

wifi: Itnig HQ / oficinaitnig

Ruby ルビー

a programmer's best friend

About me

Albert Bellonch



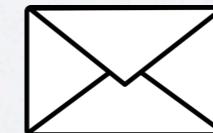
Quipu CTO



@abellonch



/albertbellonch



albert@itnig.net

About the **workshop**

- Two main blocks
 - From **10am** to **2pm**
 - From **3pm** to **7pm**
- We will combine **theory** and **practice** and **theory** and **practice** and **theory** and **practice**
- Please, **interrupt me at any moment**
- And have fun too

About you

Name

Background

Languages you know

Why Ruby?

My goal

```
people.each(&:learn_some_ruby!)
```

Some setup

Some setup

- Clone and download the following git repo to your computer:
 - **github.com/albertbellonch/basic-ruby**
- Install Ruby
 - **RVM/rbenv** for Mac OS X / UNIX
 - **RubyInstaller** for Windows

**A bit of an
introduction**



Creation

- Created by Japanese computer scientist and software programmer **Yukihiro Matsumoto** in **1993**
- Focused on balancing:
 - **Functional** programming (Haskell, Clojure, Erlang)
 - **Imperative** programming (FORTRAN, Basic, C, Java)
- Takes the **name** after an online chat session with a friend

Conception

- Conceived to be **powerful** and yet **fun** to code with
- It is a high level programming language, focused on **solving human needs** instead of computer's
- Tends to **minimize confusion** for users
- It **optimizes** the programmer's work

Evolution

- First publication of version **0.95** came in late **1995**
- Following versions were released every 1-2 years: **1.2** (1998), **1.4** (1999), **1.6** (2000), **1.8** (2003)
- Ruby **1.8** persisted until this summer (deprecated!)
- Ruby **1.9** was released in **2007**, with many changes over 1.8, being the most used version of the language
- Ruby **2.0** was released this February, **20 years** after it was born and adding some more new features

Implementations

- There are several implementations of the language
 - **MRI** is the *de-facto* one
 - **YARV** is starting to replace it in newer versions
 - **Rubinius**, partially implemented in Ruby
 - **JRuby**, implemented in the JVM
 - **RubyMotion**, created to be run on iOS/OS X
 - **MacRuby**, which runs Objective-C and uses LLVM
 - ...and many many more.
- There is no standard specification of it, although the differences between implementations' features are minimal

Exercices time!

- What you like from the languages you have used
- What you don't like, or will improve

Setup

- For **Mac OS X** or **UNIX**
 - **RVM** (rvm.io), maybe the most used
 - **rbenv** (github.com/sstephenson/rbenv) also recommended
 - Both allow having several language versions, and switching between them
- For **Windows**
 - **RubyInstaller for Windows** (rubyinstaller.org), only installs a version of the language
- There are alternative, not-that-recommended ways

Gems

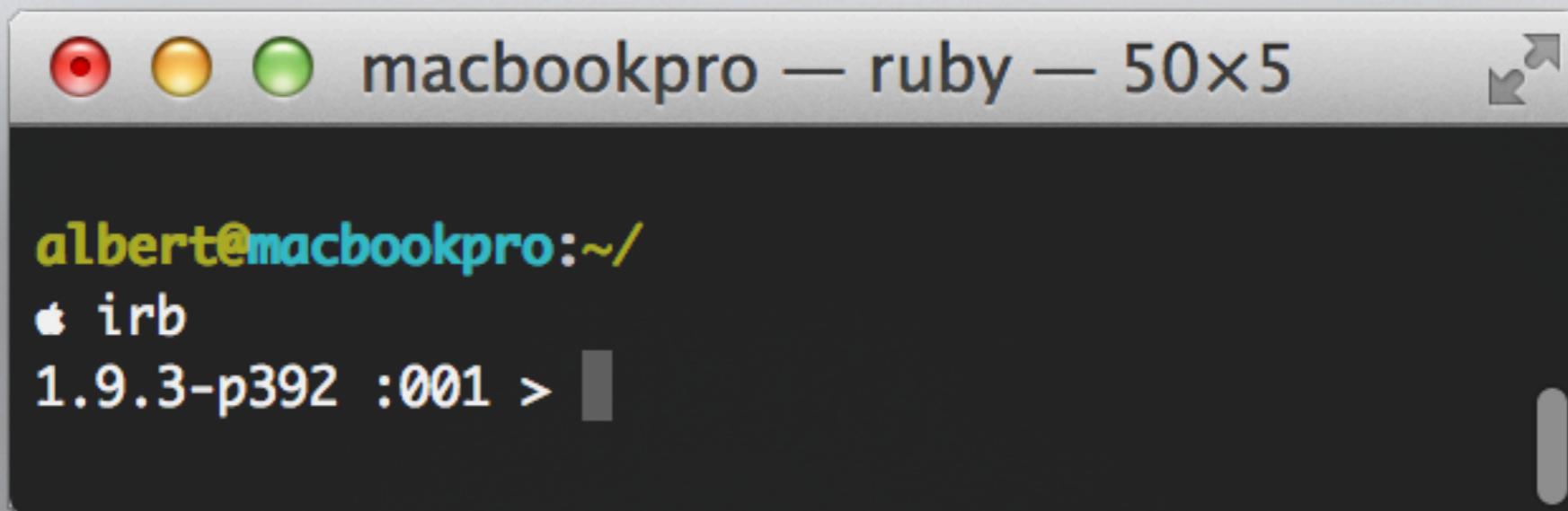
- Gems are packages of code encapsulated that you can easily add to your projects
- Are managed through **RubyGems** (rubygems.org)
- Many are hosted on **Github**, which is great to inspect the code, or fork it to improve it
- As of this week, **64.138 gems** with **2.1G** downloads

Present

- Really active community, evolving the language and its gems in a daily basis
- Very popular these last years due to the creation of Ruby on Rails, conceived in 2005 by Danish programmer David Heinemeier Hansson
- Lots of conferences are held every year: RubyConf, EuRuKo, RuPy, Baruco, and a lot more

Let's go get hands on

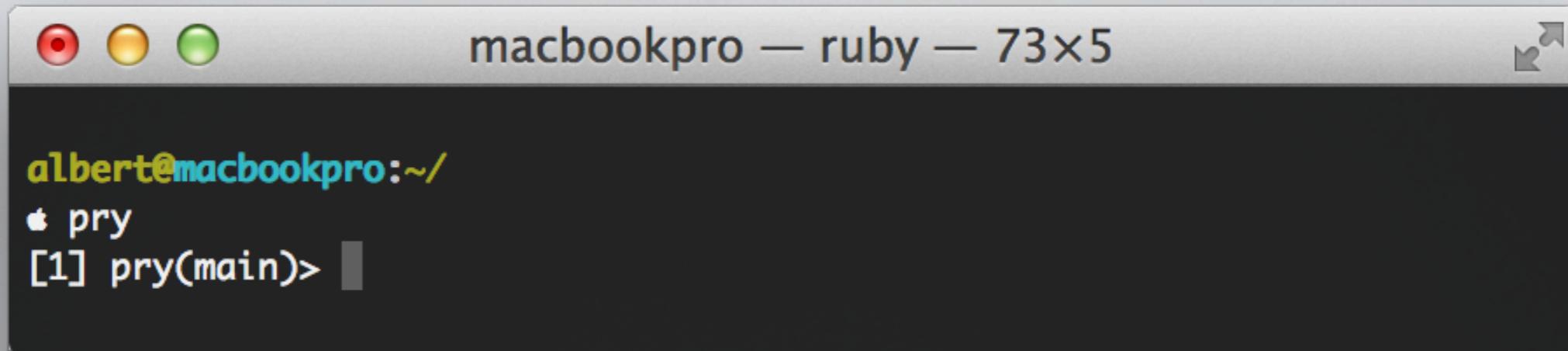
Your friend, the **Interactive Ruby Shell (irb)**



Try it!

Meet **pry** too

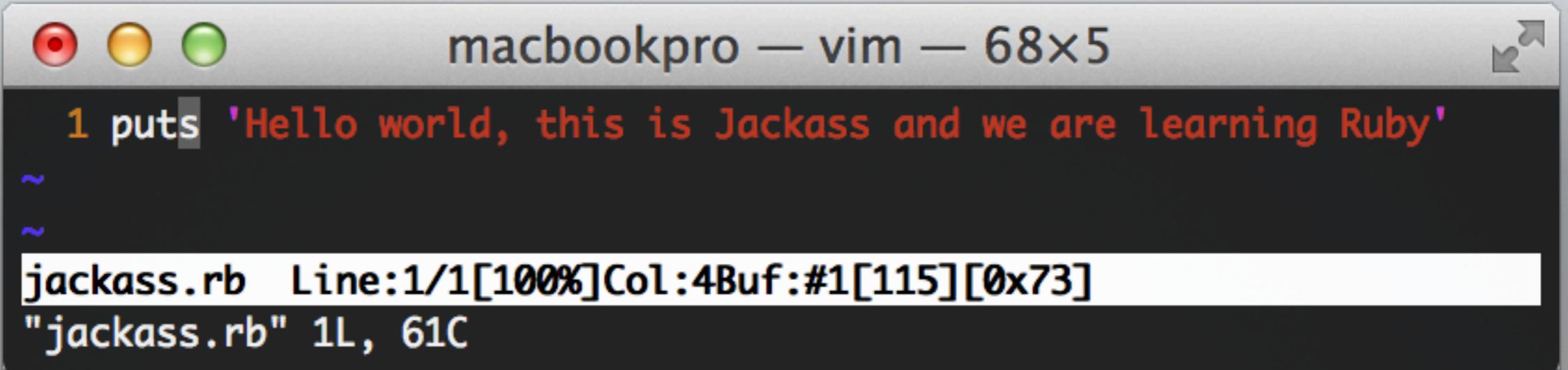
An alternative to the standard Ruby shell



The screenshot shows a Mac OS X terminal window with a dark theme. The window title is "macbookpro — ruby — 73x5". The terminal prompt is "albert@macbookpro:~/". Below the prompt, the text "➔ pry" is followed by "[1] pry(main)>". The terminal has the classic red, yellow, and green window control buttons at the top left and a close button at the top right.

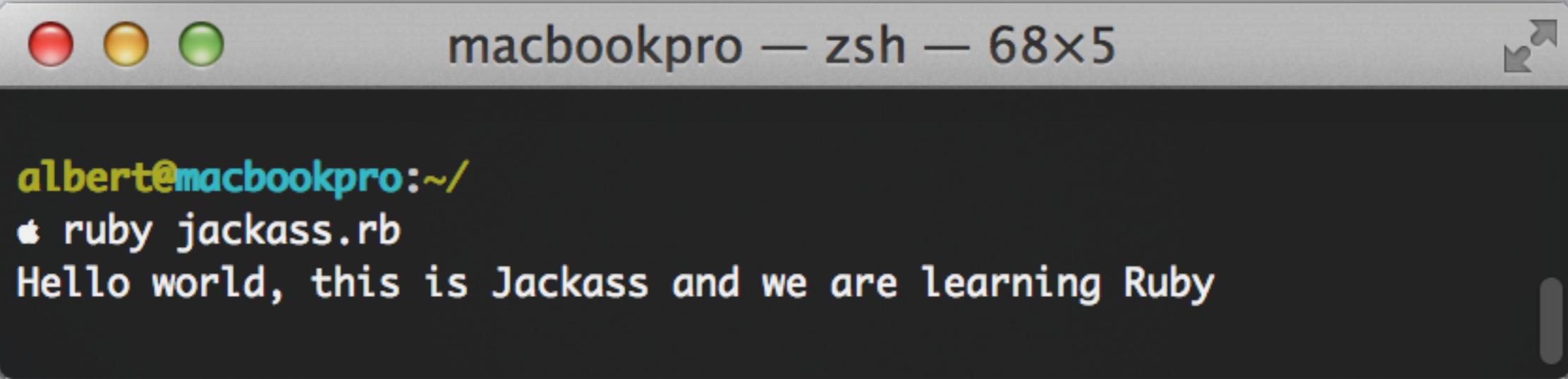
And give it a shot!

Executing some Ruby



macbookpro — vim — 68x5

```
1 puts 'Hello world, this is Jackass and we are learning Ruby'  
~  
~  
jackass.rb  Line:1/1[100%]Col:4Buf:#1[115][0x73]  
"jackass.rb" 1L, 61C
```



macbookpro — zsh — 68x5

```
albert@macbookpro:~/  
$ ruby jackass.rb  
Hello world, this is Jackass and we are learning Ruby
```

Executing more Ruby

macbookpro — vim — 69x6

```
1 puts 'Give me your name, please: '
2 name = gets.chomp
3 puts "I don't like your name, #{name}."  
~
```

name.rb Line:1/3[33%]Col:1Buf:#1[112][0x70]
"name.rb" 3L, 94C

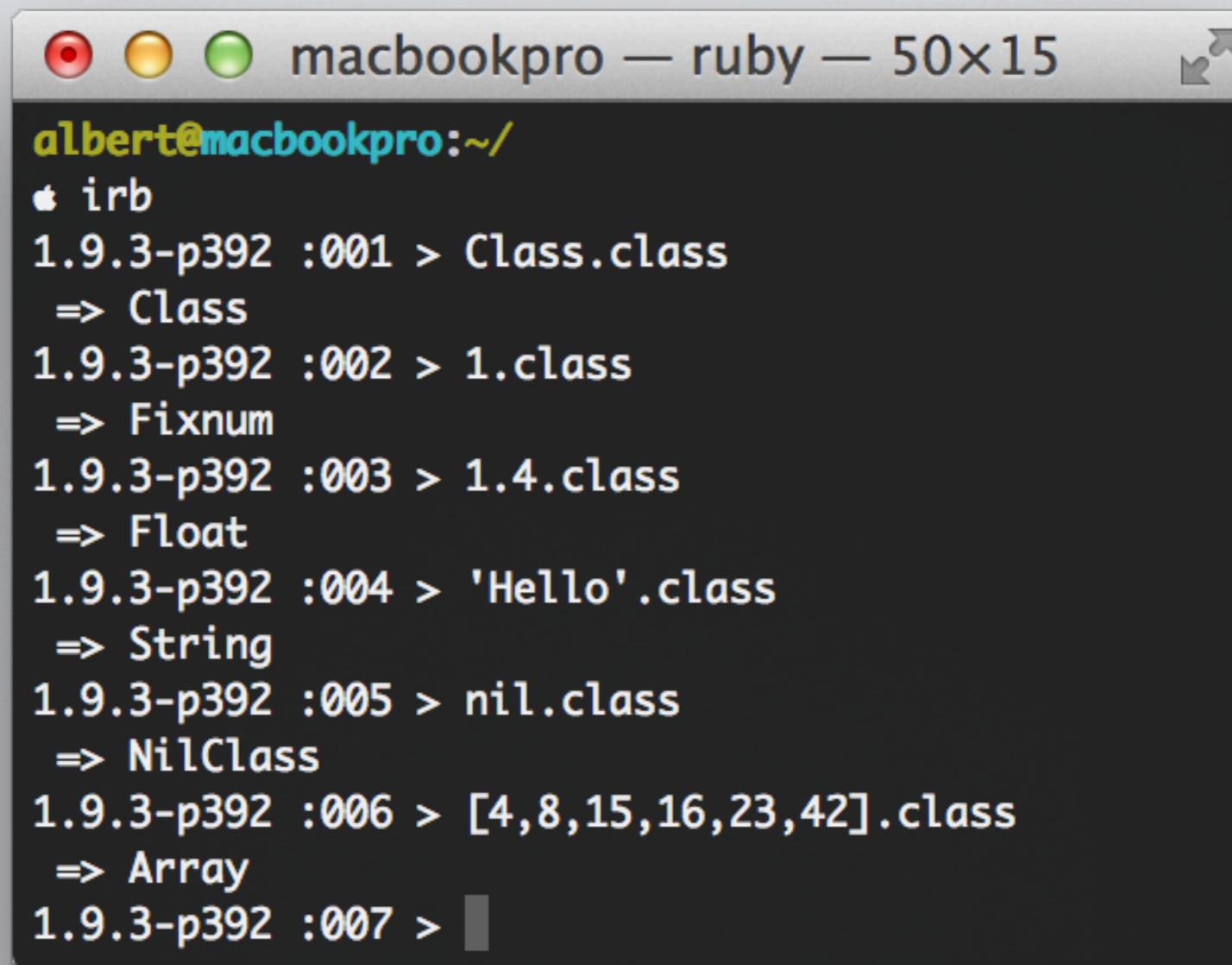
macbookpro — zsh — 69x6

```
albert@macbookpro:~/  
$ ruby name.rb
Give me your name, please:
Ruby
I don't like your name, Ruby.
```

Ruby **basics**

Essentials

- Everything in Ruby is an object



```
albert@macbookpro:~/  
• irb  
1.9.3-p392 :001 > Class.class  
=> Class  
1.9.3-p392 :002 > 1.class  
=> Fixnum  
1.9.3-p392 :003 > 1.4.class  
=> Float  
1.9.3-p392 :004 > 'Hello'.class  
=> String  
1.9.3-p392 :005 > nil.class  
=> NilClass  
1.9.3-p392 :006 > [4,8,15,16,23,42].class  
=> Array  
1.9.3-p392 :007 > |
```

...and everything returned is an object too

Essentials

- Objects are typically created via the **new** method on the class, returning an instance of that class
- Within each class, **instance methods** can be defined, which use the context of that specific instance
 - **e.g.** the brand of a specific car
- **Class methods** can be defined too, which apply to the class without any context
 - **e.g.** the list of all the cars for a specific brand

Essentials

- **Methods are invoked** by sending a message to a class instance (a.k.a. calling a method)
- Invoking a method requires a method **name** and some optional **parameters** *
- Methods are defined using the keyword **def**, followed by the name of the method, the parameters of the method, and the keyword **end**

Essentials

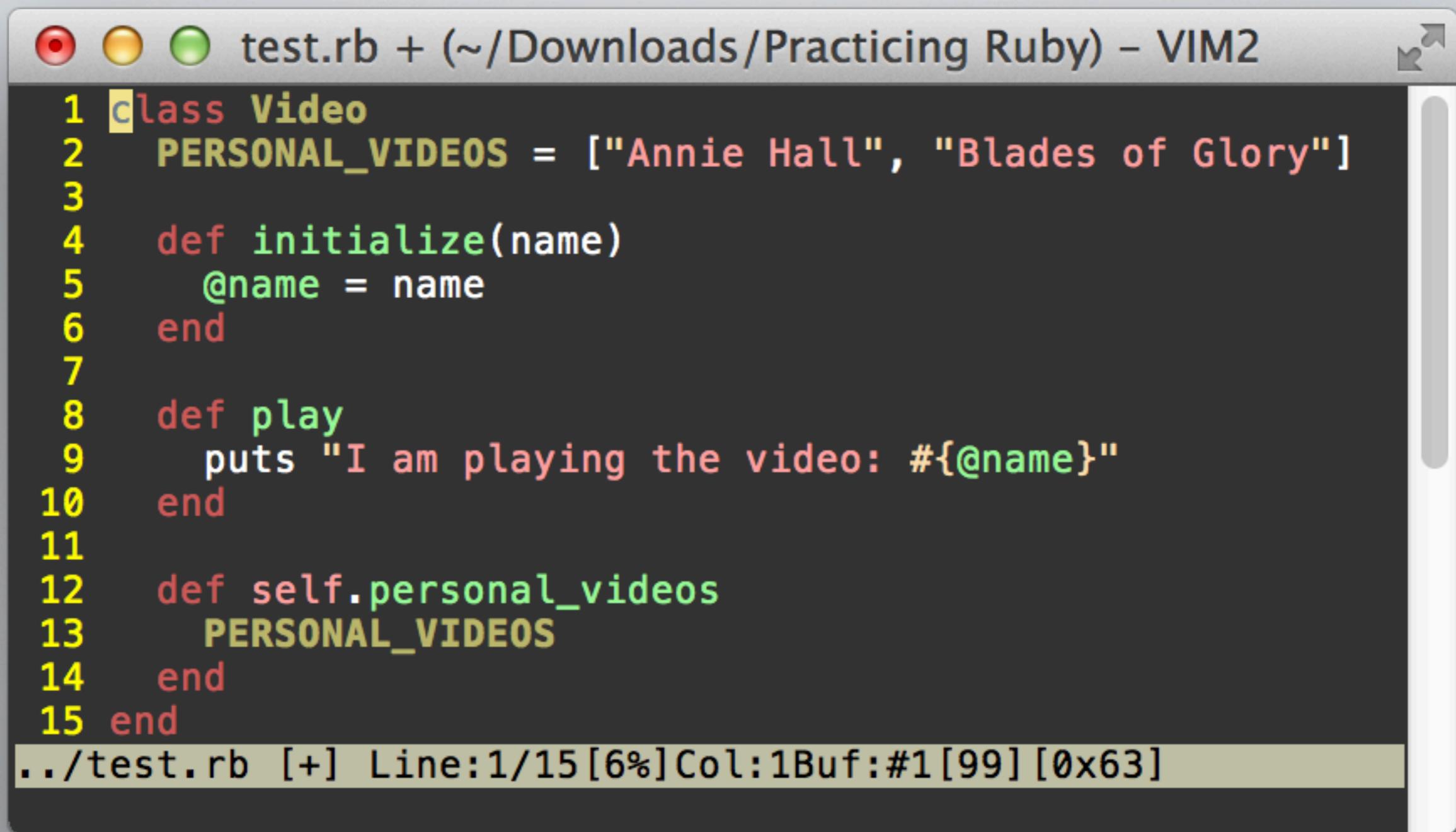
- Comments start with a **#**, and it works until the end of the line
- Indentation doesn't matter (**please** use two spaces)
- Classes are defined with **def**, the capitalized class name, the whole class body, and the **end** keyword
- Methods return the results with the **return** keyword
- The **initialize** method is used to pass some data when instanciating a class

Essentials

- Naming
- **local variables:** cat, dog, an_object, this_workshop
- **instance variables:** @x, @pony, @some_data
- **class variables:** @@total, @@some_sum, @@god
- **global variables:** \$do_this, \$is_winter, \$hello
- **class names:** String, Panda, Object, Class, Fixnum
- **constant names:** IVA, PLANCK_CONSTANT

Essentials

- A class example



A screenshot of a terminal window titled "test.rb + (~/Downloads/Practicing Ruby) - VIM2". The window shows the following Ruby code:

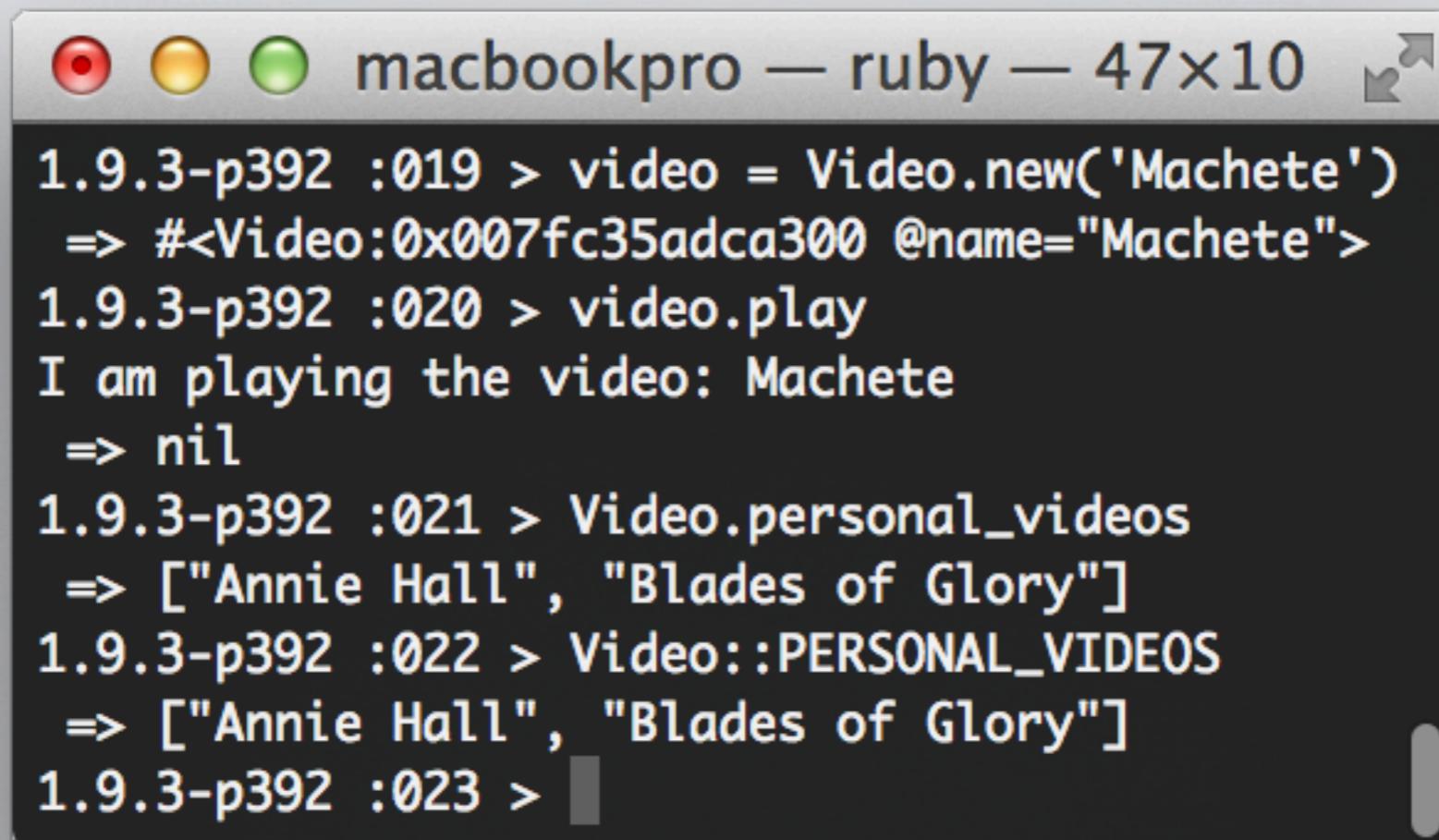
```
1 class Video
2   PERSONAL_VIDEOS = ["Annie Hall", "Blades of Glory"]
3
4   def initialize(name)
5     @name = name
6   end
7
8   def play
9     puts "I am playing the video: #{@name}"
10  end
11
12  def self.personal_videos
13    PERSONAL_VIDEOS
14  end
15 end
```

The code defines a class named "Video" with a class variable "PERSONAL_VIDEOS" containing two movie titles. It includes methods for initializing a video with a name and for playing it. There is also a class method "personal_videos" that returns the array of personal videos.

At the bottom of the terminal window, the status bar displays: ".../test.rb [+] Line:1/15 [6%] Col:1 Buf:#1 [99] [0x63]".

Essentials

- A class example



```
macbookpro — ruby — 47x10
1.9.3-p392 :019 > video = Video.new('Machete')
=> #<Video:0x007fc35adca300 @name="Machete">
1.9.3-p392 :020 > video.play
I am playing the video: Machete
=> nil
1.9.3-p392 :021 > Video.personal_videos
=> ["Annie Hall", "Blades of Glory"]
1.9.3-p392 :022 > Video::PERSONAL_VIDEOS
=> ["Annie Hall", "Blades of Glory"]
1.9.3-p392 :023 >
```

Exercices time!

Exercises I-2 from essentials

Arrays and hashes

- Indexed collections that are accessible using a key
- Arrays
 - The key is the integer
 - Can be initialized with an array literal: [1, 2, 'hey']
 - Access is done via square brackets:
[1, 2, 3, 5, nil, 4][1] => 3
 - There are some specific other ways to declare them:
%w{ this contains five strings }

Arrays and hashes

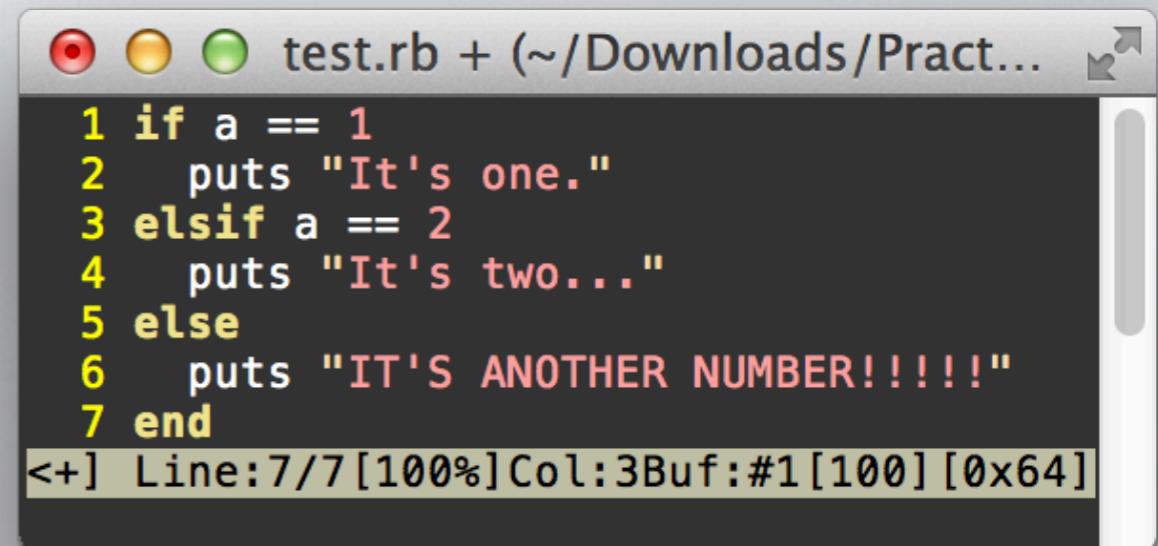
- Hashes
 - The key is any object
 - Can be initialized with hash literals:
`{ “Ferrari” => “Italy”, “Citroën” => “France”, “Ford” => “USA” }`
 - Accessed too via square brackets:
`brand_countries[‘Ford’] => ‘USA’`
 - We’ll see them more in detail later!

Symbols

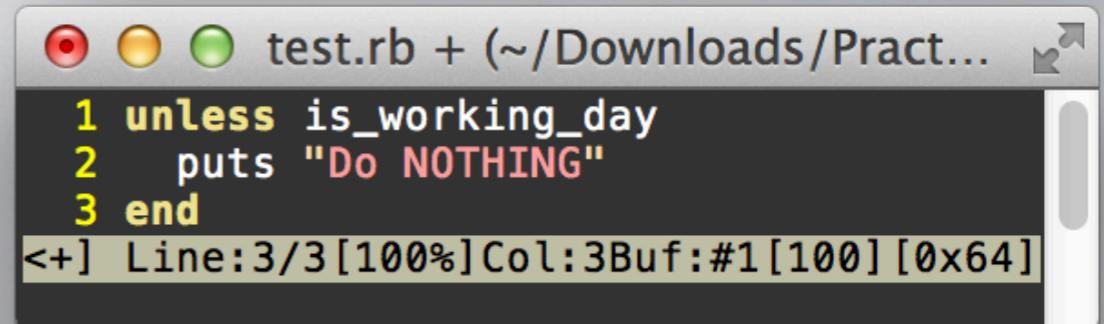
- Constant names that do not have to be declared previously, and that are unique
- Written preceded with a colon: `:workshop`
- Normally used as the strings in hashes:
`{ :a_random_number => 224, :one => 1 }`
- Used in Ruby 1.9 hash syntax (which supports the old)
`{ a_random_number: 224, one: 1 }`
- I prefer the latter syntax!

Control structures: if & else

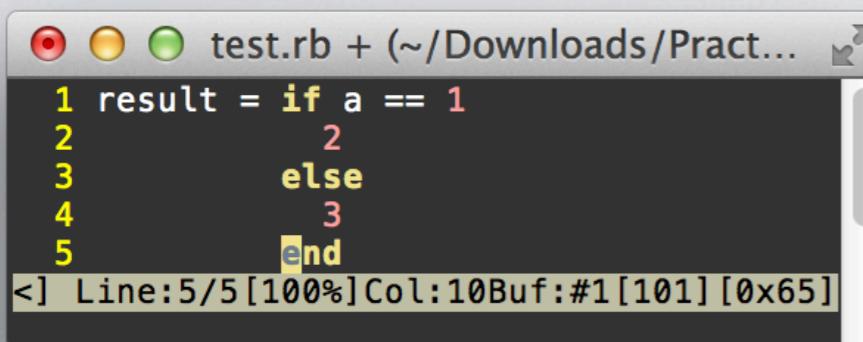
- The conditional structures are kind of standard
- We also have unless...end
- They can be inline
- They return the final line of the branch



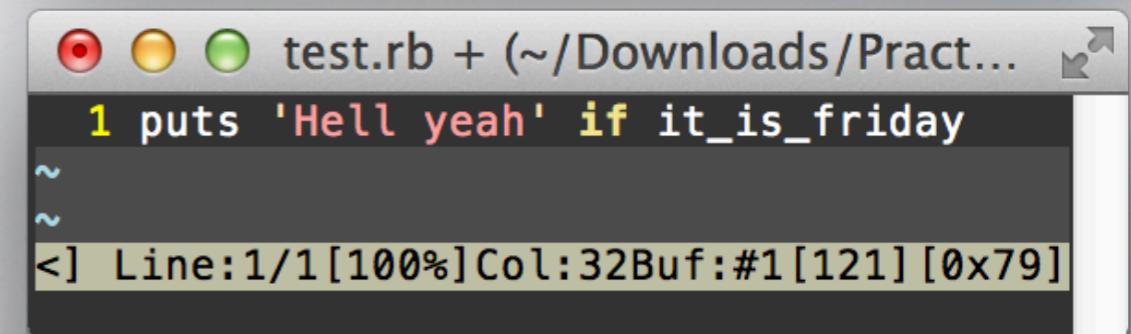
```
test.rb + (~/Downloads/Pract...
1 if a == 1
2   puts "It's one."
3 elsif a == 2
4   puts "It's two..."
5 else
6   puts "IT'S ANOTHER NUMBER!!!!"
7 end
<+> Line:7/7[100%]Col:3Buf:#1[100][0x64]
```



```
test.rb + (~/Downloads/Pract...
1 unless is_working_day
2   puts "Do NOTHING"
3 end
<+> Line:3/3[100%]Col:3Buf:#1[100][0x64]
```



```
test.rb + (~/Downloads/Pract...
1 result = if a == 1
2   2
3   else
4   3
5 end
<+> Line:5/5[100%]Col:10Buf:#1[101][0x65]
```

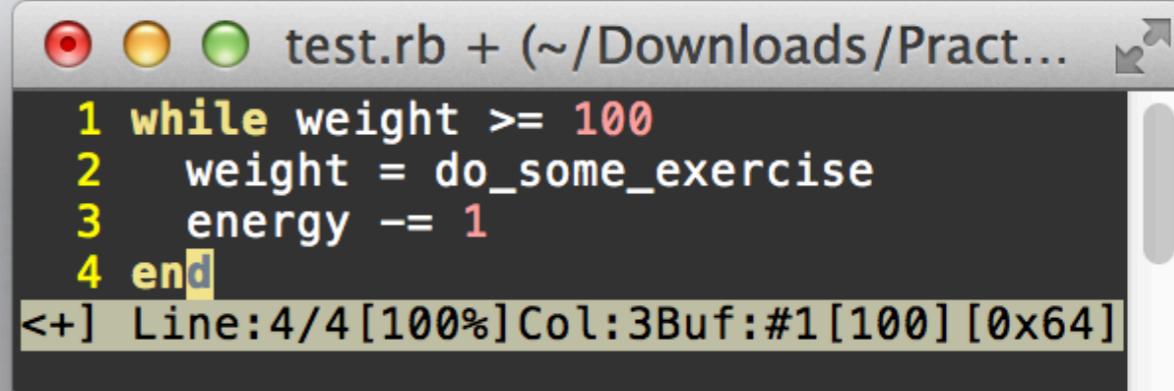


```
test.rb + (~/Downloads/Pract...
1 puts 'Hell yeah' if it_is_friday
~
~
```

<+> Line:1/1[100%]Col:32Buf:#1[121][0x79]

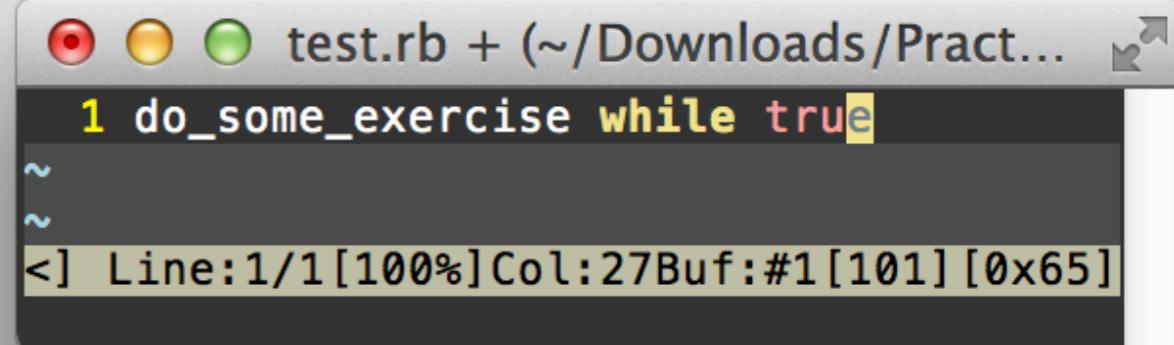
Control structures: while

- Again, pretty standard
- Can also be inline
- I used them once in three years, and it could be done without it



```
test.rb + (~/Downloads/Pract...)
```

```
1 while weight >= 100
2   weight = do_some_exercise
3   energy -= 1
4 end
<+> Line:4/4 [100%] Col:3Buf:#1 [100] [0x64]
```

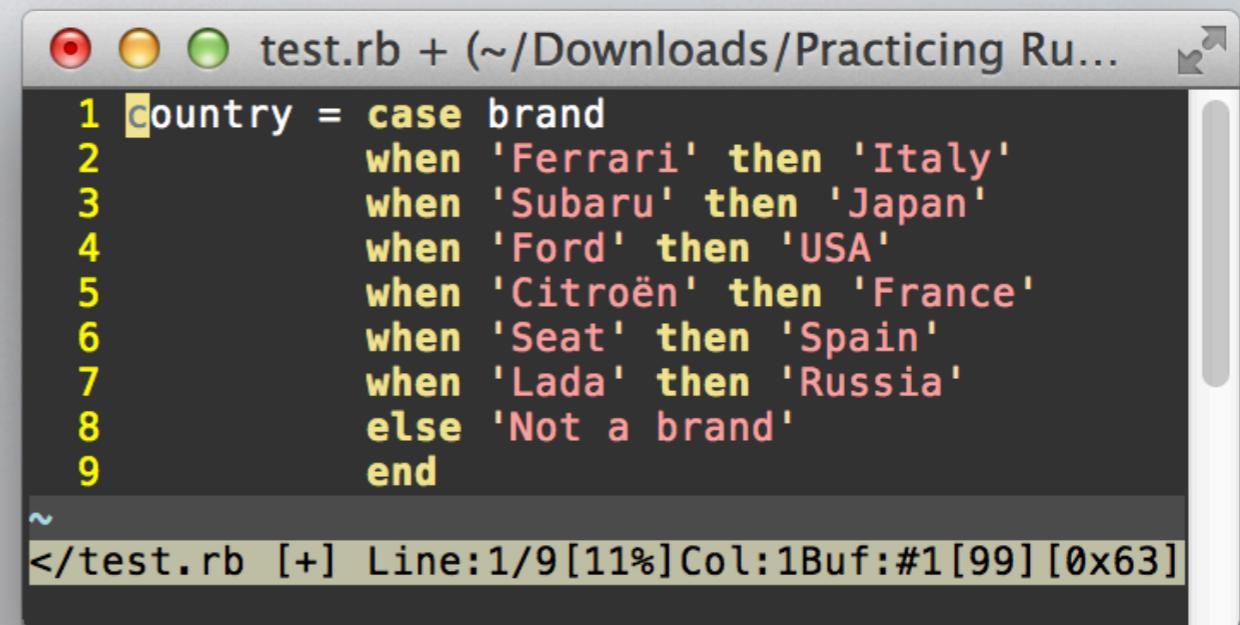


```
test.rb + (~/Downloads/Pract...)
```

```
1 do_some_exercise while true
~
~
```

```
<+> Line:1/1 [100%] Col:27Buf:#1 [101] [0x65]
```

Control structures: case



```
test.rb + (~/Downloads/Practicing Ru...)
```

```
1 country = case brand
2   when 'Ferrari' then 'Italy'
3   when 'Subaru' then 'Japan'
4   when 'Ford' then 'USA'
5   when 'Citroën' then 'France'
6   when 'Seat' then 'Spain'
7   when 'Lada' then 'Russia'
8   else 'Not a brand'
9 end
```

```
~/test.rb [+] Line:1/9[11%]Col:1Buf:#1[99][0x63]
```

- I won't lie... pretty standard
- Returns the corresponding case value, or nil if not found

Regular expressions

- A way of specifying a pattern of characters to be matched in a string
- Typically written between slashes: /a(b|c)d^e?f/
- Can be used with strings with `string =~ regexp` or `string.scan(regexp)`
- Can be very complex, and tricky sometimes
- Pro tip: use rubular.com

Exercices time!

Exercises 3-5 from essentials

Blocks and iterators

- One of the best features in Ruby
- Code blocks are pieces of code that you associate with methods, as if it was a parameter
- Typical uses: callbacks, to pass chunks of code, iterators
- Delimited with { ... } (normally one line) and
do ... end (normally multiple lines)

Blocks and iterators

- A use of a block inside a method:
`do_a_thing_later { puts 'Hi' }`
- The method would be defined like (block not required)
`def do_a_thing_later(&block) ...`
- You can call the chunk of code with `yield` or
`block.call` (if present)

Blocks and iterators

- A simple example

A screenshot of a terminal window titled "test.rb + (~/Downloads/Prac...)" showing the following Ruby code:

```
1 def do_it_thrice
2   puts "I will do it three times"
3   yield
4   yield
5   yield
6   puts "...and I'm done"
7 end
<] Line:7/7[100%]Col:1Buf:#1[101] [0x65]
```

The code defines a method `do_it_thrice` that prints a message and then yields three times. The `end` keyword is highlighted in yellow. Below the code, the terminal shows the command prompt and the output of the method execution.

```
1.9.3-p392 :040 > do_it_thrice { puts 'Hi everyone!' }
I will do it three times
Hi everyone!
Hi everyone!
Hi everyone!
...and I'm done
```

Blocks and iterators

- Blocks can use variables from the method

A screenshot of a terminal window titled "test.rb + (~/Downloads/Practici...)" with three colored tabs (red, yellow, green). The window displays the following Ruby code:

```
1 def for_one_and_three_and_six
2   puts "I will do the same for 1,3,6"
3   yield(1)
4   yield(3)
5   yield(6)
6   puts "...and I'm gonna rest now"
7 end
```

Below the code, the terminal shows the output of running the script:

```
< [+] Line:6/7[85%] Col:33Buf:#1[119] [0x77]
```

```
1.9.3-p392 :048 > for_one_and_three_and_six { |n| puts "I have been passed #{n}" }
I will do the same for 1,3,6
I have been passed 1
I have been passed 3
I have been passed 6
...and I'm gonna rest now
```

Blocks and iterators

- Blocks are used in Ruby to implement **iterators**, which are methods that return successive elements of a collection

```
1.9.3-p392 :049 > [1,2,3,4,5,6,7].each do |n|
1.9.3-p392 :050 >     puts "This is number #{n}"
1.9.3-p392 :051?>   end
This is number 1
This is number 2
This is number 3
This is number 4
This is number 5
This is number 6
This is number 7
```

Exercices time!

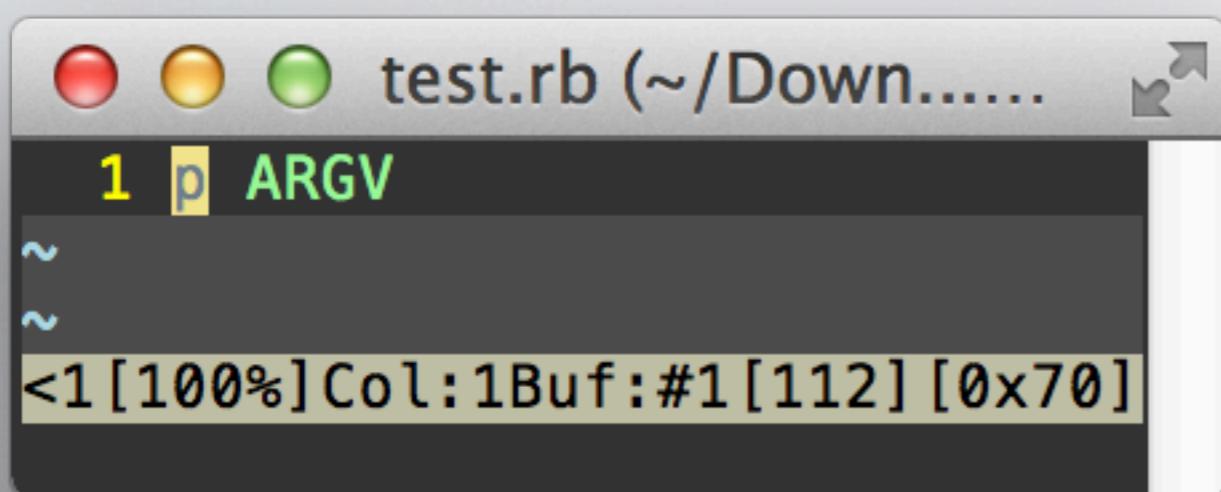
Exercises 6-8 from essentials

Input/output

- **puts** writes its argument(s) to the standard output, and adds a newline character (`\n`)
- **p** does like **puts** but calling **inspect** to the argument(s), which outputs an internal representation
- **print** just prints the argument(s)
- **printf** prints the argument(s) based on a format string (like Perl or C)
- **gets** reads from the standard output until there is a newline character (`\n`)

Command line

- When running Ruby code from the command line, the arguments are stored into a **ARGV** constant
- **ARGV** is simply an array with arguments as strings



A screenshot of a terminal window titled "test.rb (~/Down.....)". The window contains the following text:
1 p ARGV
~
~
<1 [100%] Col:1Buf:#1[112] [0x70]

```
ruby test.rb these are some arguments to you my dear
["these", "are", "some", "arguments", "to", "you", "my", "dear"]
```

Exercices time!

Exercises 9-10 from essentials

About classes and objects

Classes and objects

- A **class** is a construct that abstracts and defines objects sharing some common characteristics
- A dog, a person, a book... can be defined as classes
- An **object** is an instance of a class, which combines state (data), behavior (methods) and identity (uniqueness)
- Your dog, yourself, “Pride and Prejudice”... can be defined as objects of classes

Classes and objects

- With **initialize**, you can create an object of a class, passing some state
- This state is passed in form of variables, that can be stored as **instance variables**
- These variables are only accessible from within that object
- We can use **attr_reader** and **attr_accessor** to expose them through wrappers

Classes and objects

```
test.rb (~/Downloads/Cur...)
```

```
1 class WalkingPerson
2   attr_accessor :status
3   attr_reader :name
4
5   def initialize(name, status)
6     @name = name
7     @status = status
8   end
9 end

< Line:9/9[100%]Col:1Buf:#1[101][0x65]
:
```

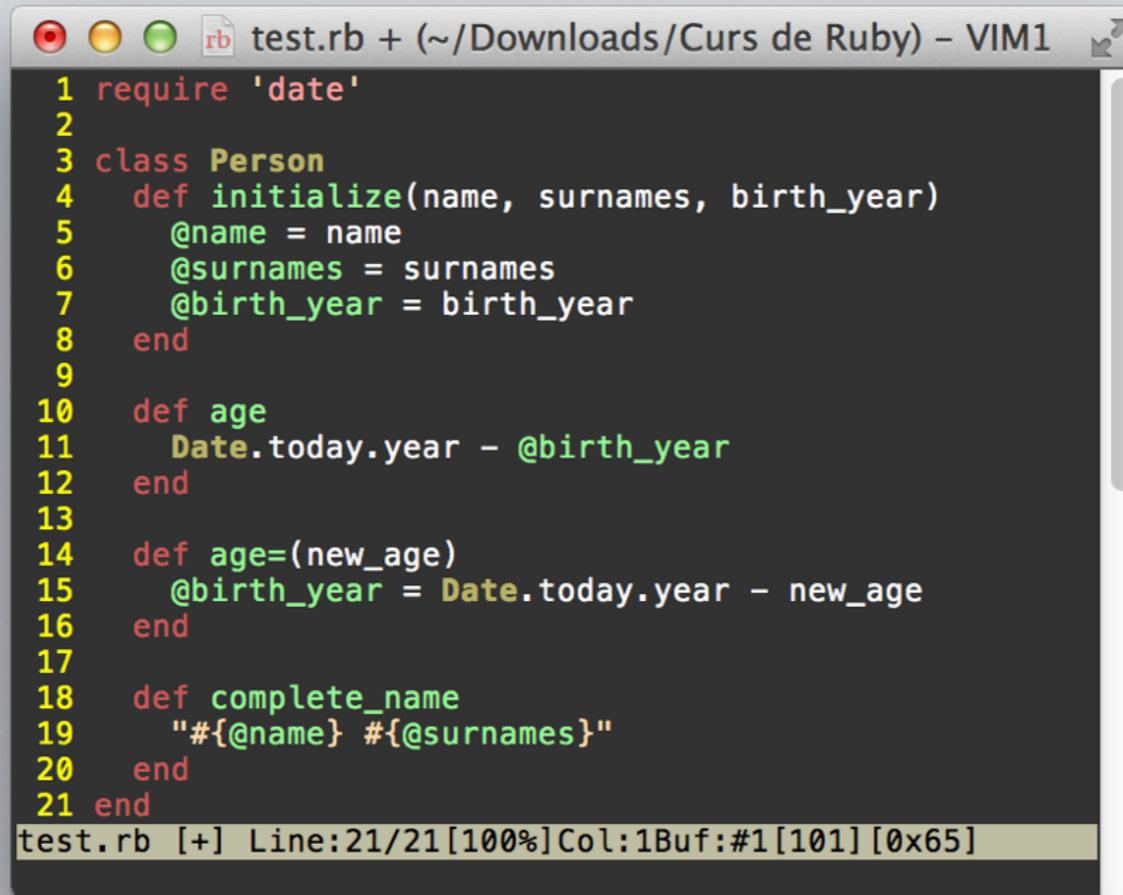
```
macbookpro — ruby — 82×15

1.9.3-p392 :010 > a_person = WalkingPerson.new('Albert', 'fresh as a rose')
=> #<WalkingPerson:0x007fbf43a7bed0 @name="Albert", @status="fresh as a rose">
1.9.3-p392 :011 > a_person.name
=> "Albert"
1.9.3-p392 :012 > a_person.status
=> "fresh as a rose"
1.9.3-p392 :013 > a_person.status = 'a bit tired'
=> "a bit tired"
1.9.3-p392 :014 > a_person.status
=> "a bit tired"
1.9.3-p392 :015 > a_person.name = 'Rose'
NoMethodError: undefined method `name=' for #<WalkingPerson:0x007fbf43a7bed0>
from (irb):15
from /Users/albert/.rvm/rubies/ruby-1.9.3-p392/bin/irb:16:in `<main>'
1.9.3-p392 :016 >
```

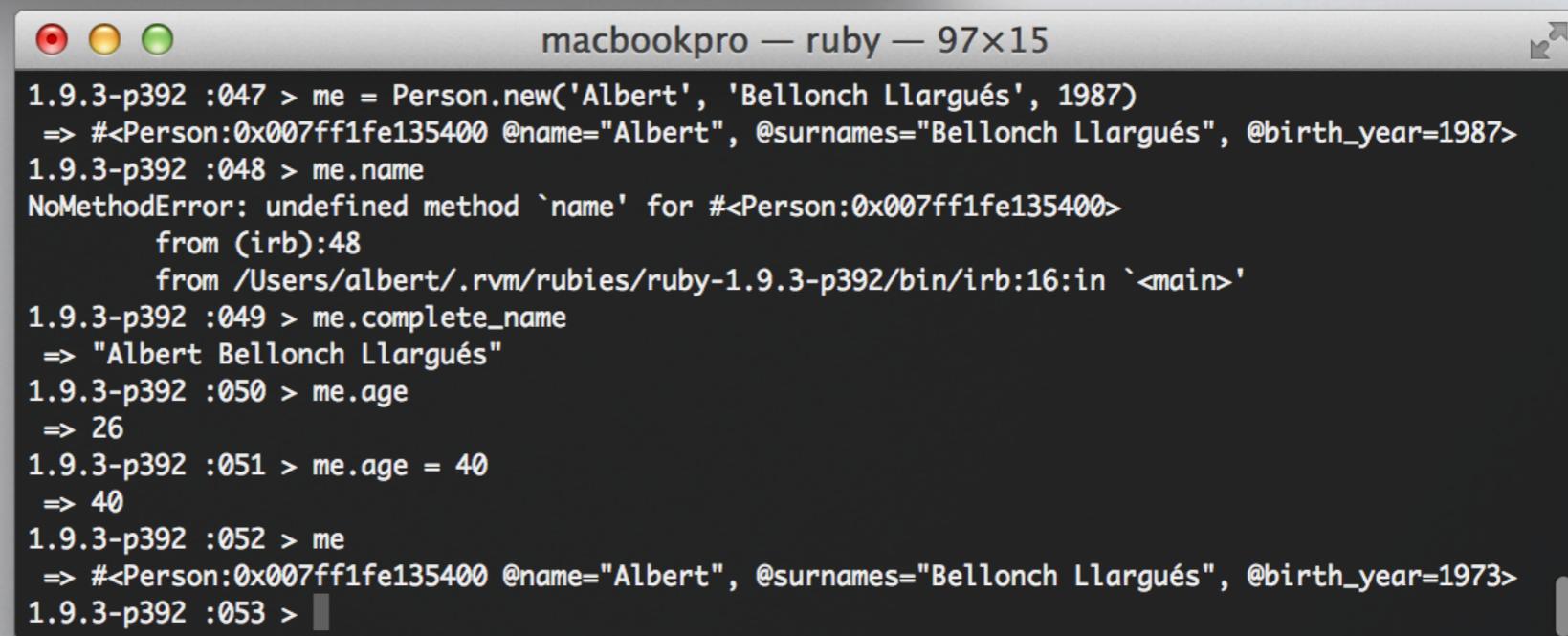
Classes and objects

- **Virtual attributes** are methods that use the state of the object, thus using the object's instance variables
- They can also be used to modify the state, assigning a new value for that virtual attribute
- And also, we can even define methods that do not exactly correspond to an attribute, but are related to the logic of the object

Classes and objects



```
test.rb + (~/Downloads/Curs de Ruby) - VIM1
1 require 'date'
2
3 class Person
4   def initialize(name, surnames, birth_year)
5     @name = name
6     @surnames = surnames
7     @birth_year = birth_year
8   end
9
10  def age
11    Date.today.year - @birth_year
12  end
13
14  def age=(new_age)
15    @birth_year = Date.today.year - new_age
16  end
17
18  def complete_name
19    "#{@name} #{@surnames}"
20  end
21 end
test.rb [+] Line:21/21[100%] Col:1Buf:#1[101] [0x65]
```



```
macbookpro — ruby — 97x15
1.9.3-p392 :047 > me = Person.new('Albert', 'Bellonch Llargués', 1987)
=> #<Person:0x007ff1fe135400 @name="Albert", @surnames="Bellonch Llargués", @birth_year=1987>
1.9.3-p392 :048 > me.name
NoMethodError: undefined method `name' for #<Person:0x007ff1fe135400>
from (irb):48
from /Users/albert/.rvm/rubies/ruby-1.9.3-p392/bin/irb:16:in `<main>'
1.9.3-p392 :049 > me.complete_name
=> "Albert Bellonch Llargués"
1.9.3-p392 :050 > me.age
=> 26
1.9.3-p392 :051 > me.age = 40
=> 40
1.9.3-p392 :052 > me
=> #<Person:0x007ff1fe135400 @name="Albert", @surnames="Bellonch Llargués", @birth_year=1973>
1.9.3-p392 :053 >
```

Classes and objects

- As classes encapsulate pieces of logic, almost always they interoperate together
- You would use other classes as we have been doing with **irb**
- We can use **require_relative** to load external classes into our class

Classes and objects

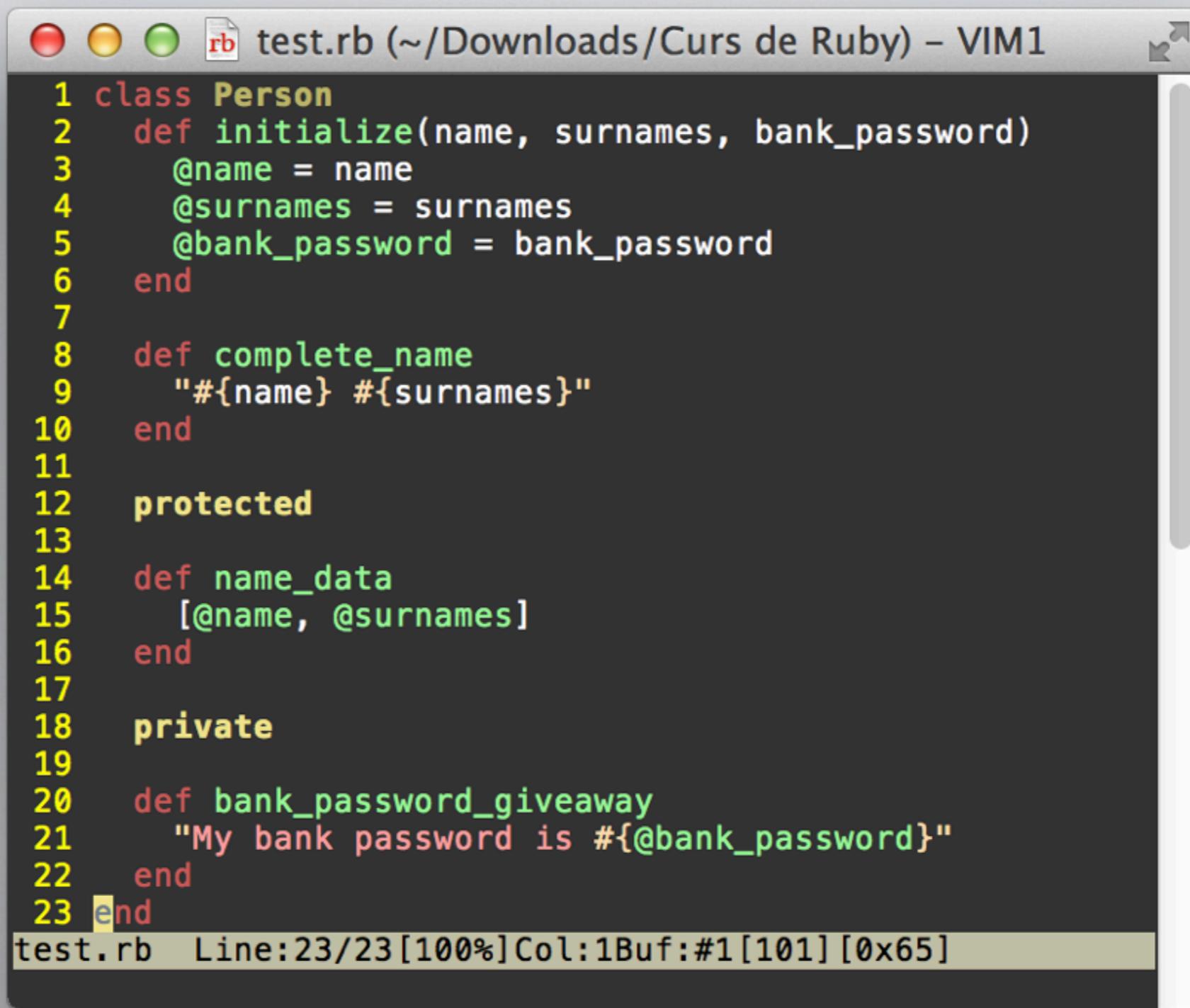
```
test.rb + (~/Downloads/Curs de Ruby) - VIM1
1 require 'date'
2 require_relative 'age_calculator.rb'
3
4 class Person
5   def initialize(name, surnames, birth_year)
6     @name = name
7     @surnames = surnames
8     @birth_year = birth_year
9   end
10
11  def age
12    AgeCalculator.new(@birth_year).age
13  end
14
15  def age=(new_age)
16    @birth_year = Date.today.year - new_age
17  end
18
19  def complete_name
20    "#{@name} #{@surnames}"
21  end
22 end
test.rb [+] Line:1/22[4%]Col:1Buf:#1[114][0x72]
```

```
test.rb + (~/Downloads/Curs d...
1 class AgeCalculator
2   def initialize(birth_year)
3     @birth_year = birth_year
4   end
5
6   def age
7     Date.today.year - @birth_year
8   end
9 end
<rb [+] Line:9/9[100%]Col:1Buf:#1[101][0x65]
```

Classes and objects

- Normally you don't want some of your methods to be available to other classes
- methods are **public** by default
- protected methods are placed after the keyword **protected**, and only accessible by subclasses of our class
- private methods are placed after the keyword **private**, and only accessible to our own class

Classes and objects



A screenshot of a VIM editor window titled "test.rb (~/Downloads/Curs de Ruby) - VIM1". The window shows the following Ruby code:

```
1 class Person
2   def initialize(name, surnames, bank_password)
3     @name = name
4     @surnames = surnames
5     @bank_password = bank_password
6   end
7
8   def complete_name
9     "#{name} #{surnames}"
10  end
11
12  protected
13
14  def name_data
15    [@name, @surnames]
16  end
17
18  private
19
20  def bank_password_giveaway
21    "My bank password is #{@bank_password}"
22  end
23 end
```

The status bar at the bottom of the window displays "test.rb Line:23/23 [100%] Col:1 Buf:#1 [101] [0x65]".

Exercices time!

All 3 exercises from classes & objects

Collection classes, blocks and iterators

Arrays

- We already know something about them
- There are two different ways of creating them
 - Literals: [1, 2, 3,], %w{ a b c }, %i{ one two }
 - Object: a = Array.new; a[1] = 'Here'
 - Indexed by the [] operator
 - Can hold any type of object [1, %w{ a b }, Cat.new]
 - array << element adds element to array

Arrays

- Many interesting methods defined within the Array class
(check <http://ruby-doc.org/core-2.0.0/Array.html>)
 - You can use **push** and **pop** as if it was a pile
 - **compact** returns the same array but without *nils*
 - **drop** removes the first n elements of it
 - **empty?** returns a boolean with its emptiness
 - **flatten** integrates subarrays into the main array
 - **include?** says if an element is there or not
 - **reverse** turns the array around
 - **sample** gives us a random element
 - **select** returns the elements for which the block is true
 - **size** returns the number of elements

Hashes

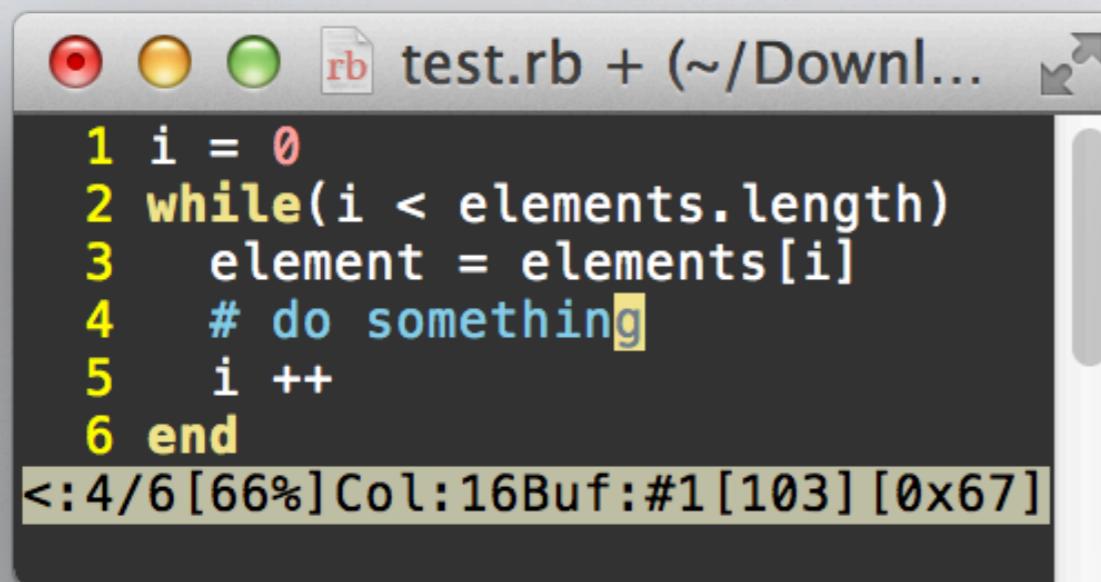
- I repeat myself: we already know something about them
- We also know there are two different syntaxes to declare them: { “a”=>1, “b”=>2 } vs { a: 1, b: 2 }
- We can also initialize it with `Hash.new`, the first and one parameter being the default value when there a key does not exist
- From Ruby 1.9, the order is remembered

Hashes

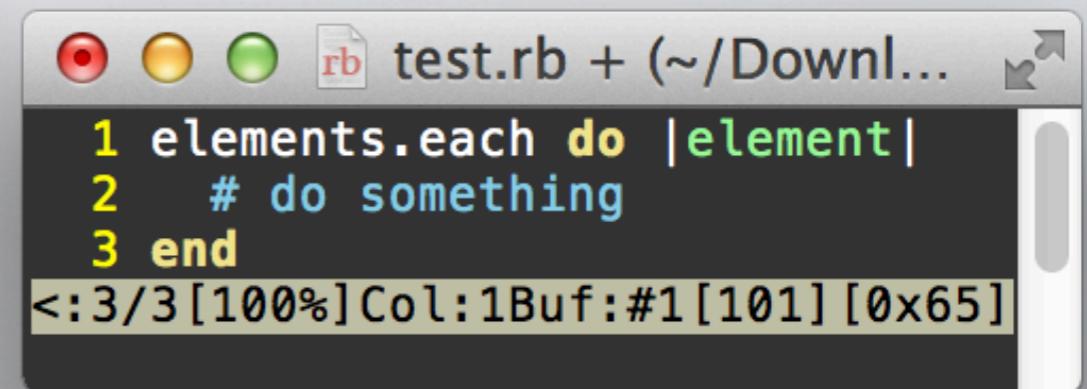
- Also, many interesting methods defined within the Hash class (check <http://ruby-doc.org/core-2.0.0/Hash.html>)
 - **keys** and **values** return... the keys and the values
 - **delete** takes a key as a parameter and deletes it from the hash
 - **empty?** returns if there are values or not
 - **invert** inverts keys and values
 - **merge** takes a hash as a parameter and adds its information to the current hash
 - **to_a** returns an array in which every element is an array with both key and value

Blocks and iterators

- We treated (a bit) of it
- Ruby allows us to switch left for right



```
i = 0
while(i < elements.length)
  element = elements[i]
  # do something
  i ++
end
<:4/6[66%]Col:16Buf:#1[103][0x67]
```



```
elements.each do |element|
  # do something
end
<:3/3[100%]Col:1Buf:#1[101][0x65]
```

- For example, we can use **each** on both arrays (one iteration variable) and hashes (two)

Blocks and iterators

- There is also a **map** method for arrays and hashes
 - For each iteration variable, it replaces it with the block applied to the variable
 - An example

```
1.9.3-p392 :001 > { a: 1, b: 2, c: 3 }.map do |key, value|  
1.9.3-p392 :002 >     value * 10  
1.9.3-p392 :003?>   end  
=> [10, 20, 30]
```

- One of the most useful methods not only in arrays/ hashes, but in the whole Ruby programming language

Blocks and iterators

- When calling **with_index** to iterators like **map** or **each**, it adds the element index as a second instance variable

```
1.9.3-p392 :026 > [1,2,3,4,5].each.with_index do |n,i|
1.9.3-p392 :027 >     p [n,i]
1.9.3-p392 :028?>   end
[1, 0]
[2, 1]
[3, 2]
[4, 3]
[5, 4]
=> [1, 2, 3, 4, 5]
```

Blocks and iterators

- Inside blocks, we have the context of outside plus the iteration variable
 - If **a** exists outside the block and the iteration variable is also called **a**, calling **a** from inside a block will use the iteration variable

```
1.9.3-p392 :004 > a = 5
=> 5
1.9.3-p392 :005 > [1,2,3,4].each do |a|
1.9.3-p392 :006 >     puts a
1.9.3-p392 :007?>   end
1
2
3
4
```

Blocks and iterators

- There are also **enumerators**
- They are, let's say, external *iterators*: they just help us iterate through an object, but from outside
- Example:

```
1.9.3-p392 :008 > a = %w{this is a string array }
=> ["this", "is", "a", "string", "array"]
1.9.3-p392 :009 > h = { a: 'hash', rocks: 'pretty much' }
=> {:a=>"hash", :rocks=>"pretty much"}
1.9.3-p392 :010 > enum_a = a.to_enum; enum_h = h.to_enum
=> #<Enumerator: {:a=>"hash", :rocks=>"pretty much"}:each>
1.9.3-p392 :011 > enum_a
=> #<Enumerator: ["this", "is", "a", "string", "array"]:each>
1.9.3-p392 :012 > enum_a.next
=> "this"
1.9.3-p392 :013 > enum_a.next
=> "is"
1.9.3-p392 :014 > enum_a.next
=> "a"
```

Report erratum

this copy is (3.0 printing, November 2010)

- Useful to, for example, iterate through two collections at once

Exercices time!

All 7 exercises from collections,
blocks and iterators

Standard data types

Numbers

- There are integers (**Fixnum** and **Bignum**), floating point (**Float**), rational (**Rational**) and complex (**Complex**) numbers
- Integers can be represented
 - As you would expect: **12345**
 - With a specific base: **0o12** , **0d54** , **0x102** , **0b100**
 - In negative: **-69**
 - With underscores: **123_456_789_100**

Numbers

- Floating point numbers are declared as expected:
 - `1.56`, `-34.109`, `2.0`, `.49`
- Rationals and complex numbers have no literal syntax, so we just use the class:
 - `Rational(2,3)` or `Rational("2/3")`
 - `Complex(3,4)` or `Complex("3+4i")`
- Rationals are really awesome
 - `Rational(2,3) * Rational(3,2)` is just 1, not `1.000001` or `0.9999999`

Numbers

- When operating with numbers of the same class, the result will be of the same class (with some exceptions like I/O or adding two fixnums into a bignum)
- If the numbers are from different classes, the most general class is used:
 - $1 + 2 \Rightarrow 3$
 - $1.0 + 5 \Rightarrow 6.0$
 - $3 + \text{Rational}(2, 3) \Rightarrow 11/3$
 - $2.3 + \text{Rational}(1, 6) \Rightarrow 2.4666666666666663$

Numbers

- Integer iterators
 - `3.times { puts 'I will not watch Jersey Shore' }`
 - `1.upto(9) { |i| puts "From 1 to 9, now #{i} " }`
 - `5.downto(1) { |i| puts "Decreasing (#{i}/5)" }`
 - `100.step(200,20) { |i| puts "Goin' (#{i}/200)" }`

Strings

- Yeah, sequences of characters
- Defined with string literals, either single quoted or double quoted: **‘Mamma’** or **“Double quotes!”**
- Double quoted strings support more escape sequences, and even interpolation

```
1.9.3-p392 :001 > name = 'Joe'  
=> "Joe"  
1.9.3-p392 :002 > "My name is #{name}"  
=> "My name is Joe"  
1.9.3-p392 :003 >
```

- And you can add them: “A” + “ dog” => “A dog”

Strings

- Unconventional ways to define them
 - With %q or %Q and equal delimiters in the sides
 - %q/hey I'm a text between slashes/
 - %Q!but I am more cool!
 - %{LOL, the q is optional}
 - With here documents
 - Format: <<-DELIMITER DELIMITER
 - Example:

```
1.9.3-p392 :001 > rolling = <<-THIS_IS_A_RUBY_WORKSHOP
1.9.3-p392 :002"> I can't get no, oh no, no, no
1.9.3-p392 :003"> A hey, hey, hey, that's what I say
1.9.3-p392 :004"> I can't get no satisfaction
1.9.3-p392 :005"> THIS_IS_A_RUBY_WORKSHOP
eos => "I can't get no, oh no, no, no\nA hey, hey, hey, that's what I say\nI can't get no satisfaction\n"
```

Strings

- There are a lot of interesting methods defined within the String class (check <http://ruby-doc.org/core-2.0.0/String.html>)
- Methods do not modify the original string (except !'s)

Strings

- Some interesting methods
 - **capitalize**: first letter in upcase, rest in downcase
 - **chop**: erases the last character
 - **empty?**: tells us if it's empty or not
 - **include?**: tells us if it includes another string passed by parameter or not
 - **match**: matches a regexp and returns any results
 - **size**: returns the character length
 - **split**: divides de string in an array of substrings based on a pattern
 - **to_i**: converts it to integer if possible

Ranges

- Ranges are sequences of some kind of symbol, from one point to another, in an ascending way
- In Ruby, ranges are defined through the `..` and `...` operators
- Examples:
 - `-5..69`
 - `'a'..'t'`
 - `0...'learning Ruby'.length`
 - `5..2` (empty!)

Ranges

- With `..`, you include the extremes
- With `... ,`, you exclude the last value
- Ranges can be converted to arrays with `to_a`
- They can also be converted to enumerators with `to_enum`
- You can use many methods that apply to arrays, like `include?`, `min`, `max` or `each`

Exercices time!

All 5 exercises from basic data types

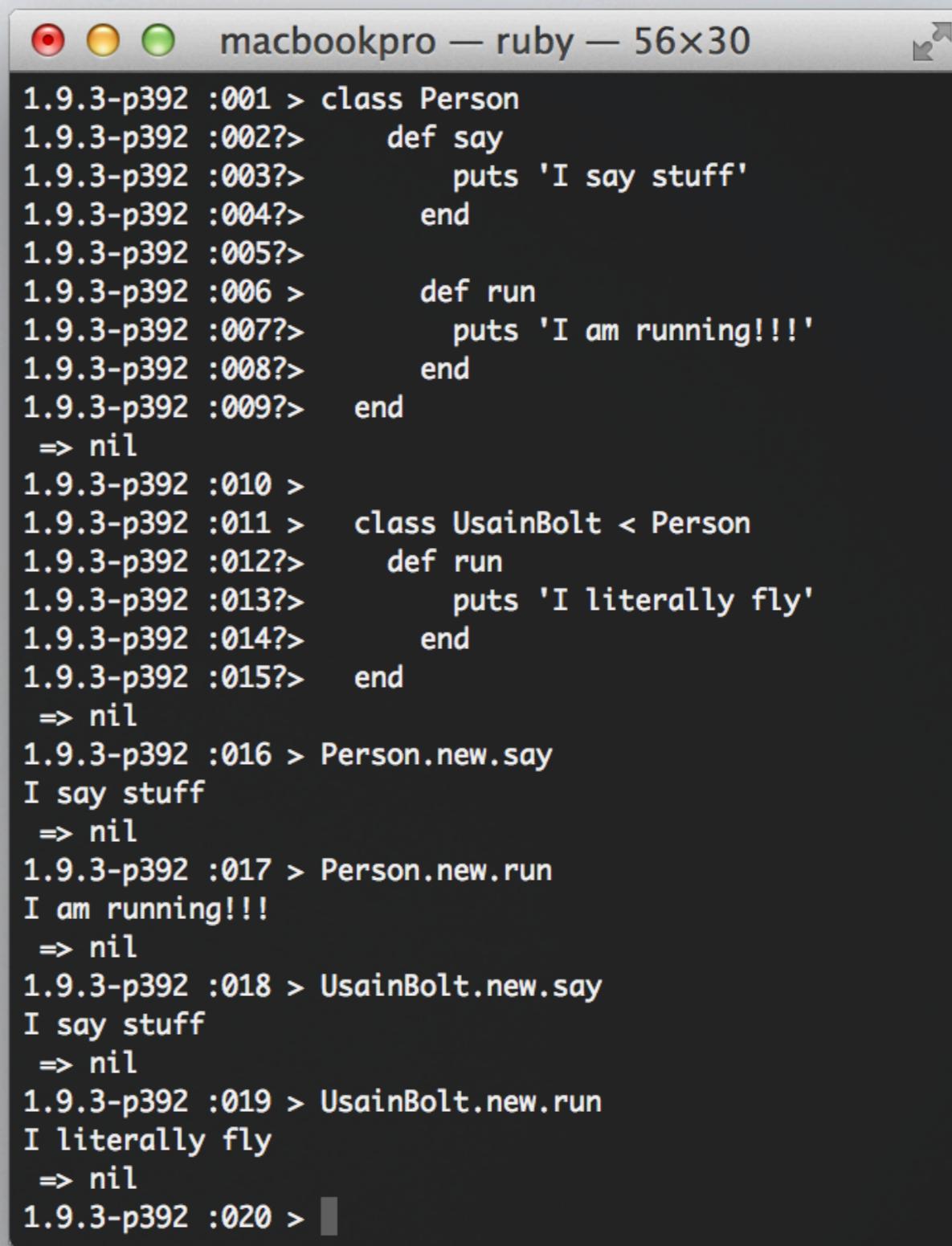
Other stuff

Inheritance and modules

- It allows to refine or specialize a certain class, thus being a subclass
- The subclass inherits everything (methods, state)
- If a subclass defines a method with the same name as one of the parent class, the subclass' method is used
- Unless specified otherwise, any Ruby object has the Object class as a superclass

Inheritance and modules

- Example



```
macbookpro — ruby — 56x30
1.9.3-p392 :001 > class Person
1.9.3-p392 :002?>     def say
1.9.3-p392 :003?>         puts 'I say stuff'
1.9.3-p392 :004?>     end
1.9.3-p392 :005?>
1.9.3-p392 :006 >     def run
1.9.3-p392 :007?>         puts 'I am running!!!'
1.9.3-p392 :008?>     end
1.9.3-p392 :009?> end
=> nil
1.9.3-p392 :010 >
1.9.3-p392 :011 > class UsainBolt < Person
1.9.3-p392 :012?>     def run
1.9.3-p392 :013?>         puts 'I literally fly'
1.9.3-p392 :014?>     end
1.9.3-p392 :015?> end
=> nil
1.9.3-p392 :016 > Person.new.say
I say stuff
=> nil
1.9.3-p392 :017 > Person.new.run
I am running!!!
=> nil
1.9.3-p392 :018 > UsainBolt.new.say
I say stuff
=> nil
1.9.3-p392 :019 > UsainBolt.new.run
I literally fly
=> nil
1.9.3-p392 :020 >
```

Inheritance and modules

- It is very common to use inheritance in order to:
 - **Extract a common logic.** If A and B share something, let's put it in C and make A and B inherit from C.
 - **Extend a behaviour.** If A wants to use all stuff from B, then A inherits from B.

Inheritance and modules

- What if we want to inherit from more than class at once?
- Some languages implement it but it's complex and dangerous, and we want simplicity
- **Modules** group methods, classes and constants
- A natural result of writing reusable code: we end up with libraries that can be applicable to several places

Inheritance and modules

```
1.9.3-p392 :001 > module Sayer
1.9.3-p392 :002?>     def say
1.9.3-p392 :003?>         puts 'I can speak'
1.9.3-p392 :004?>     end
1.9.3-p392 :005?> end
=> nil
1.9.3-p392 :006 >
1.9.3-p392 :007 > class Person
1.9.3-p392 :008?> include Sayer
1.9.3-p392 :009?> end
=> Person
1.9.3-p392 :010 >
1.9.3-p392 :011 > class TalkingDog
1.9.3-p392 :012?> include Sayer
1.9.3-p392 :013?> end
=> TalkingDog
1.9.3-p392 :014 > Person.new.say
I can speak
=> nil
1.9.3-p392 :015 > TalkingDog.new.say
I can speak
=> nil
```

- Example

Exceptions

- Errors **always** happen, sooner or later
- We have to manage them properly
- Return codes are the traditional way, but difficult to maintain and standardize
- **Exceptions** package information about the error, and Ruby provides a nice environment to catch them
- Similar in Java, for example

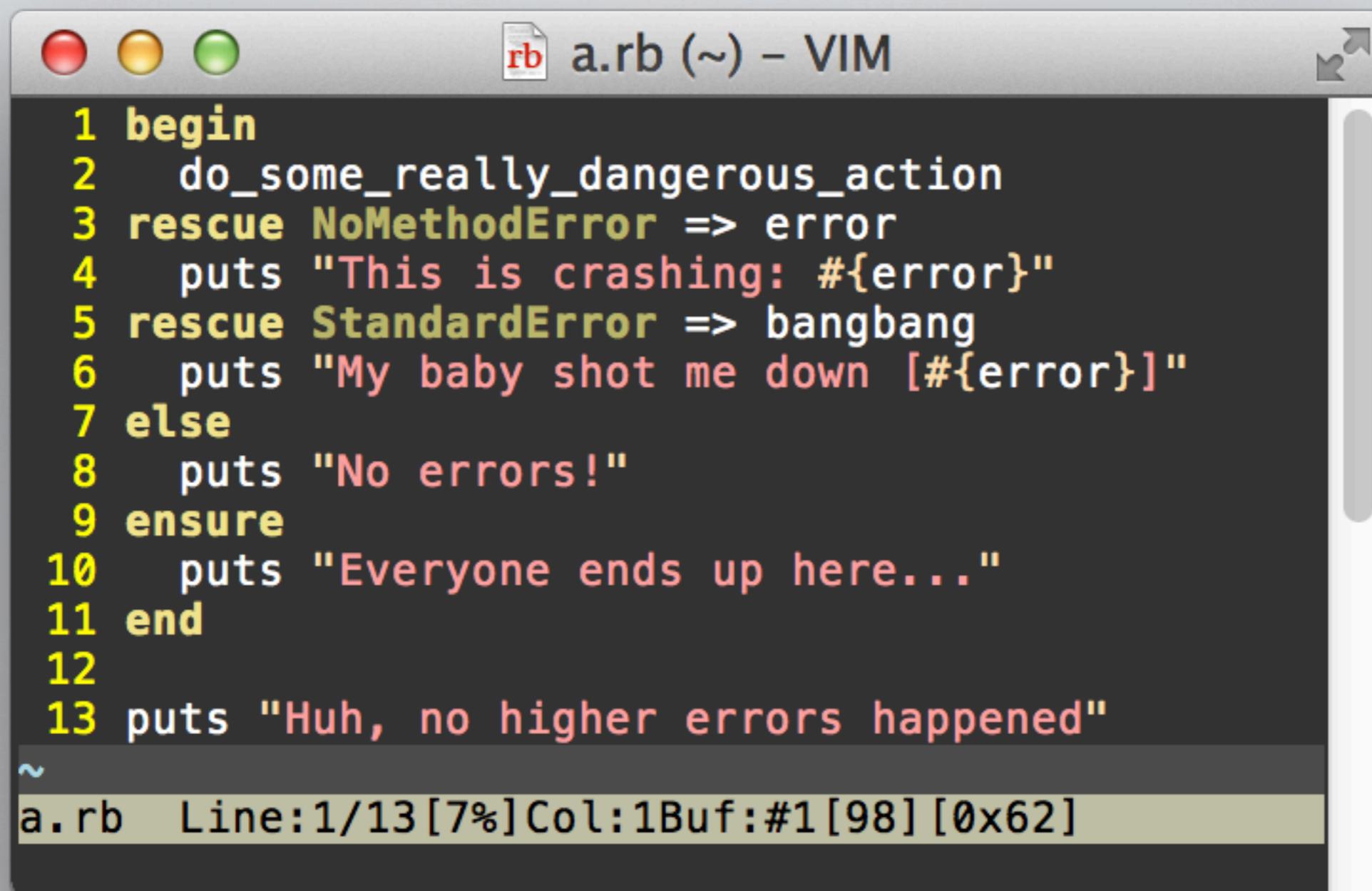
Exceptions

- The exception goes up the calling stack until it is handled
- If it is not handled, execution will stop (in classical monolithic applications) or, in web applications, we will receive a nice 500 error
- Tidy hierarchy defined as default
- We can define our own exceptions, which might package domain-specific more information

Exceptions

- Block with `begin .. rescue .. end`
- Normal code goes between `begin` and `rescue`
- Error handling goes between `rescue` and `end`
- You can capture the `Exception` object after `rescue`
- We can handle different exceptions differently
- We can use `else` for non-exception cases
- Finally, `ensure` gets executed either way

Exceptions



The screenshot shows a VIM window with a dark background and light-colored text. The window title is "a.rb (~) - VIM". The file contains the following Ruby code:

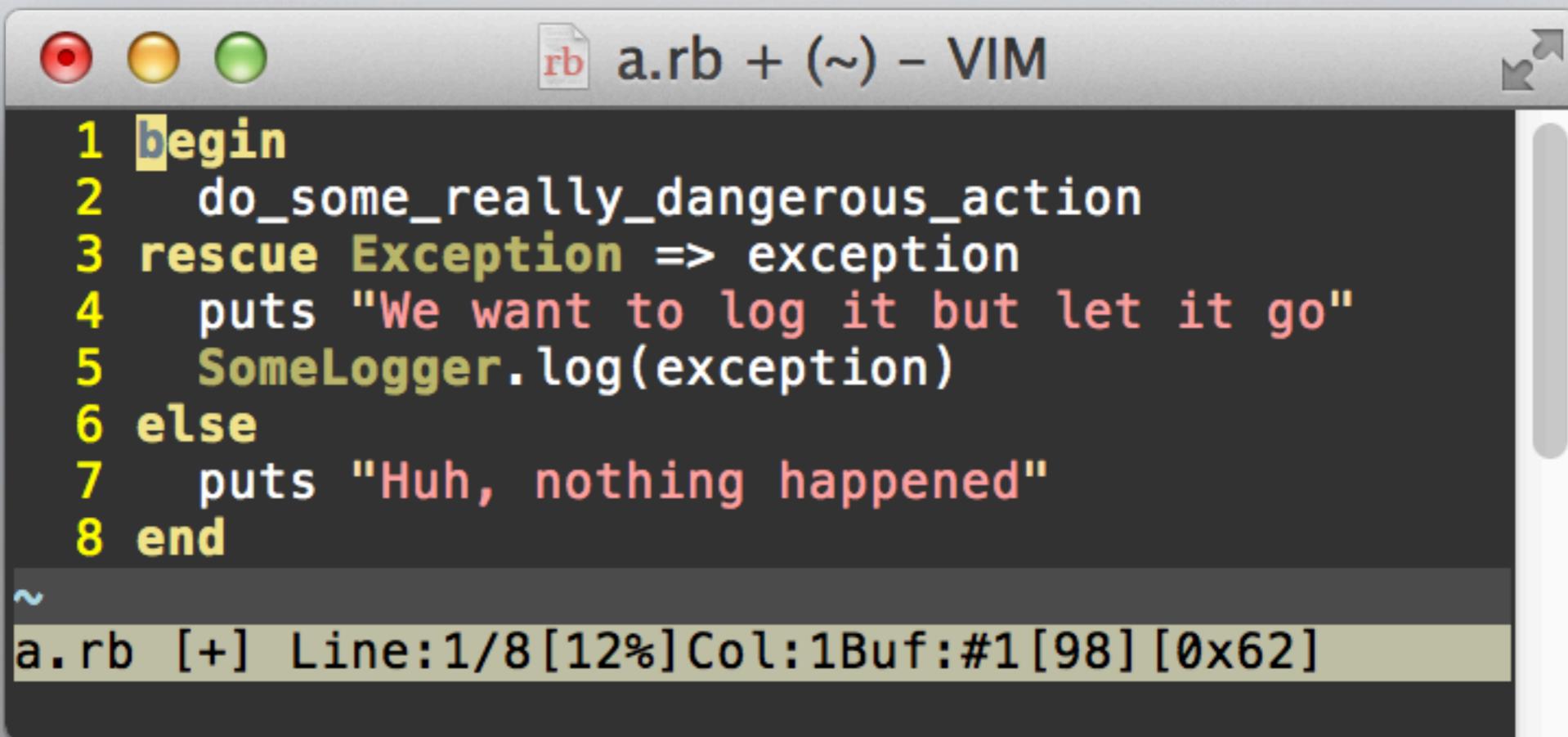
```
1 begin
2   do_some_really_dangerous_action
3 rescue NoMethodError => error
4   puts "This is crashing: #{error}"
5 rescue StandardError => bangbang
6   puts "My baby shot me down [#{error}]"
7 else
8   puts "No errors!"
9 ensure
10  puts "Everyone ends up here..."
11 end
12
13 puts "Huh, no higher errors happened"
```

The status bar at the bottom shows the file name "a.rb", line information "Line:1/13[7%]", column information "Col:1Buf:#1[98] [0x62]", and a cursor icon.

Exceptions

- We can use `raise` to, well, raise exceptions
- If no parameters are passed, it raises the current exception, or `RuntimeError` otherwise
- If a string is passed, it raises a `RuntimeError` with the string as the message
- If some more arguments are passed; they can be the exception, a message, etc.

Exceptions



A screenshot of a VIM window displaying Ruby code. The window title is "a.rb + (~) - VIM". The code is:

```
1 begin
2   do_some_really_dangerous_action
3 rescue Exception => exception
4   puts "We want to log it but let it go"
5   SomeLogger.log(exception)
6 else
7   puts "Huh, nothing happened"
8 end
```

The status bar at the bottom shows the file name "a.rb", line count "[+]", line number "Line:1/8", percentage "12%", column number "Col:1", buffer number "Buf:#1", character offset "98", and memory address "[0x62]."

Exercices time!

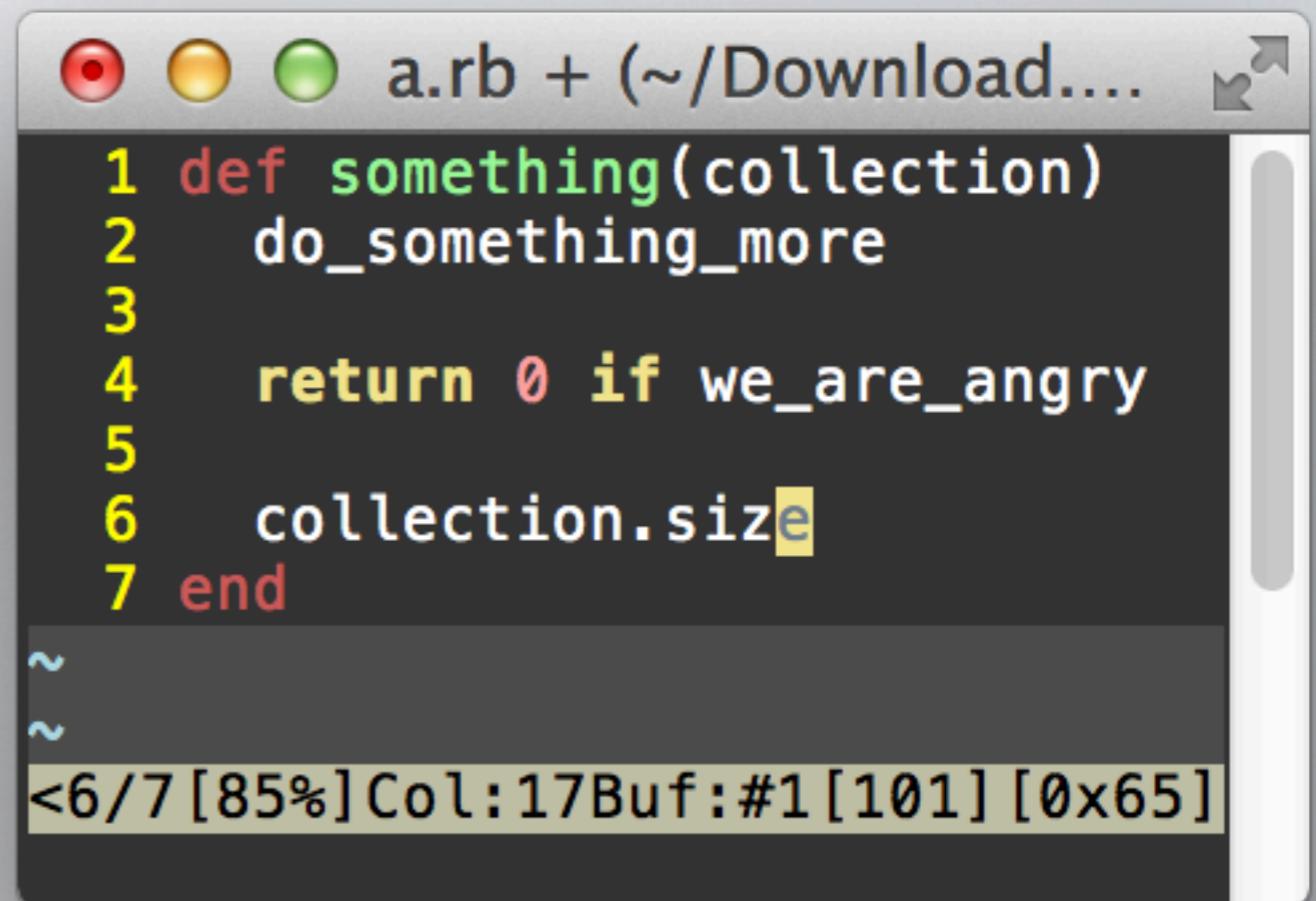
All 4 exercises from other stuff

Tricks and differential treats

If I haven't done it already, **now** I will
convince you that **Ruby** is awesome

Implicit return

- A method returns the last expression of the method (unless we use `return`)
- Less and more beautiful code!



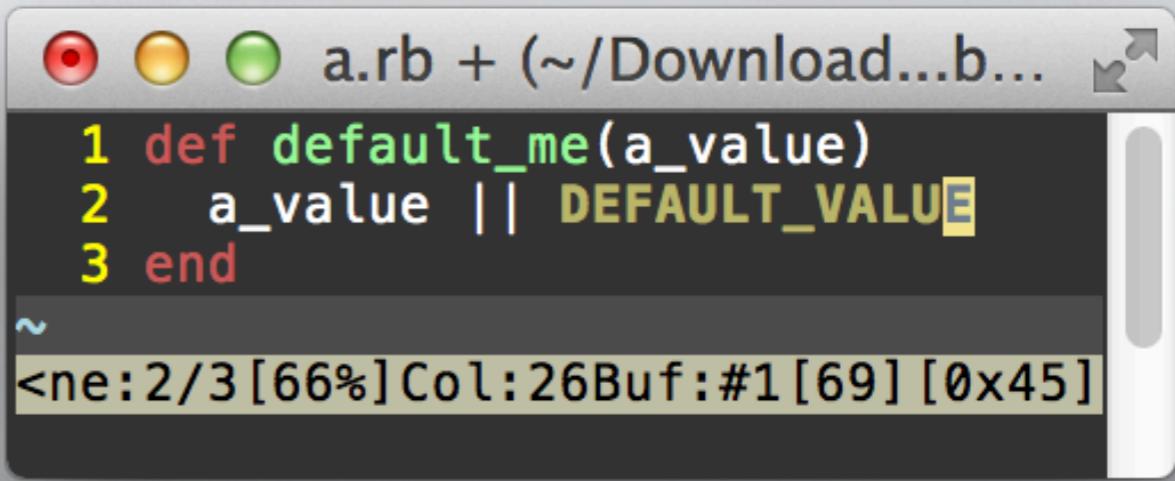
A screenshot of a terminal window titled "a.rb + (~/Download....)". The window contains the following Ruby code:

```
1 def something(collection)
2   do_something_more
3
4   return 0 if we_are_angry
5
6   collection.size
7 end
```

The terminal window shows the cursor at the end of the word "size". At the bottom of the window, the status bar displays: "<6/7 [85%] Col:17 Buf:#1 [101] [0x65]".

Use the ORs in assignments

- Using ORs can help us write less code too, in assignations
- For example, to provide default values to variables



```
a.rb + (~/Download...b...
1 def default_me(a_value)
2   a_value || DEFAULT_VALUE
3 end
~<ne:2/3 [66%] Col:26Buf:#1 [69] [0x45]
```

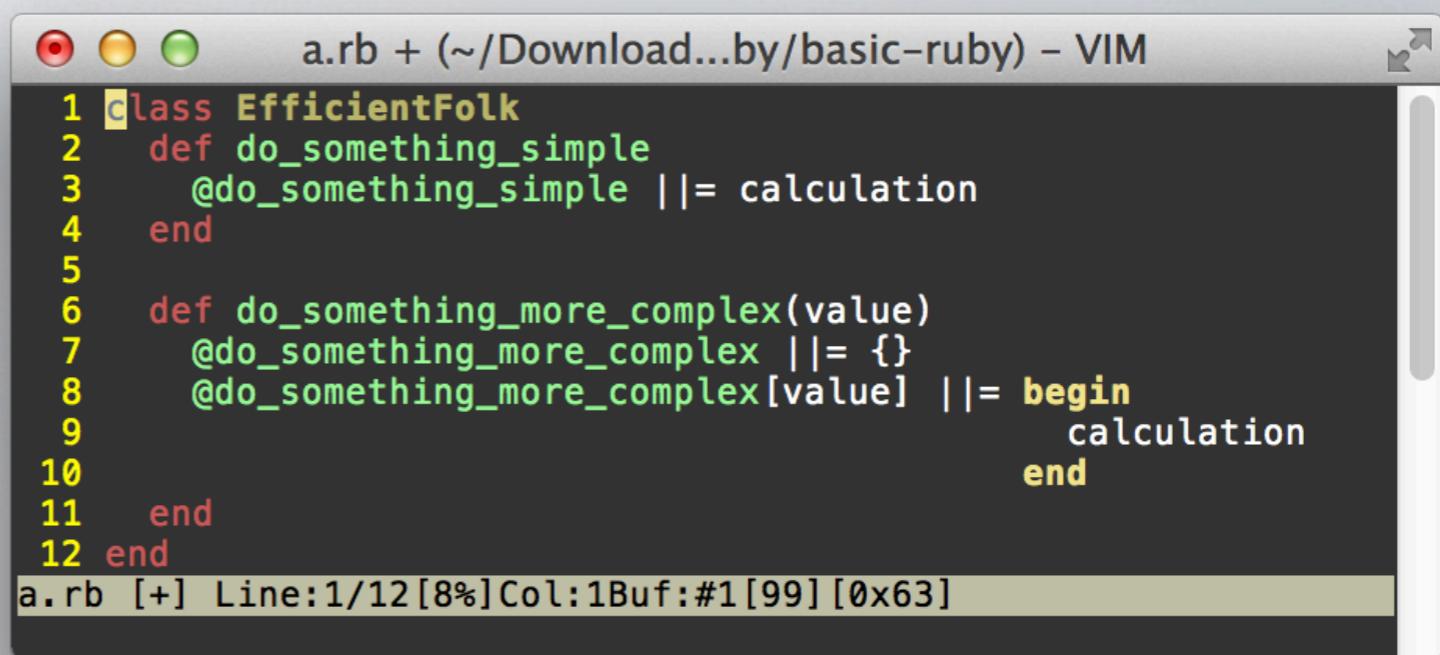
The screenshot shows a terminal window with the following Ruby code:

```
1 def default_me(a_value)
2   a_value || DEFAULT_VALUE
3 end
```

The code defines a method `default_me` that takes a parameter `a_value`. It returns `a_value` if it's not nil, or the constant `DEFAULT_VALUE` if it is. The terminal window also shows the file name `a.rb` and some status information at the bottom.

Memoization

- Basically caches the result of a method
- And there is no second point



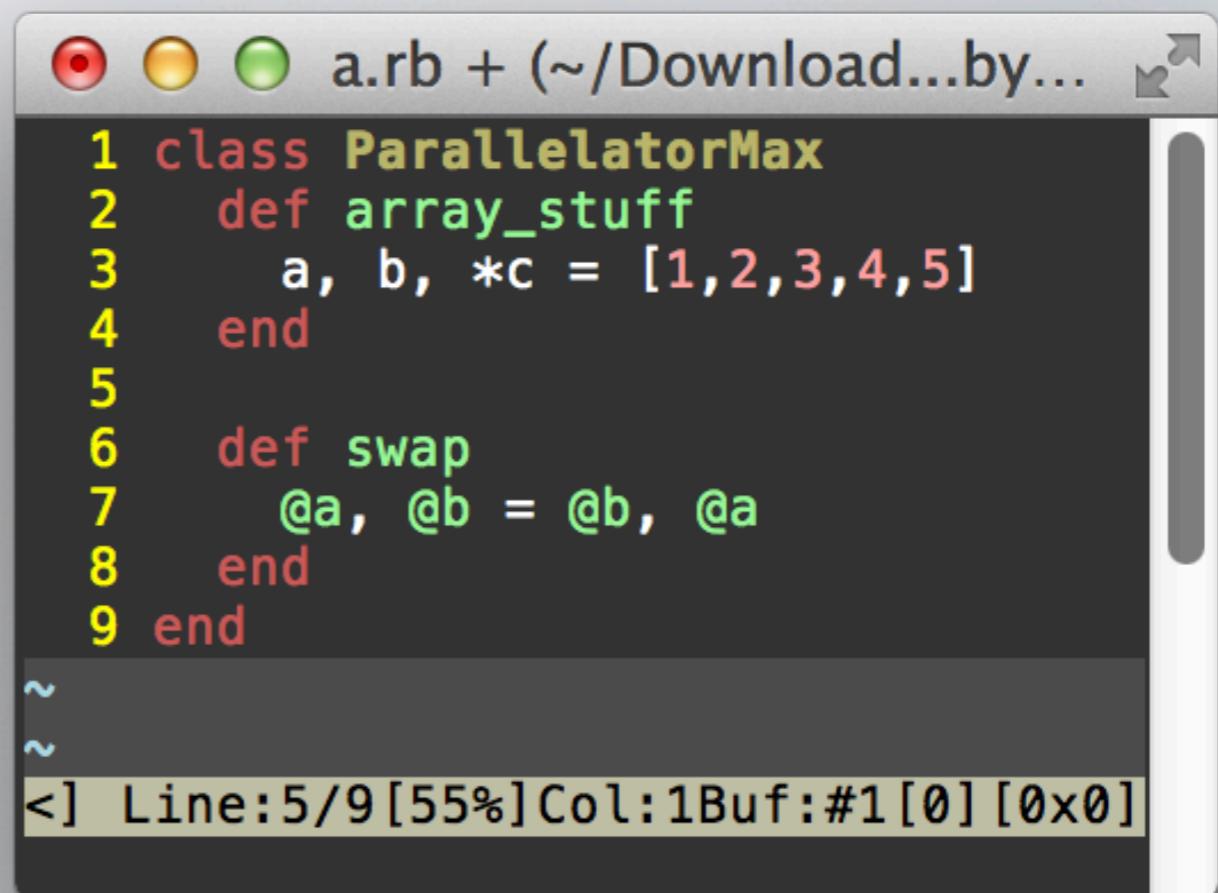
A screenshot of the VIM text editor showing a Ruby script named 'a.rb'. The code defines a class 'EfficientFolk' with two methods: 'do_something_simple' and 'do_something_more_complex'. The 'do_something_simple' method simply returns the value of the instance variable '@do_something_simple'. The 'do_something_more_complex' method uses a hash to cache results based on the input 'value'. If the result is already in the hash, it is returned; otherwise, it is calculated and stored in the hash before being returned. The code is color-coded for readability.

```
1 class EfficientFolk
2   def do_something_simple
3     @do_something_simple ||= calculation
4   end
5
6   def do_something_more_complex(value)
7     @do_something_more_complex ||= {}
8     @do_something_more_complex[value] ||= begin
9       calculation
10      end
11    end
12 end
```

a.rb [+] Line:1/12 [8%] Col:1 Buf:#1 [99] [0x63]

Parallel assignation

- Allows multiple assignments in one line
- Relieves us from using the damn middle variable when swapping



```
a.rb + (~/Download...by...)
```

```
1 class ParallelatorMax
2   def array_stuff
3     a, b, *c = [1,2,3,4,5]
4   end
5
6   def swap
7     @a, @b = @b, @a
8   end
9 end
```

```
~
```

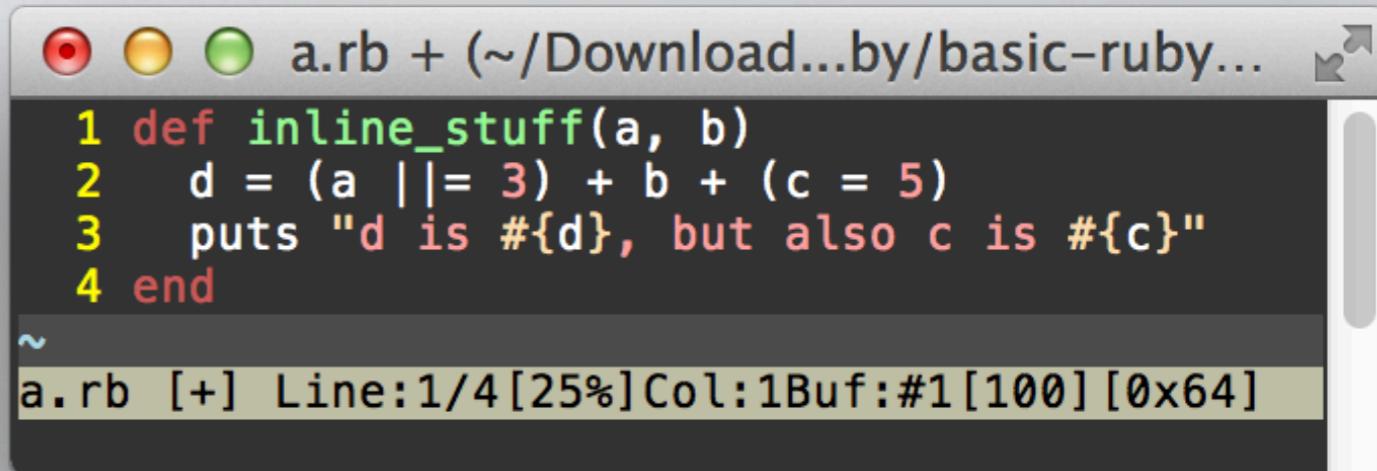
```
~
```

```
<] Line:5/9[55%]Col:1Buf:#1[0] [0x0]
```

A screenshot of a terminal window titled "a.rb + (~/Download...by...)". The window contains Ruby code. Lines 1 through 4 define a class named "ParallelatorMax" with a method "array_stuff" that takes a list of five integers and assigns them to variables a, b, and c. Lines 6 through 9 define another method "swap" that swaps the values of @a and @b. The bottom status bar shows the line number as 5/9, a column number of 1, and a buffer size of #1[0] [0x0].

Inline assignation

- Consists of assigning a value to (or initializing) a variable before doing something else, in the same line
- Exploits the *everything returns something* rule
- Do not abuse it



A screenshot of a terminal window titled "a.rb + (~/Download...by/basic-ruby...)". The window contains the following Ruby code:

```
1 def inline_stuff(a, b)
2   d = (a ||= 3) + b + (c = 5)
3   puts "d is #{d}, but also c is #{c}"
4 end
```

The status bar at the bottom of the terminal window shows: "a.rb [+] Line:1/4 [25%] Col:1 Buf:#1 [100] [0x64]".

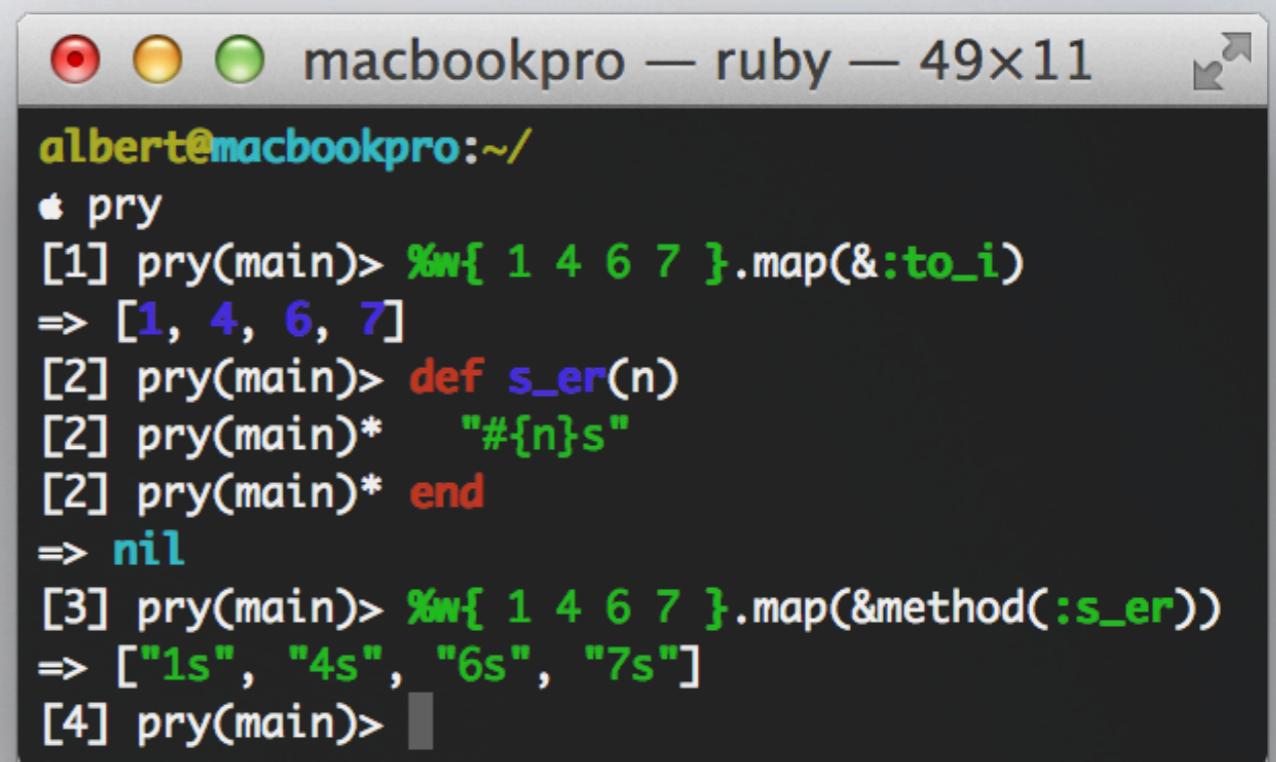
- A special (kind of) variable
- It can be used to skip assigning an array element to a variable
- In irb or pry, it is the last calculated value

```
macbookpro — ruby — 50x9
albert@macbookpro:~/ pry
[1] pry(main)> first, _, *others = [1,2,3,4,5,6]
=> [1, 2, 3, 4, 5, 6]
[2] pry(main)> first
=> 1
[3] pry(main)> others
=> [3, 4, 5, 6]
[4] pry(main)>
```

```
macbookpro — ruby — 48x7
albert@macbookpro:~/ pry
[1] pry(main)> a = [1,2,3]
=> [1, 2, 3]
[2] pry(main)> _
=> [1, 2, 3]
[3] pry(main)>
```

One-line iterators

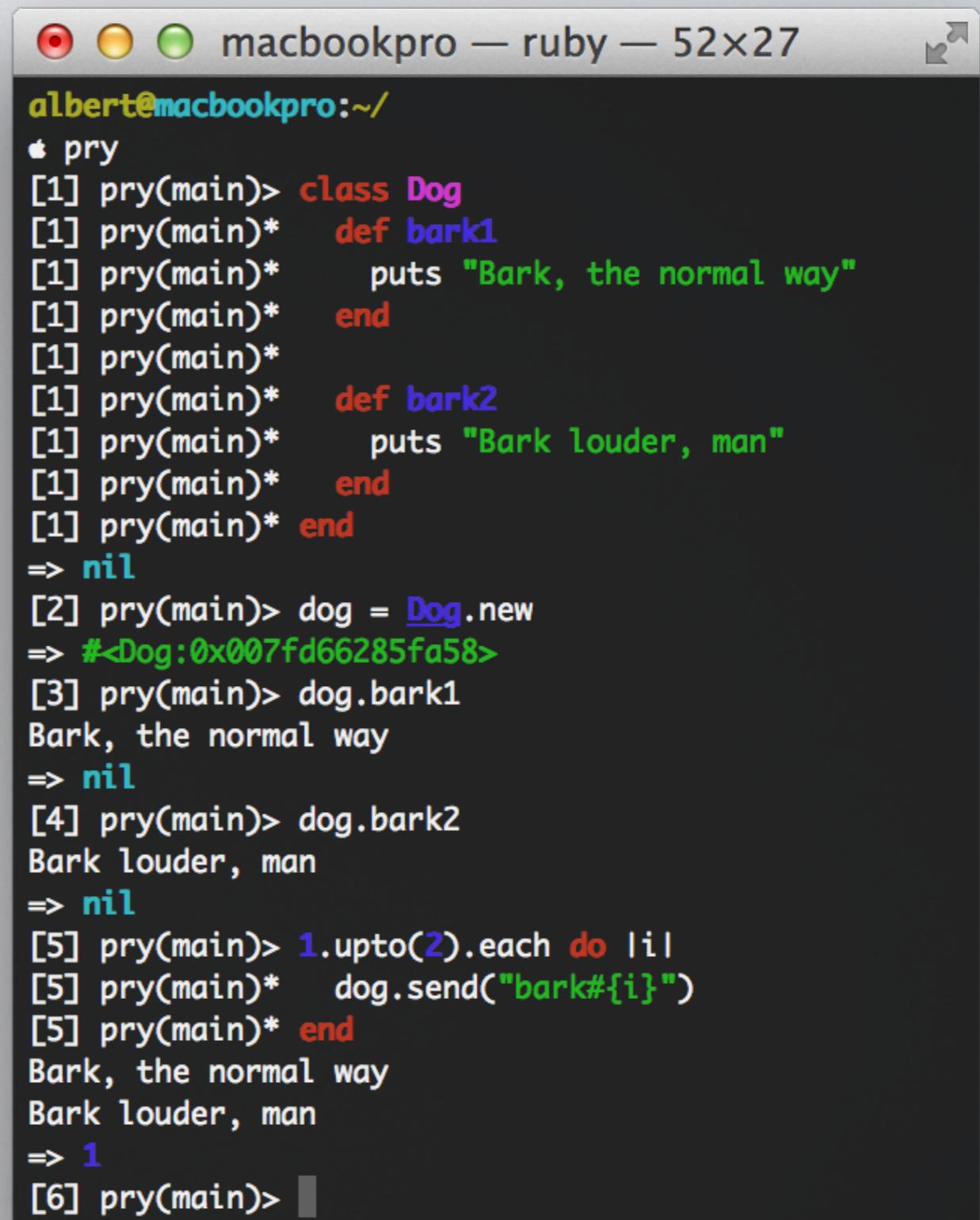
- Based on a collection of objects, they are used to
 - Call a method defined within the object
 - Call a method passing the object as a parameter
 - Remember my goal?



```
albert@macbookpro:~/  
• pry  
[1] pry(main)> %w{ 1 4 6 7 }.map(&:to_i)  
=> [1, 4, 6, 7]  
[2] pry(main)> def s_er(n)  
[2] pry(main)*   "#{n}s"  
[2] pry(main)* end  
=> nil  
[3] pry(main)> %w{ 1 4 6 7 }.map(&method(:s_er))  
=> ["1s", "4s", "6s", "7s"]  
[4] pry(main)>
```

send

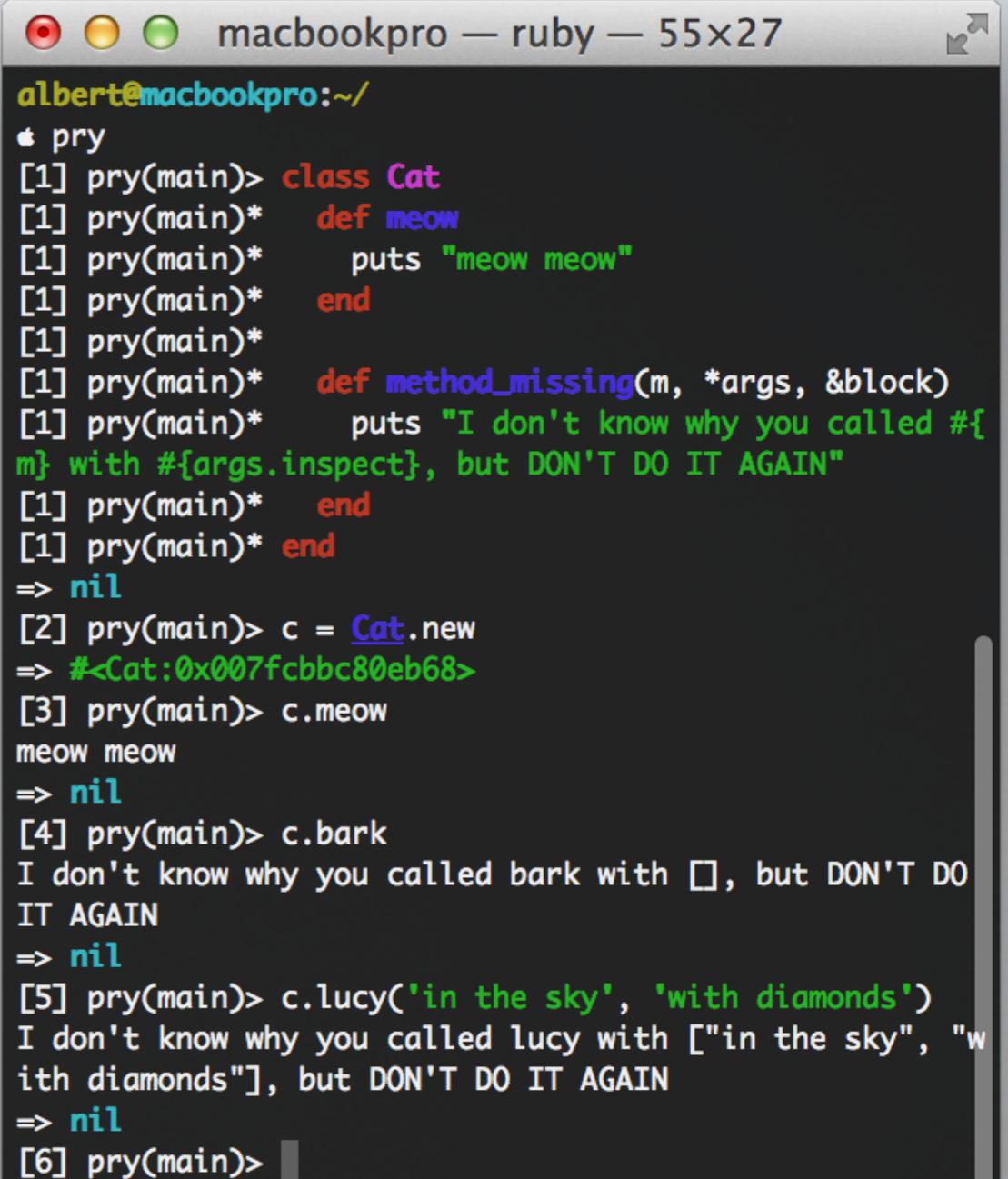
- It is like a gun
- A key to Ruby metaprogramming
- It allows us to call a method in an object, being able to specify the called method **in runtime**
- Very **powerful**



```
albert@macbookpro:~/  
$ pry  
[1] pry(main)> class Dog  
[1] pry(main)*   def bark1  
[1] pry(main)*     puts "Bark, the normal way"  
[1] pry(main)*   end  
[1] pry(main)*  
[1] pry(main)*   def bark2  
[1] pry(main)*     puts "Bark louder, man"  
[1] pry(main)*   end  
[1] pry(main)* end  
=> nil  
[2] pry(main)> dog = Dog.new  
=> #<Dog:0x007fd66285fa58>  
[3] pry(main)> dog.bark1  
Bark, the normal way  
=> nil  
[4] pry(main)> dog.bark2  
Bark louder, man  
=> nil  
[5] pry(main)> 1.upto(2).each do |i|  
[5] pry(main)*   dog.send("bark#{i}")  
[5] pry(main)* end  
Bark, the normal way  
Bark louder, man  
=> 1  
[6] pry(main)>
```

method_missing

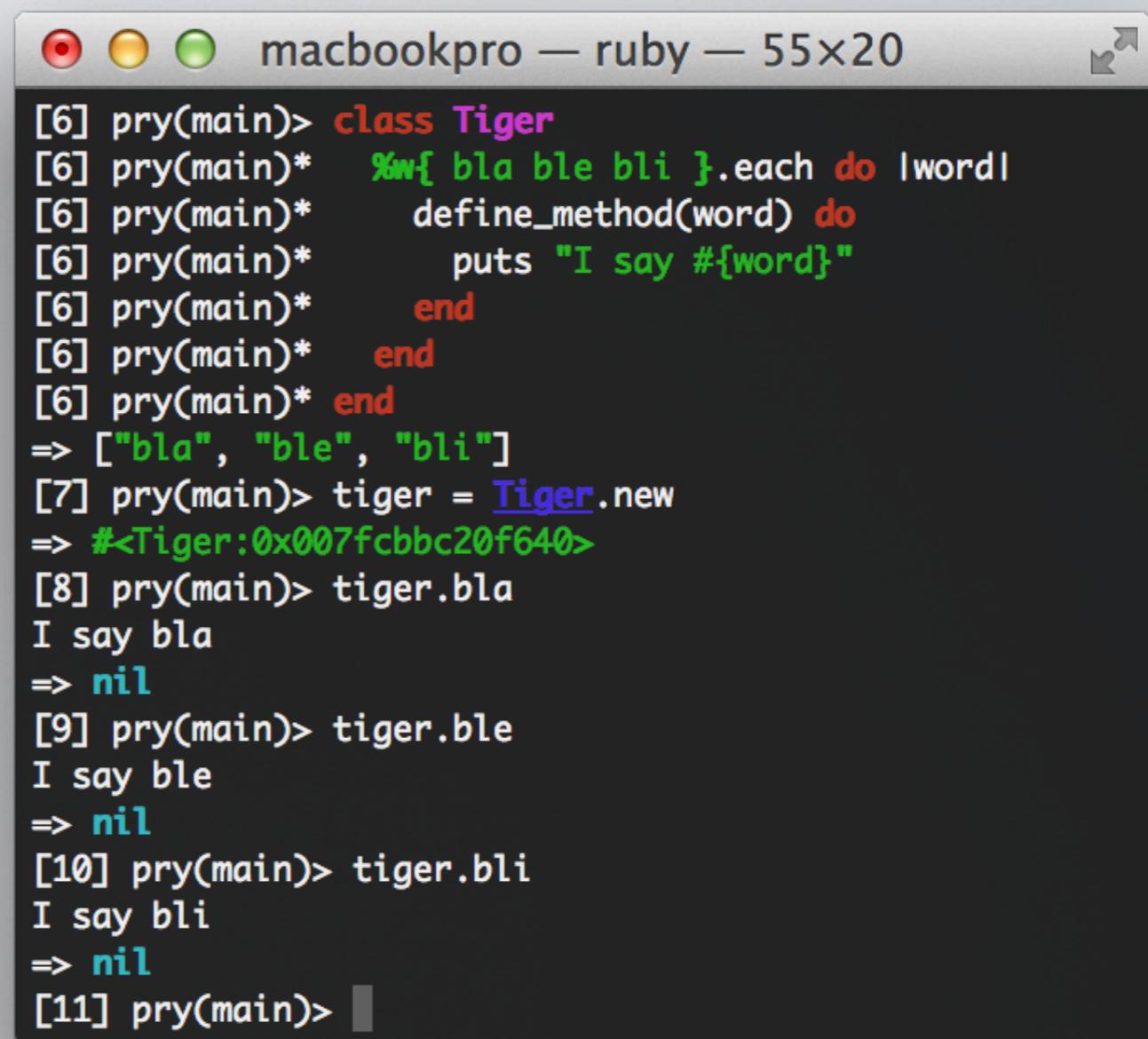
- When calling a method to an object, if there is no matching candidate, you can define a handler method for these occasions
- Also key for metaprogramming



```
albert@macbookpro:~/
$ pry
[1] pry(main)> class Cat
[1] pry(main)*   def meow
[1] pry(main)*     puts "meow meow"
[1] pry(main)*   end
[1] pry(main)*
[1] pry(main)*   def method_missing(m, *args, &block)
[1] pry(main)*     puts "I don't know why you called #{m} with #{args.inspect}, but DON'T DO IT AGAIN"
[1] pry(main)*   end
[1] pry(main)* end
=> nil
[2] pry(main)> c = Cat.new
=> #<Cat:0x007fcbbc80eb68>
[3] pry(main)> c.meow
meow meow
=> nil
[4] pry(main)> c.bark
I don't know why you called bark with □, but DON'T DO
IT AGAIN
=> nil
[5] pry(main)> c.lucy('in the sky', 'with diamonds')
I don't know why you called lucy with ["in the sky", "w
ith diamonds"], but DON'T DO IT AGAIN
=> nil
[6] pry(main)>
```

define_method

- Allows us to define methods in runtime
- This way we can reduce similar definitions, and avoid duplicating code



```
[6] pry(main)> class Tiger
[6] pry(main)*   %w{ bla ble bli }.each do |word|
[6] pry(main)*     define_method(word) do
[6] pry(main)*       puts "I say #{word}"
[6] pry(main)*     end
[6] pry(main)*   end
[6] pry(main)* end
=> ["bla", "ble", "bli"]
[7] pry(main)> tiger = Tiger.new
=> #<Tiger:0x007fcbbc20f640>
[8] pry(main)> tiger.bla
I say bla
=> nil
[9] pry(main)> tiger.ble
I say ble
=> nil
[10] pry(main)> tiger.bli
I say bli
=> nil
[11] pry(main)>
```

A bit of **Ruby on Rails**

Ruby on Rails

- We are really fast
- It is focused mainly in two principles:
 - Don't repeat yourself (**DRY**)
 - Convention over configuration (**CoC**)
- Main Ruby web framework
- Others: **Sinatra**, **Padrino**, and many more

Ruby on Rails

- Famous for being able to create a minimum blog in about 20 minutes
- Let's see it **in person** here*

http://guides.rubyonrails.org/getting_started.html

Continue learning

Continue learning

- *Beginning Ruby, From Novice to Professional*, by Peter Cooper (<http://beginningruby.org/>)
- *Programming Ruby 1.9 & 2.0: The Pragmatic Programmer's Guide*, by D.Thomas, C. Fowler and Andy Hunt (<http://pragprog.com/book/ruby4/programming-ruby-1-9-2-0>)
- *Metaprogramming Ruby: Program Like the Ruby Pros*, by Paolo Perrotta (<http://pragprog.com/book/ppmetr/metaprogramming-ruby>)

Continue learning

- rubylang.org
- mislav.uniqpath.com/poignant-guide/
- tryruby.org
- humblelittlerubybook.com/
- ruby.learncodethehardway.org/

**Contact me
if you have any doubt**

And that would be all