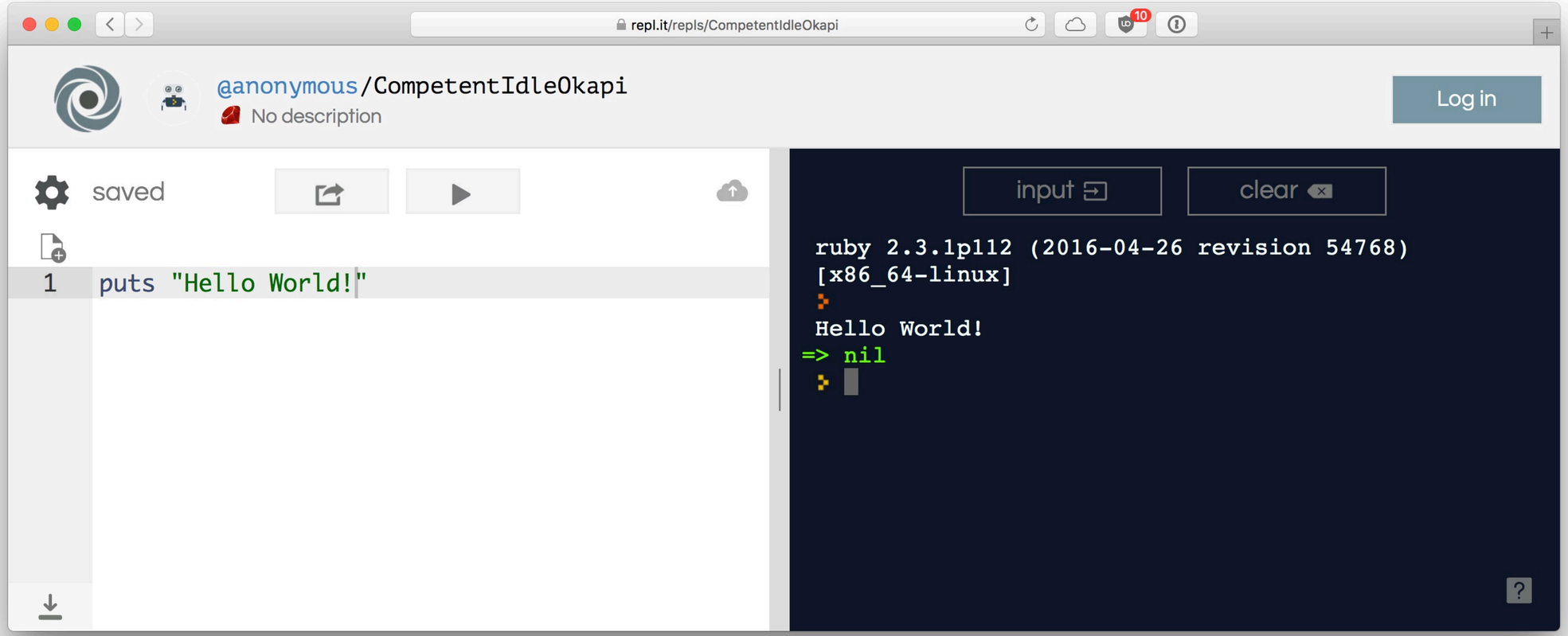
program.rb



Executing your program in the Terminal

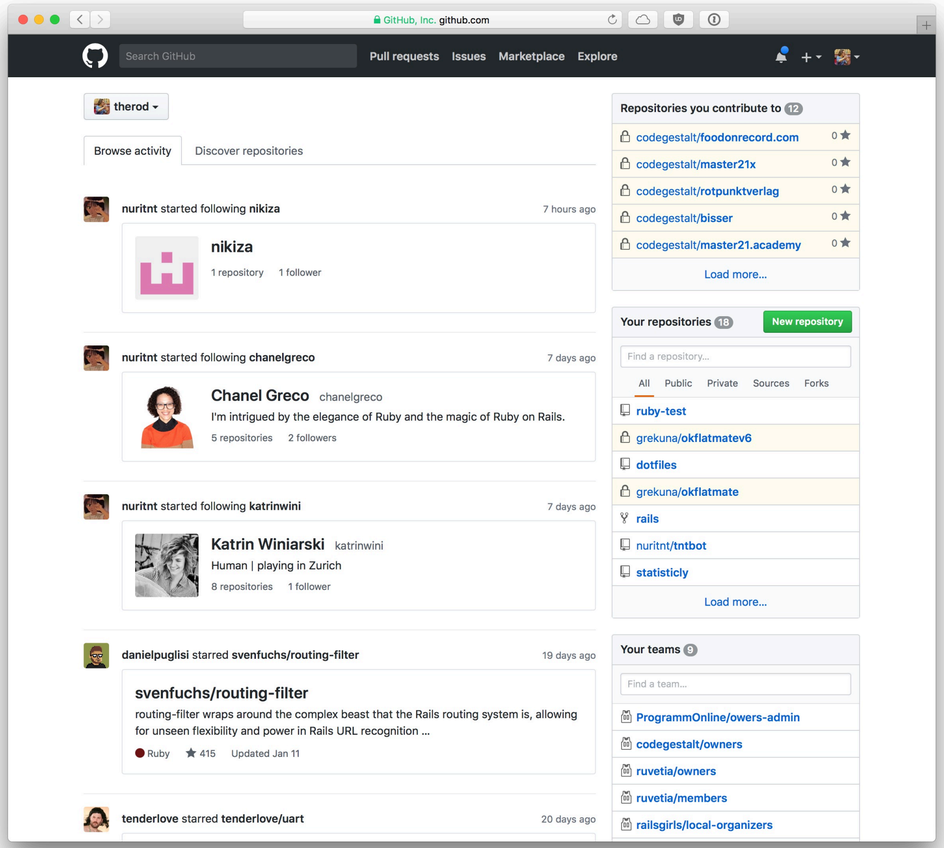
Use **irb** in a separate Terminal window to play around.
Type **exit** to close irb again.

Another tip

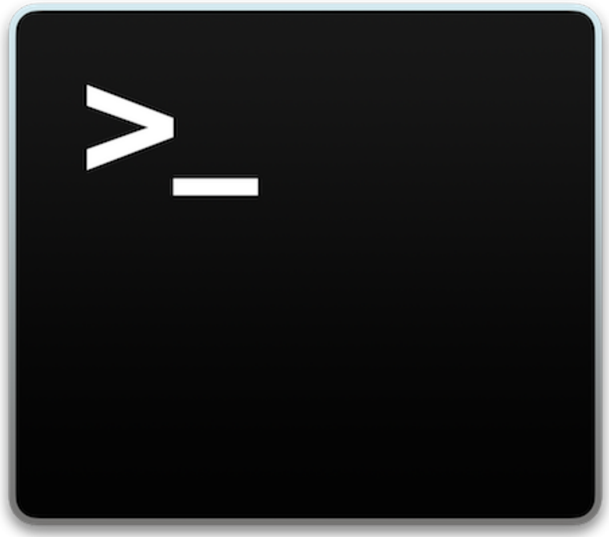


https://repl.it/languages/ruby

What can we use Ruby for?

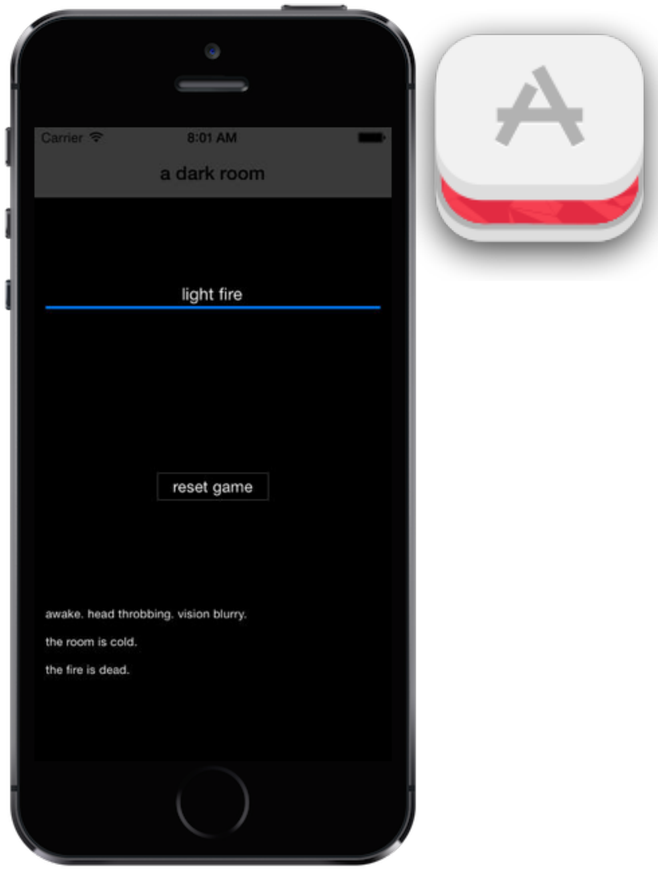


Web Applications



Shell Scripts

What can we use Ruby for?



http://www.rubymotion.com

http://shoesrb.com

Read this carefully

There are two milestones when you learn how to program.

The first milestone is **being able to read code that others have written and make sense of it**. Most step by step tutorials and books can help you reach this milestone in a short amount of time.

The second milestone is much harder to reach. **The ability to be confronted with a specific problem and then solve it on your own.** This means no step by step guide, no code that already exists – only what you have really learned so far can help here (and Google).

The sooner you move away from tutorials and start working on your own projects the better!

Real world examples that might pop up

- Backup System for a Customer
- OCR (Optical Character Recognition) tool for scanning PDFs
- News Site Crawler (Ecco)
- PDF name reader
- Viseca Crawler
- Creating and reading transactions on the Raiffeisen eBanking system
- Checking a ticket vendor page and sending a notification as soon as the tickets become available
- Image cropper for Rotpunktverlag
- Script to automatically cut video files

Writing software is hard

DHH Follow
Creator of Ruby on Rails, Founder & CTO at Basecamp (formerly 37signals), NYT Best-selling author of REWORK and REMOTE, and Le Mans class-winning racing driver.
Dec 27, 2016 · 5 min read

Writing software is hard

Top highlight

Good software is uncommon because writing it is hard. In the abstract, we all know that it is hard. We talk incessantly about how it's hard. And yet, we also collectively seem shocked—just shocked!—when the expectable happens and the software we're exposed to or is working on turns out poor.

Because consider the alternative: If I blame my tools or my process or my stakeholders or the full moon, I get to exonerate myself, and my ego, but I'm left with far less motivation to improve and very little insight into how. If I instead accept at least partial responsibility, there's a clear place to start improving.

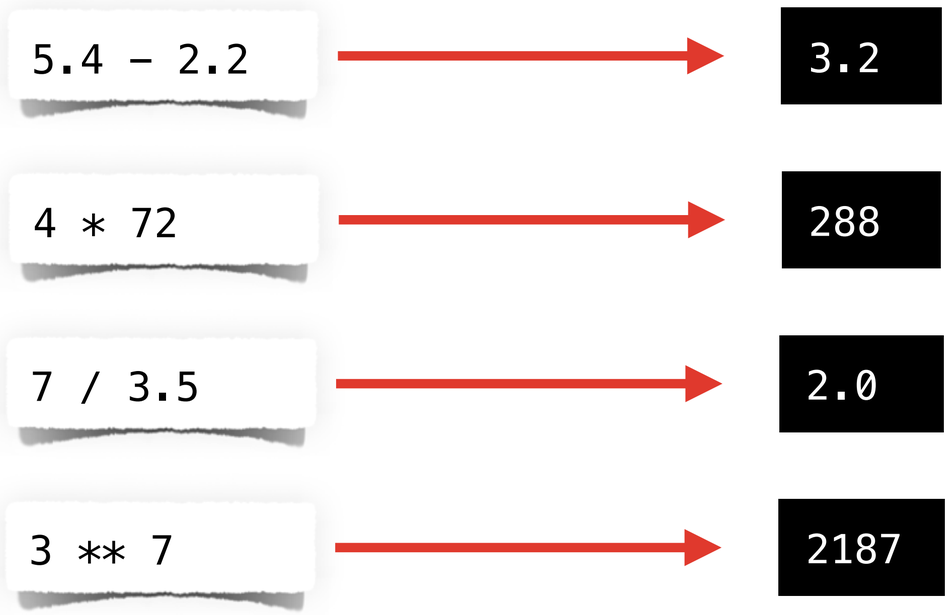
https://m.signalvnoise.com/writing-software-is-hard-388d5e982ad9

Basics I

Math, strings, variables, printing and getting input

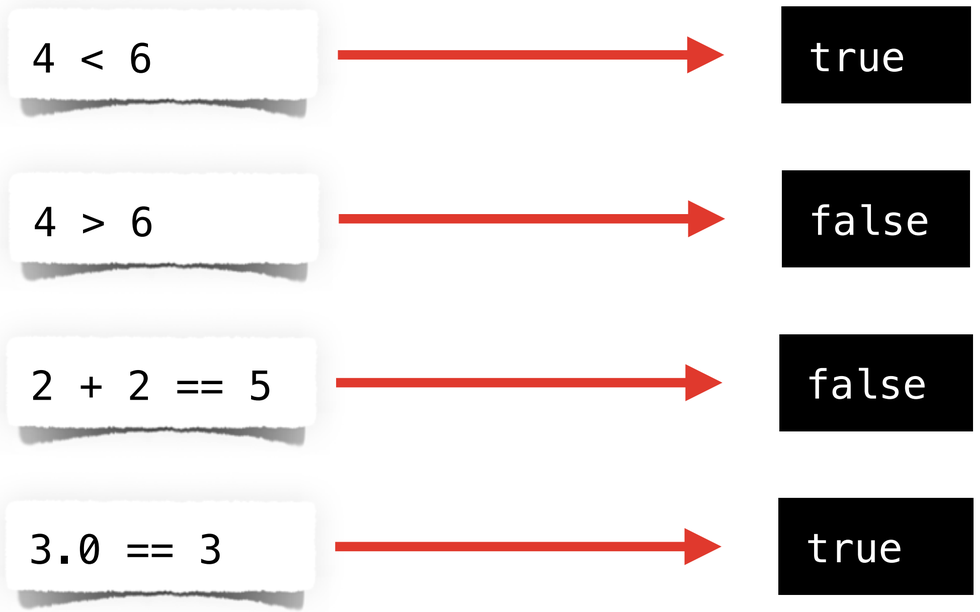
Basic math operations

The + symbol is for addition, - for subtraction, * for multiplication, / for division, and ** for exponentiation.



Comparing values

use < and > to compare two values and see if one is less than or greater than another. Use == (two equal signs) to see if the values are equal.



Strings

A **String** is a series of text characters. A string is either surrounded with double quotes (“) or single quotes (‘). Both work a little differently.

Printing on screen

We can display in the terminal by using either **puts** or **print**. By default **puts creates a new line at the end of the string** where print does not.

```
puts “Hello brave new world”
puts “I like Ruby!”
```

```
Hello brave new world
I like Ruby!
```

```
print “Hello ”
print “world!”
```

```
Hello world!
```

String interpolation

Use the #{ ... } notation inside a **double quoted string** to add Ruby code within the string.

```
puts “The answer is #{6 * 7}.”
```

```
The answer is 42.
```

```
puts ‘What is #{3 + 2}?’
```

```
What is #{3 + 2}?
```

Variables

Ruby lets us create **variables** – names that refer to values. You assign to a variable with the = symbol.

```
small = 8
medium = 13
puts small + medium
```

```
21
```

```
favorite_cake = “Cheese”
ok_cake = “Carrot”
puts ok_cake + “, “ + favorite_cake
```

```
Carrot, Cheese
```

Use all lowercase letters in variable names. Avoid numbers; they’re rarely used. Separate words with underscores.

Getting input from the user

The **gets.chomp** method read a line from standard input (characters typed in the terminal window.) When you call **gets** it halts the program until the users presses the Enter key.

```
print “Please enter your name:”
name = gets.chomp
puts “Nice to meet you, #{name}!”
```

```
Please enter your name: Rodrigo
Nice to meet you, Rodrigo!
```

Basics I - Demo 🤓

Math, strings, variables, printing and getting input

Basics II

Everything in Ruby is an object

Everything is an object!

Ruby is an *object-oriented* language (we'll learn what that means exactly in Part II). For now all you need to know that your data has useful **methods** attached directly to it.

Other cool methods that you can try out

```
"hEllO".downcase
```

```
hello
```

```
"Hi there".length
```

```
8
```

Everything in Ruby is an object. Even something as simple as numbers.

```
43.even?
```

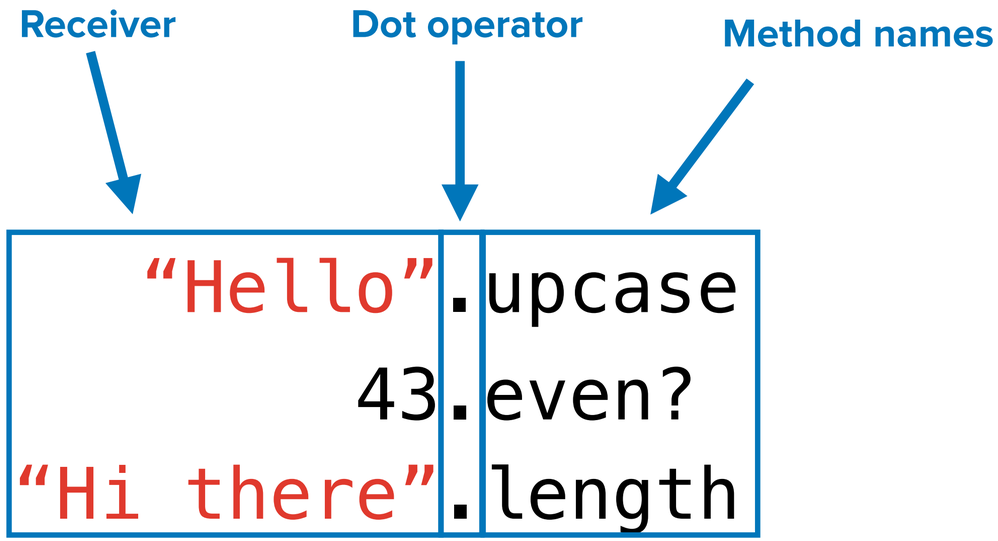
```
false
```

```
3.next
```

```
4
```

Calling a method on an object

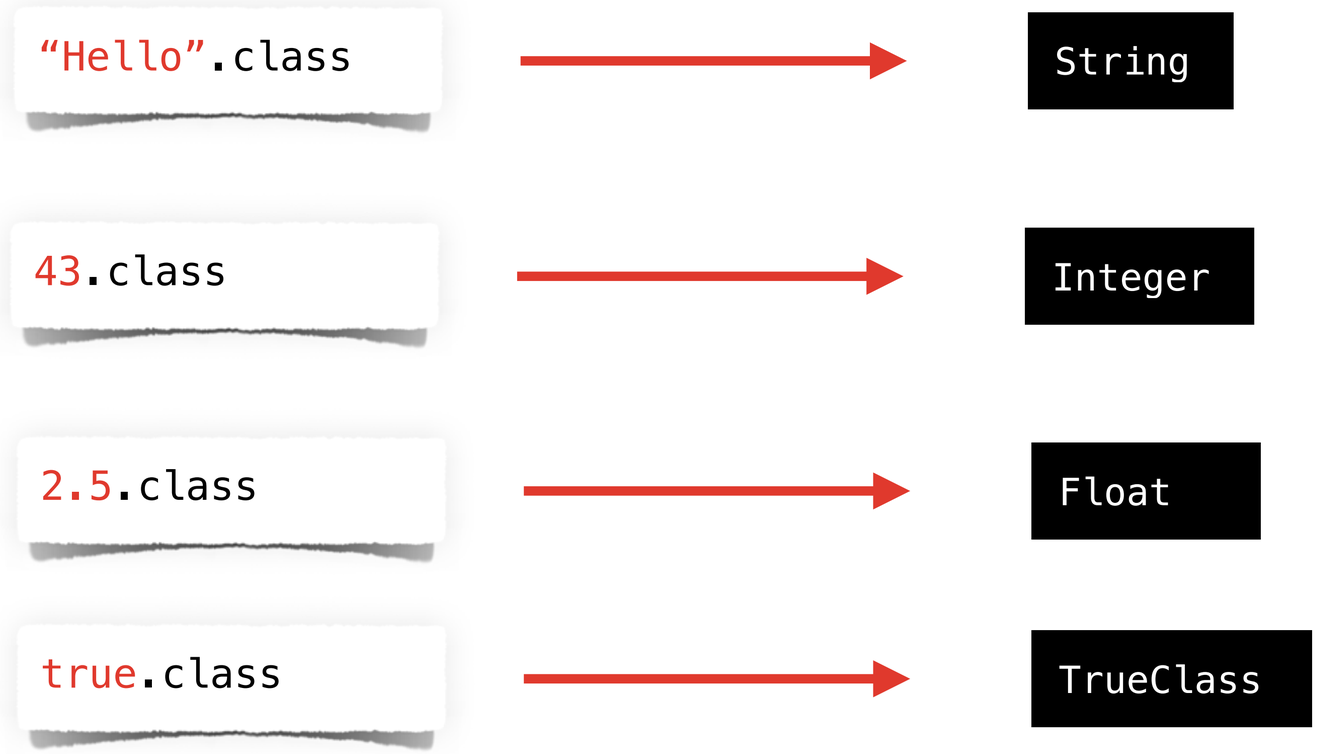
When programmer say **“I’m calling the ‘reverse’ method on my ‘Hello’ String”**. That’s exactly what they mean. The object you’re calling the method on is known as the method **receiver.** It’s whatever is to the left of the dot operator.



Think of it like passing a message to the object ***“Hey can you send me back an uppercase version of yourself?”***

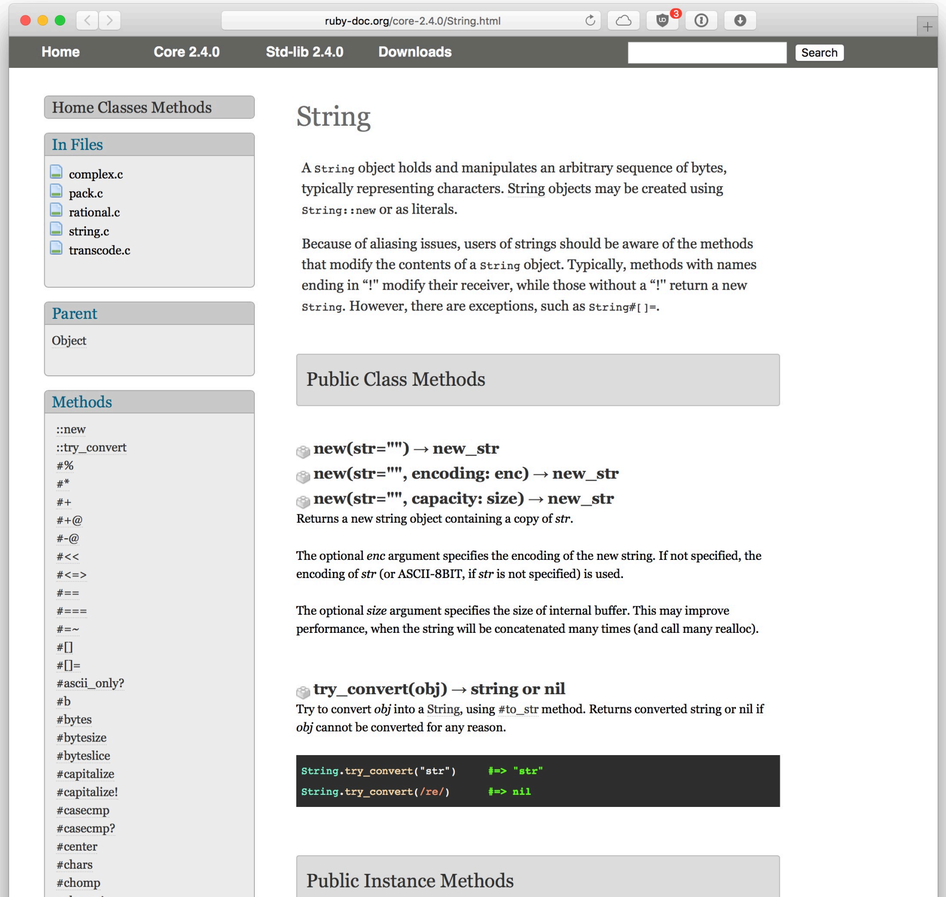
What object are we dealing with?

Some methods are unique to specific objects. But how do we find out what type of object we're dealing with?



Documentation

Now that we know what type of object we're dealing with we can go to **https://ruby-doc.org/core-2.4.0** and browse through the documentation!



empty? → true or false

Returns `true` if *str* has a length of zero.

```
"hello".empty?   #=> false
" ".empty?       #=> false
"".empty?        #=> true
```

end_with?([suffixes]+) → true or false

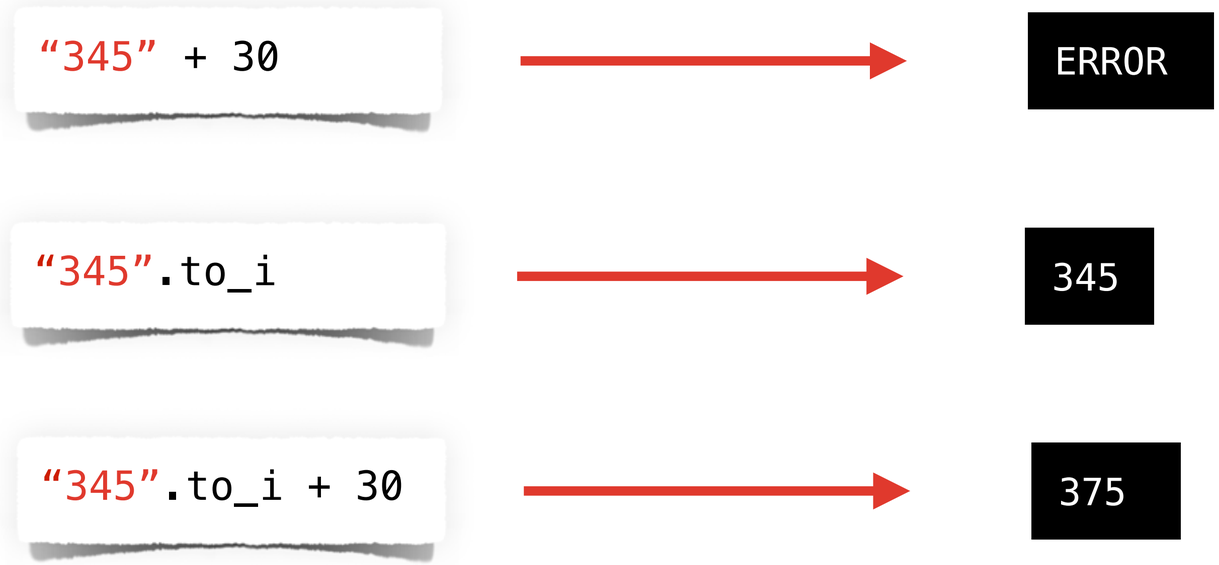
Returns true if `str` ends with one of the `suffixes` given.

```
"hello".end_with?("ello")                #=> true

# returns true if one of the +suffixes+ matches.
"hello".end_with?("heaven", "ello")      #=> true
"hello".end_with?("heaven", "paradise") #=> false
```

Sometimes we need to convert objects

We cannot do addition with strings. When we get data from the user using **gets.chomp** the input is delivered as a string - even if it's a number!



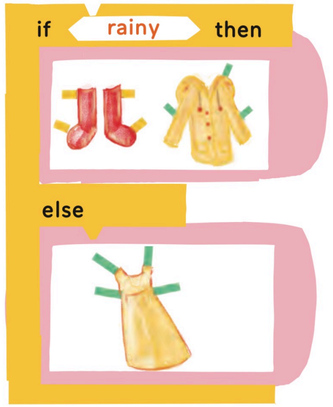
Basics II - Demo 🤓

Everything in Ruby is an object

Basics III

Conditionals

Conditionals



Like most languages, Ruby has **conditional** statements.
They cause code only to be executed if a **condition** is met.

Start of the conditional

Boolean expression

Block

```
if 1 < 2
  puts "It's true!"
end
```

```
It's true!
```

Conditional body

End of the conditional

Branching off

We can have multiple branches in the condition by using **if/elsif/else**.

```
score = 74

if score == 100
  puts "Perfect!"
elsif score >= 70
  puts "You pass!"
else
  puts "Oh boy, you're in trouble!"
end
```

```
You pass!
```

Every time you forget indentation a kitten dies!

The opposite of “if” is “unless”

```ruby
person_works_here = false

if not person_works_here
  puts "Hey, you are not allowed to be in here!"
end
```

```ruby
person_works_here = false

unless person_works_here
  puts "Hey, you are not allowed to be in here!"
end
```

Basics III - Demo 🤓

Conditionals

Basics IV

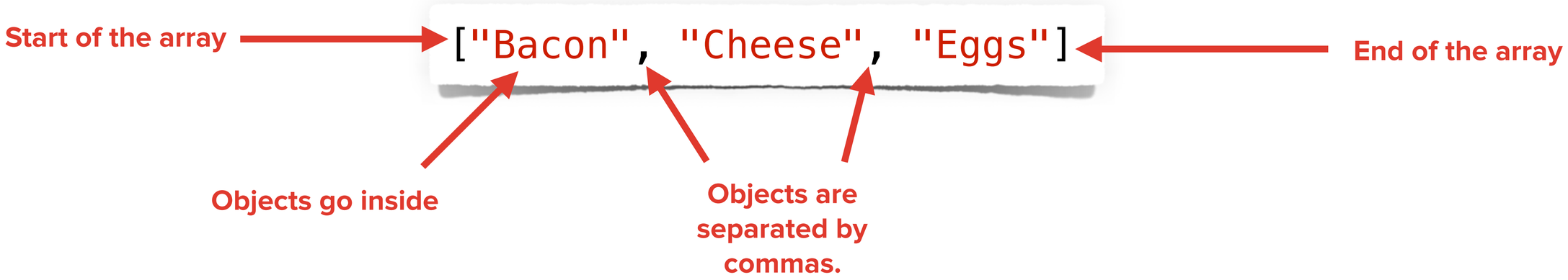
Arrays and looping over things

Arrays

Arrays are used to hold a collection of objects. It can hold objects of any type.

```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast
```

```
Bacon
Cheese
Eggs
```



Accessing arrays

Items in an array are numbered from left to right, starting with 0. This is called the array **index**.

```
["Bacon", "Cheese", "Eggs"]
```


```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast[1]
```

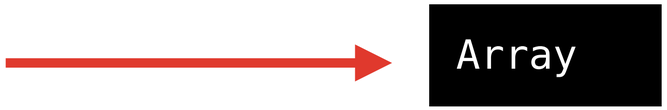
```
Cheese
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
breakfast[1] = “Sardines"
puts breakfast
```

```
Bacon
Sardines
Eggs
```

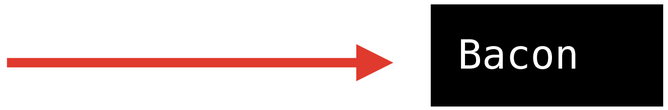
Arrays are objects, too!

```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast.class
```



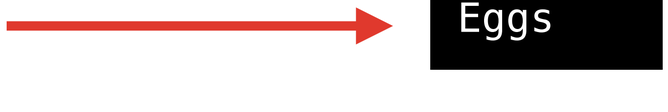
```
Array
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast.first
```



```
Bacon
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast.last
```



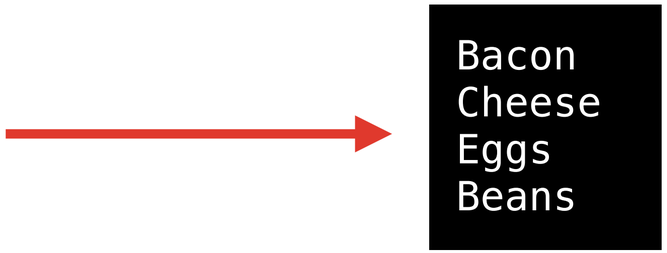
```
Eggs
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
puts breakfast.include?(“Ham")
```

```
false
```

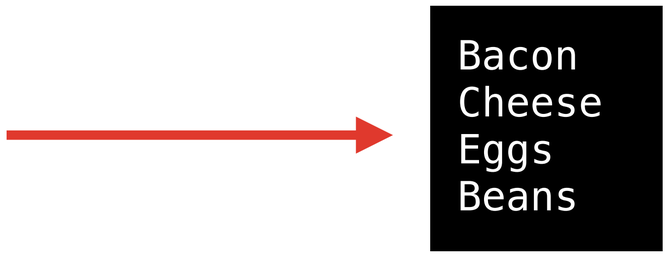
Arrays are objects, too!

```
breakfast = ["Bacon", "Cheese", "Eggs"]
breakfast.push(“Beans”)
puts breakfast
```



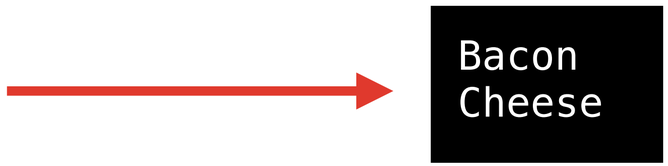
```
Bacon
Cheese
Eggs
Beans
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
breakfast << “Beans”
puts breakfast
```



```
Bacon
Cheese
Eggs
Beans
```

```
breakfast = ["Bacon", "Cheese", "Eggs"]
breakfast.pop
puts breakfast
```



```
Bacon
Cheese
```

http://ruby-doc.org/core-2.4.1/Array.html

Looping over the items

The most common used method to loop over array items is the **.each** method.

```ruby
breakfast = ["Bacon", "Cheese", "Eggs"]

breakfast.each do |food|
  puts "I'm taking #{food} and put in my mouth, yummy!"
end
```

```
I'm taking Bacon and put in my mouth, yummy!
I'm taking Cheese and put in my mouth, yummy!
I'm taking Eggs and put in my mouth, yummy!
```

```ruby
breakfast = ["Bacon", "Cheese", "Eggs"]

breakfast.each do |x|
  puts "I'm taking #{x} and put in my mouth, yummy!"
end
```

```
I'm taking Bacon and put in my mouth, yummy!
I'm taking Cheese and put in my mouth, yummy!
I'm taking Eggs and put in my mouth, yummy!
```

Looping over the items (short)

There is another way to write the same loop in one line.

```ruby
breakfast = ["Bacon", "Cheese", "Eggs"]

breakfast.each do |food|
  puts "I'm taking #{food} and put in my mouth, yummy!"
end
```

```
I'm taking Bacon and put in my mouth, yummy!
I'm taking Cheese and put in my mouth, yummy!
I'm taking Eggs and put in my mouth, yummy!
```

```ruby
breakfast = ["Bacon", "Cheese", "Eggs"]

breakfast.each { |food| puts "I'm taking #{food} and put in my mouth, yummy!" }
```

In general the first variation is preferred. **https://github.com/bbatsov/ruby-style-guide**

Another variation to loop

In most other programming languages you would write the loop like this.

```
breakfast = ["Bacon", "Cheese", "Eggs"]

index = 0
while index < breakfast.length
  puts "I'm taking #{breakfast[index]} and put in my mouth, yummy!"
  index += 1
end
```

Compare this to how we've written the loop so far. Which variant do you prefer?

```
breakfast = ["Bacon", "Cheese", "Eggs"]

breakfast.each do |food|
  puts "I'm taking #{food} and put in my mouth, yummy!"
end
```

Basics IV - Demo 🤓

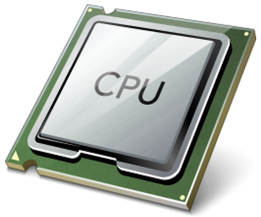
Arrays and looping over things

Basics V

Packing things up into functions

What are functions?

INPUT



OUTPUT

A **function** in programming is a set of expressions that returns a value. With functions, you can organize your code into “subroutines” that can be easily called from other areas in your program.

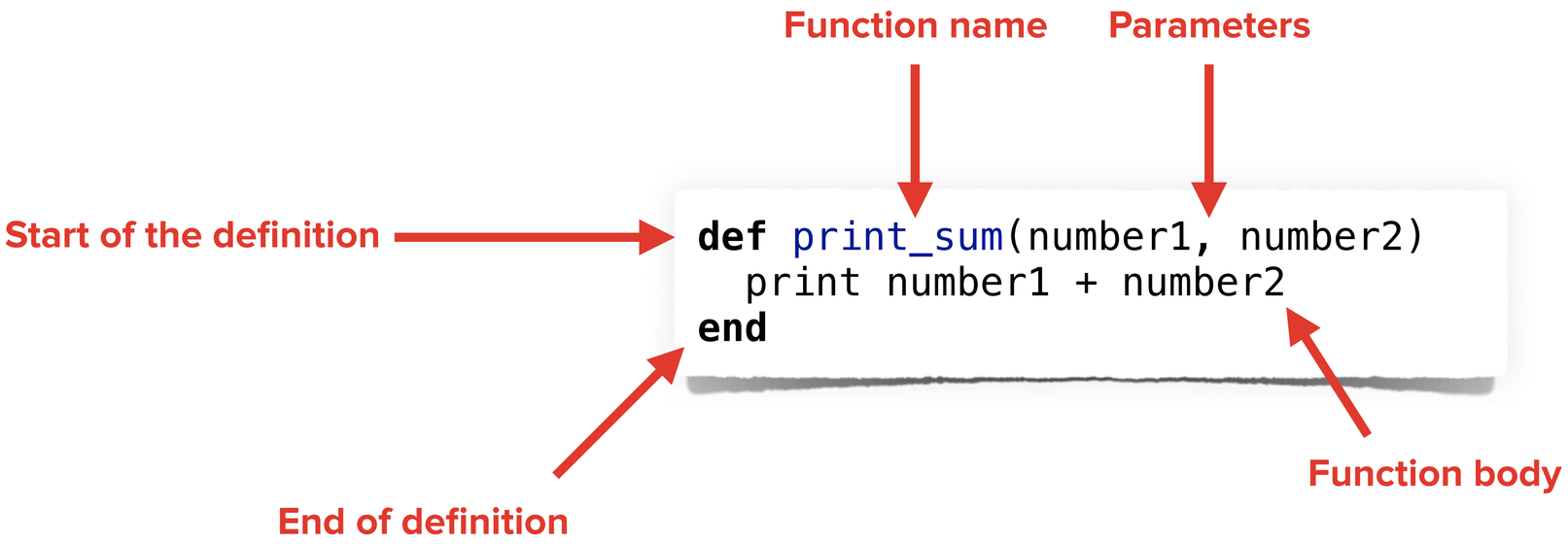
PARAMETER(S) (optional)

FUNCTION

RETURN VALUE

Defining functions

A function definition looks something like this in Ruby:



An example

A simple function without any parameters.

```ruby
def accelerate
  puts "stepping on the gas"
  puts "speeding up"
  puts "Vroooooooom!"
end

accelerate
```

```
stepping on the gas
speeding up
Vroooooooom!
```

How the same function could look like with parameters.

```ruby
def accelerate(speed)
  puts "stepping on the gas"
  puts "speeding up, now at #{speed} km/h!"
  puts "Vroooooooom!"
end

accelerate(130)
```

```
stepping on the gas
speeding up, now at 130 km/h!
Vroooooooom!
```

Functions are for multiple use!

The main idea behind functions is to **encapsulate specific behavior** into a block of code **so that you can use it over and over again** in your program.

```ruby
def accelerate(speed)
  puts "stepping on the gas"
  puts "speeding up, now at #{speed} km/h!"
  puts "Vroooooooom!"
end

accelerate(130)
accelerate(60)
accelerate(10)
```

```
stepping on the gas
speeding up, now at 130 km/h!
Vroooooooom!

stepping on the gas
speeding up, now at 60 km/h!
Vroooooooom!

stepping on the gas
speeding up, now at 10 km/h!
Vroooooooom!
```

Optional parameters

Optional parameters are great to define default behavior. That way you don't have to pass in a parameter if you don't need to.

```ruby
def accelerate(speed = 40)
  puts "stepping on the gas"
  puts "speeding up, now at #{speed} km/h!"
  puts "Vroooooooom!"
end

accelerate
accelerate(60)
accelerate
```

```
stepping on the gas
speeding up, now at 40 km/h!
Vroooooooom!

stepping on the gas
speeding up, now at 60 km/h!
Vroooooooom!

stepping on the gas
speeding up, now at 40 km/h!
Vroooooooom!
```

Every function returns something

We learned that parameters can be completely optional. But each function has to return something in the end. Let's look at an example.

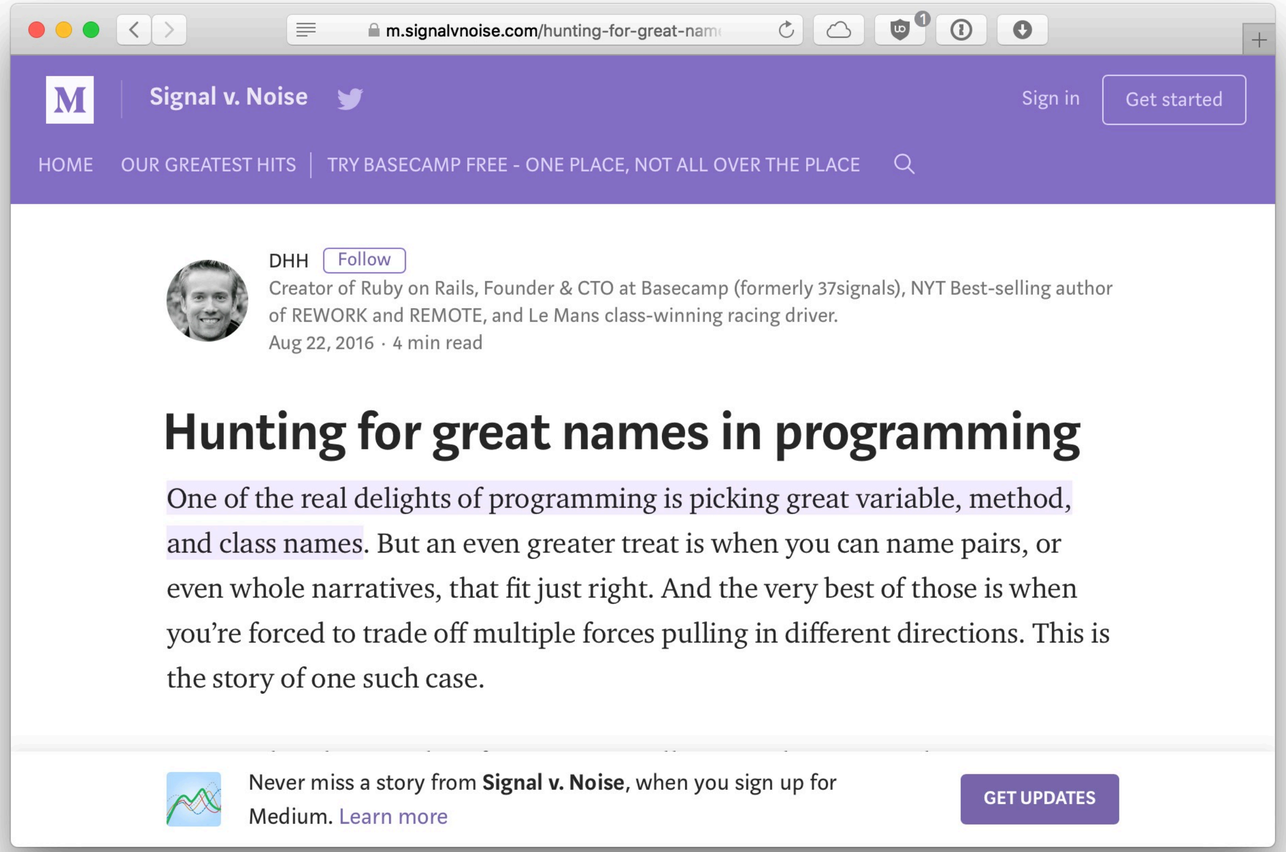
```ruby
def calculate_bmi(weight, height)
  (weight / height) / height
end

puts calculate_bmi(75, 1.80)
puts calculate_bmi(180, 1.70)
```

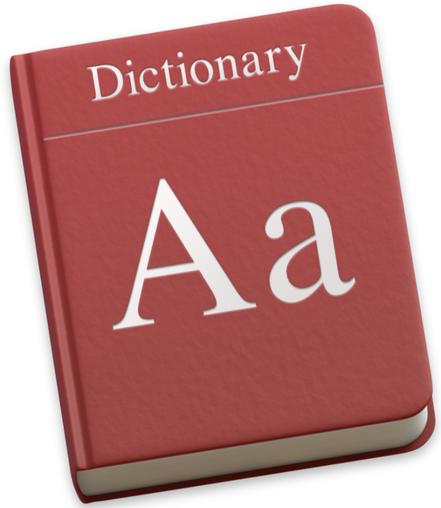
```
23.148148148148145
62.28373702422146
```

You don't actually need the **return** keyword in the above function. The value of the last expression becomes that function's return value.

A word on naming functions



> 'There are only two hard things in Computer Science: cache invalidation and naming things.'
>
> - Phil Karlton



Keep this in your Dock!

https://m.signalvnoise.com/hunting-for-great-names-in-programming-16f624c8fc03

Example

```ruby
def bmi(w, h)
  (w / h) / h
end

puts bmi(75, 1.80)
puts bmi(180, 1.70)
```

```ruby
def calculate_bmi(weight, height)
  (weight / height) / height
end

puts calculate_bmi(75, 1.80)
puts calculate_bmi(180, 1.70)
```

Somebody might need to read your code in the future (sometimes it is yourself) - always try to remember that.

Basics V - Demo 🤓

Packing things up into functions

Basics VI

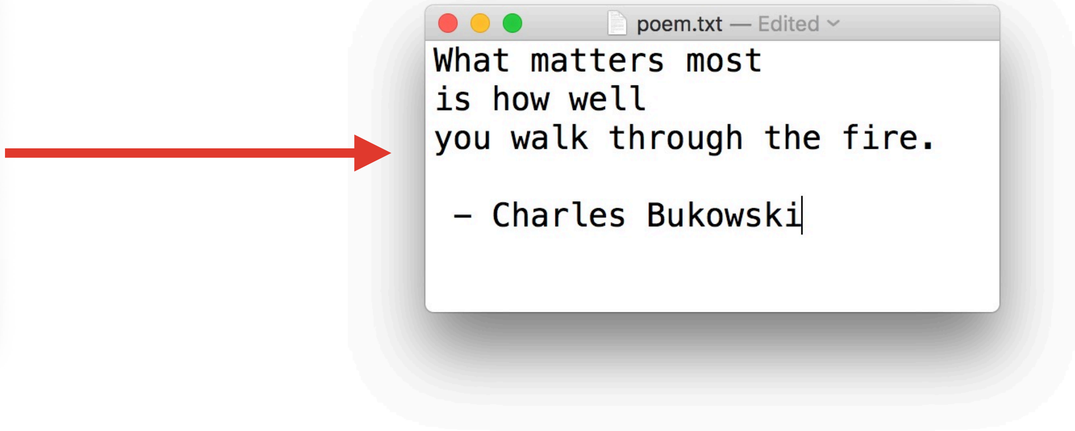
Reading and writing text files

Create a new file and write to it

The simplest way to write some text to a file is by using the **File** class.

```
file = File.new("poem.txt", "w")

file.write("What matters most\n")
file.write("is how well\n")
file.write("you walk through the fire.\n")
file.write("\n")
file.write(" - Charles Bukowski”)
file.close
```



Read a file

To read from an existing file you use **File.read**

```
text = File.read("poem.txt")
puts text
```

```
What matters most
is how well
you walk through the fire.

 - Charles Bukowski
```

Basics VI - Demo 🤓

Reading and writing text files