wifi: ltnig 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**


RUBY

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 languagues 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

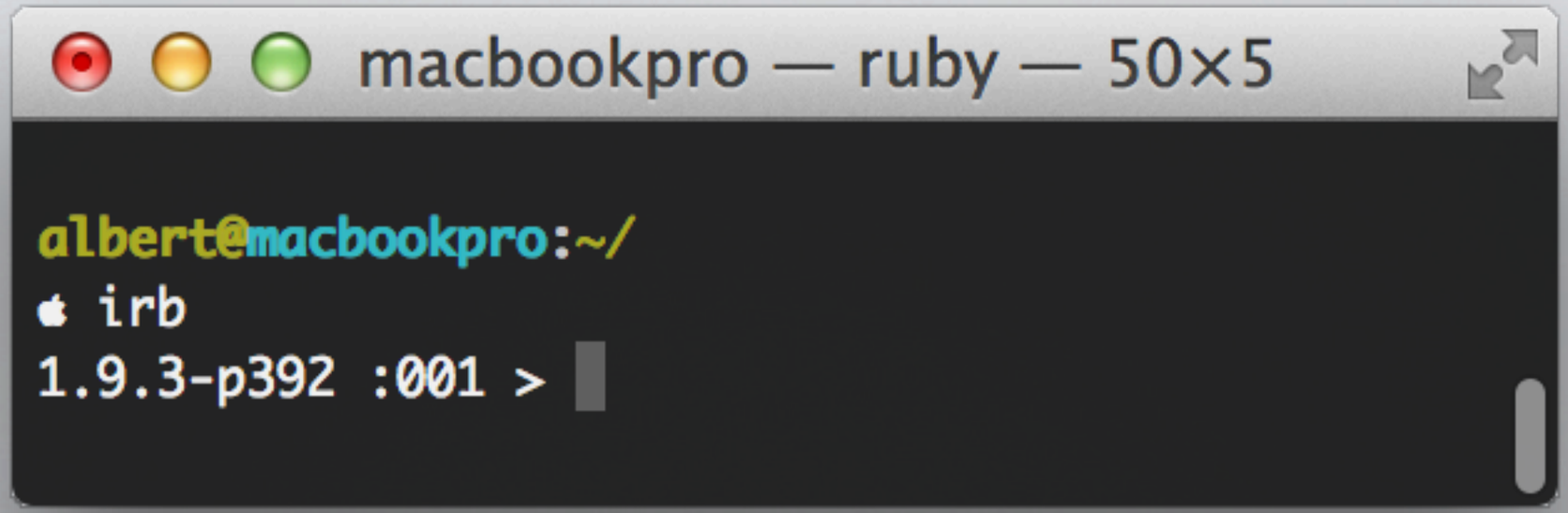
- Really active community, evolving the language and its gems in a daily basis

- 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

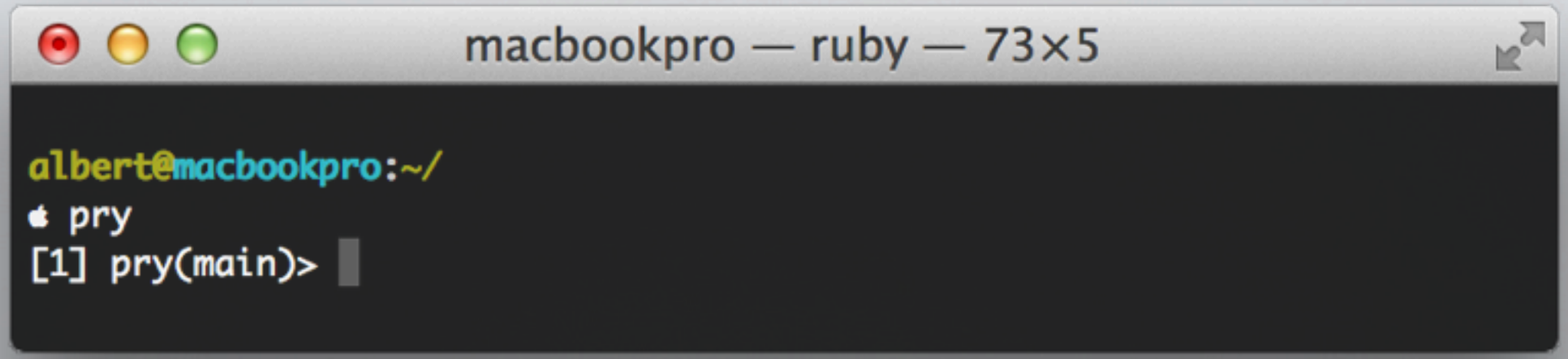
Your friend, the
Interactive Ruby Shell (irb)



Try it!

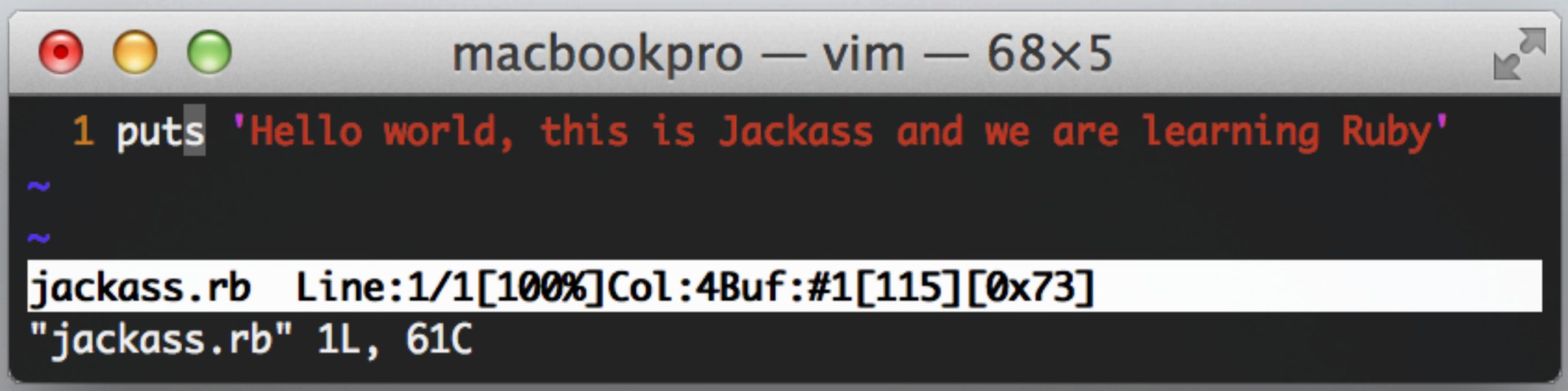
Meet pry too

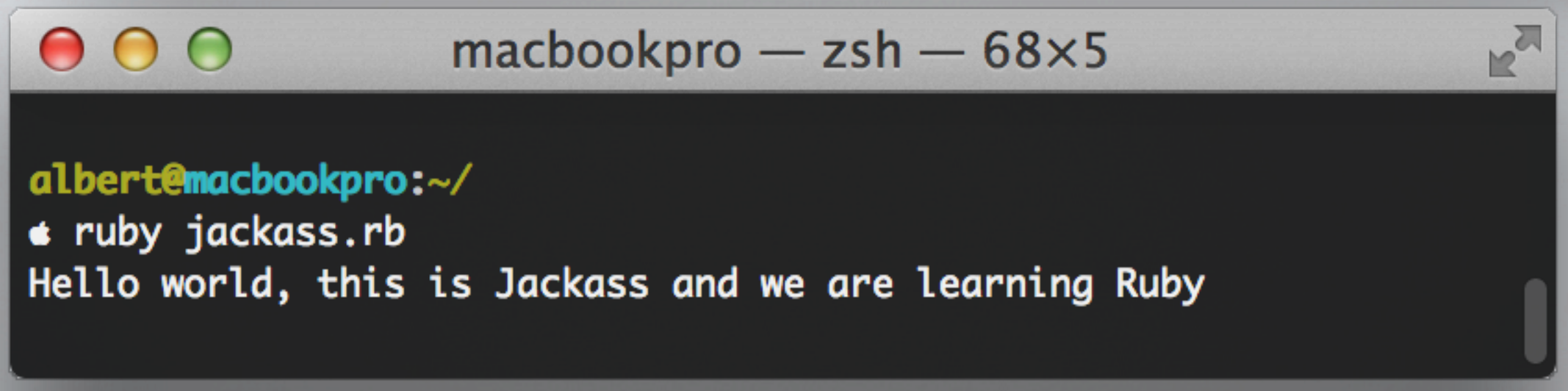
An alternative to the standard Ruby shell



And give it a shot!

Executing some Ruby





Executing more Ruby

macbookpro — vim — 69×6

```
  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
```

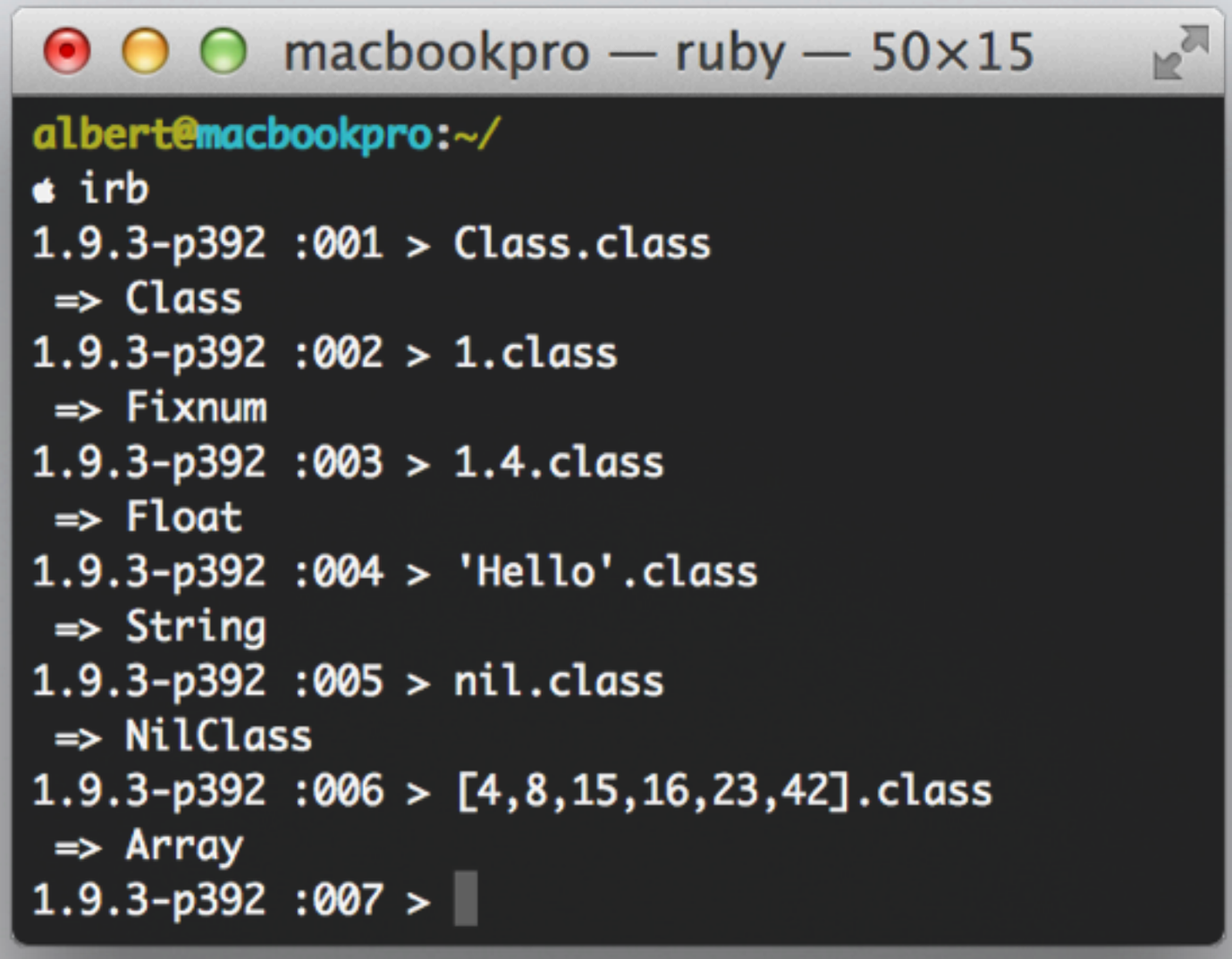
macbookpro — zsh — 69×6

```
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



...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.ka. 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

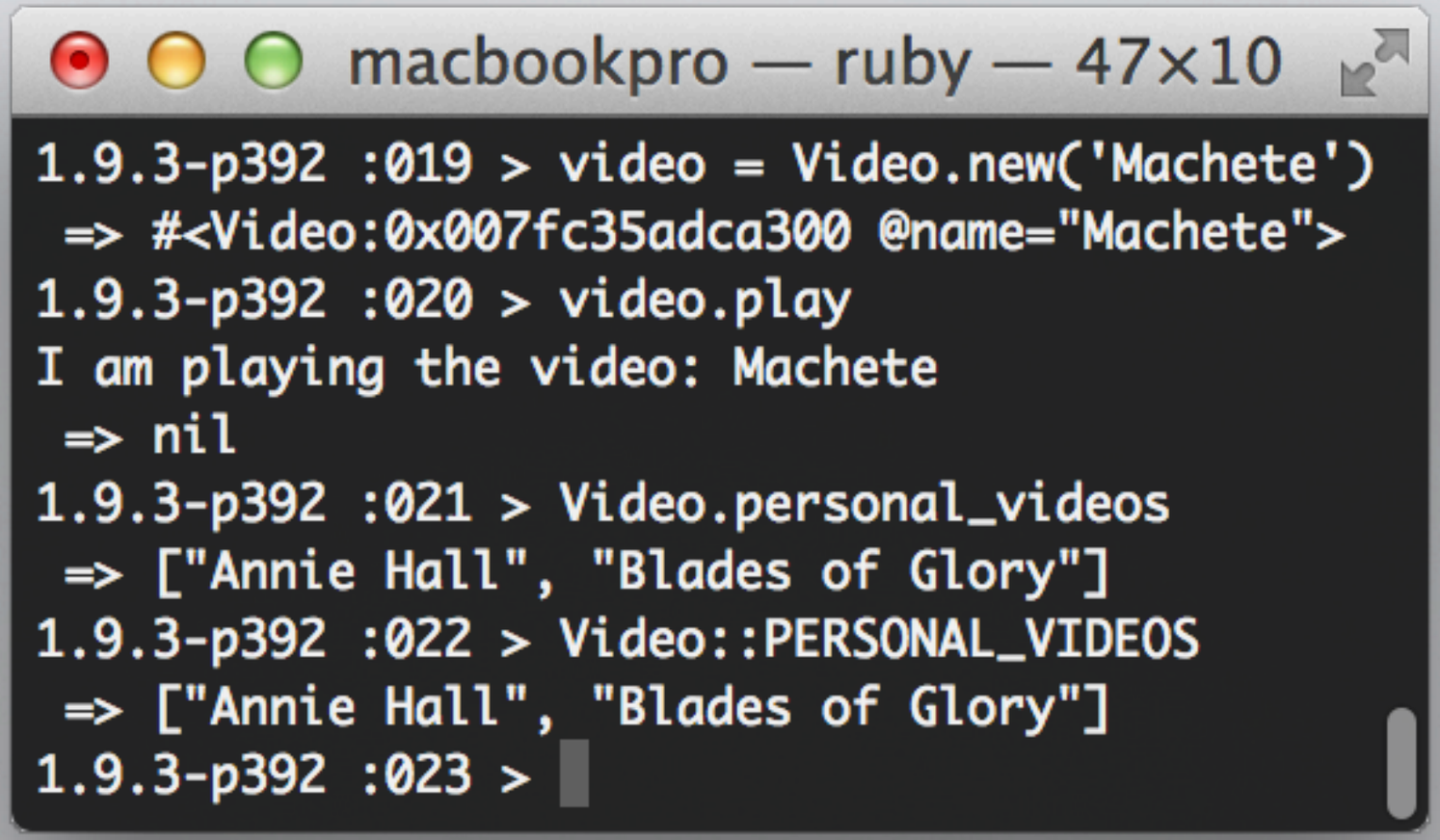
- A class example

test.rb + (~/Downloads/Practicing Ruby) - VIM2

```
 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
../test.rb [+] Line:1/15[6%]Col:1Buf:#1[99][0x63]
```

Essentials

- A class example



Exercices time!

Exercises 1-2 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 an 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 preceeded 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

```
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]
```

- We also have `unless..end`

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

- They can be inline

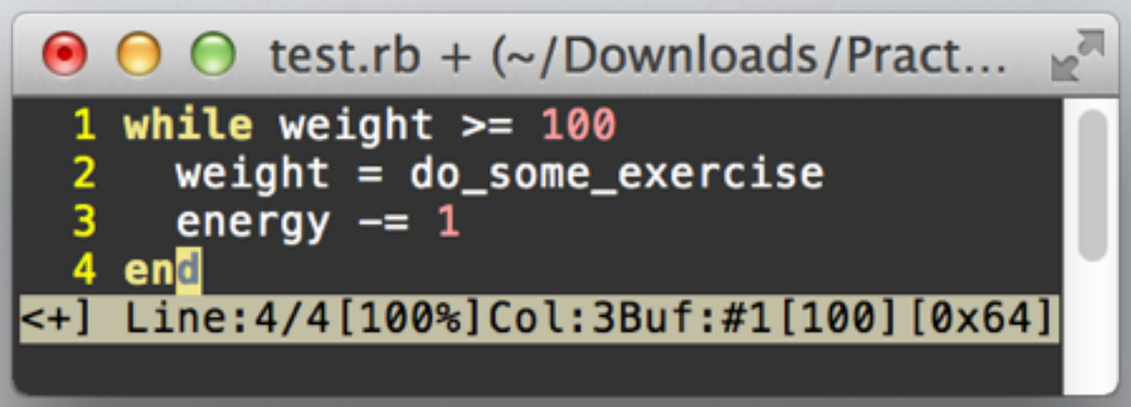
```
test.rb + (~/Downloads/Pract...
1 puts 'Hell yeah' if it_is_friday
~
~
<] Line:1/1[100%]Col:32Buf:#1[121][0x79]
```

- They return the final line of the branch

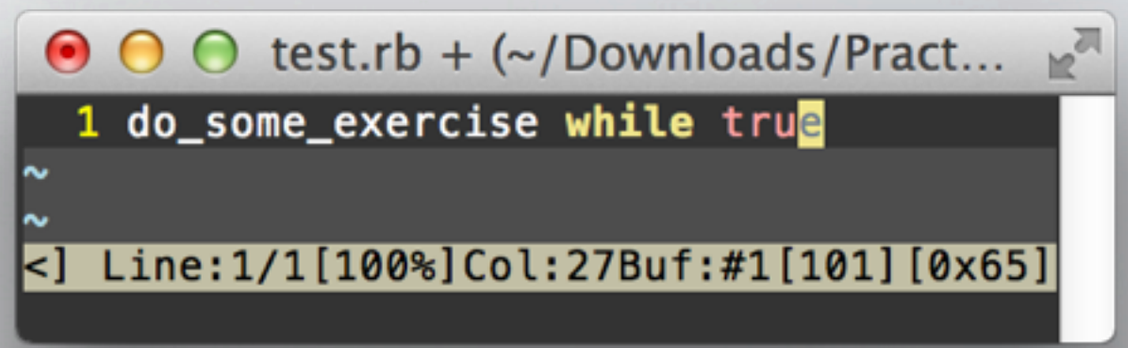
```
test.rb + (~/Downloads/Pract...
1 result = if a == 1
2            2
3          else
4            3
5          end
<] Line:5/5[100%]Col:10Buf:#1[101][0x65]
```

Control structures: while

- Again, pretty standard
- Can also be inline



- I used them once in three years, and it could be done without it



Control structures: case

- I won't lie... pretty standard

- Returns the corresponding case value, or nil if not found

```
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]
```

Regular expressions

- A way of specifying a pattern of characters to be matched in a string

- Typically written between slashes: `/a(b|c)de?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

test.rb + (~/Downloads/Prac…

```
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]
```

```
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

test.rb + (~/Downloads/Practici…

```
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
< [+] 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)

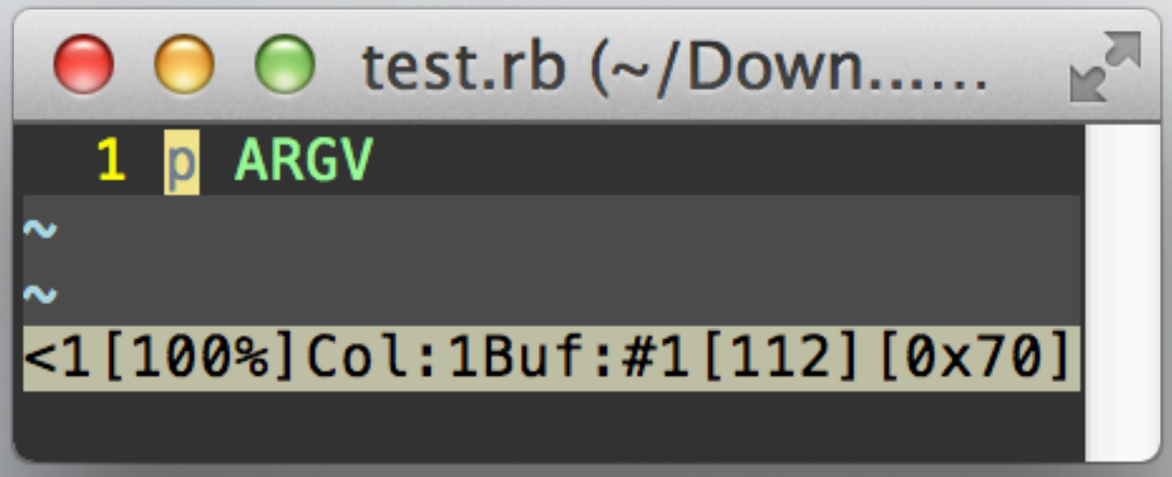
- `printf` prints the argument(s) based on a format string (like Perl or C)

- `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




```
 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

```ruby
require 'date'

class Person
  def initialize(name, surnames, birth_year)
    @name = name
    @surnames = surnames
    @birth_year = birth_year
  end

  def age
    Date.today.year - @birth_year
  end

  def age=(new_age)
    @birth_year = Date.today.year - new_age
  end

  def complete_name
    "#{@name} #{@surnames}"
  end
end
```

```
test.rb [+] Line:21/21[100%]Col:1Buf:#1[101][0x65]
```

macbookpro — ruby — 97×15

```
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

```ruby
require 'date'
require_relative 'age_calculator.rb'

class Person
  def initialize(name, surnames, birth_year)
    @name = name
    @surnames = surnames
    @birth_year = birth_year
  end

  def age
    AgeCalculator.new(@birth_year).age
  end

  def age=(new_age)
    @birth_year = Date.today.year - new_age
  end

  def complete_name
    "#{@name} #{@surnames}"
  end
end
```

test.rb [+] Line:1/22[4%]Col:1Buf:#1[114][0x72]

test.rb + (~/Downloads/Curs d...

```ruby
class AgeCalculator
  def initialize(birth_year)
    @birth_year = birth_year
  end

  def age
    Date.today.year - @birth_year
  end
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 occessible to our own class

Classes and objects

test.rb (~/Downloads/Curs de Ruby) - VIM1

```ruby
class Person
  def initialize(name, surnames, bank_password)
    @name = name
    @surnames = surnames
    @bank_password = bank_password
  end

  def complete_name
    "#{name} #{surnames}"
  end

  protected

  def name_data
    [@name, @surnames]
  end

  private

  def bank_password_giveaway
    "My bank password is #{@bank_password}"
  end
end
```

```
test.rb  Line:23/23[100%]Col:1Buf:#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 delets 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

test.rb + (~/Downl...

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

test.rb + (~/Downl...

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

- For example, wee 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"
```

- 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 1/0 or adding two fixnums into a bignum)

- If the numbers are from different classes, the most general class is used:
  - `1 + 2 => 3`
  - `1.0 + 5 => 6.0`
  - `3 + Rational(2,3) => 11/3`
  - `2.3 + Rational(1,6) => 2.466666666666663`

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

```
irb
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
 => "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 of specialize a certain class, thus being a subclass

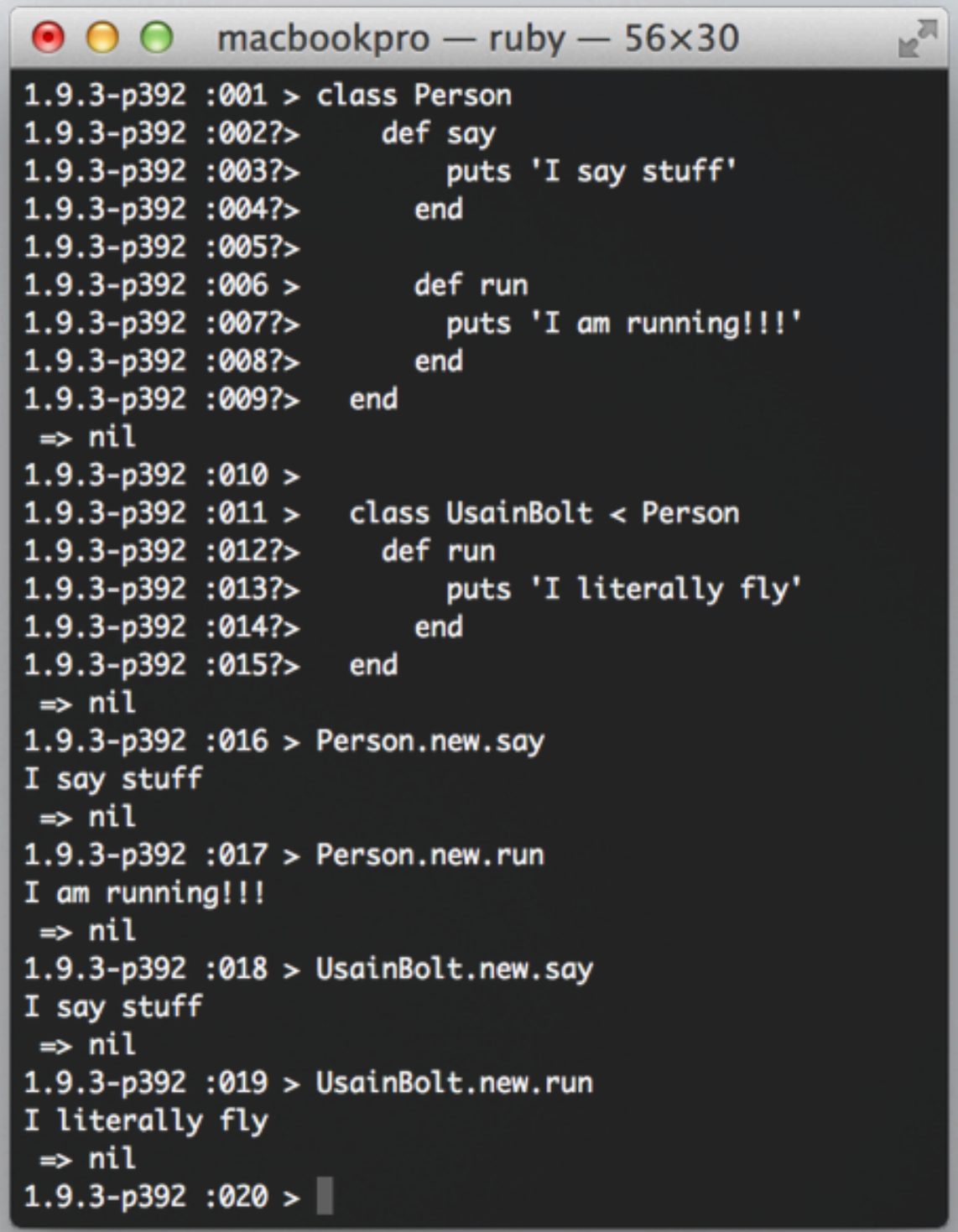
- The subclass inherits everyting (methods, state)

- 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



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 inherints 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 appliable to several places

Inheritance and modules

- Example

```
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
```

Exceptions

- Errors **always** happen, sooner or better
- We have to manage them properly
- Return codes are the traditional way, but difficult to maintain and standarize
- **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 handled different exceptions differently
- We can use `else` for non-exception cases
- Finally, `ensure` gets executed either way

Exceptions

a.rb (~) – VIM

```
 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"
~
a.rb  Line:1/13[7%]Col:1Buf:#1[98][0x62]
```

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.rb + (~/code/quipuapp) - VIM2

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

a.rb [+] Line:1/9[11%]Col:1Buf:#1[98][0x62]

Exercices time!

All 4 exercises from other stuff

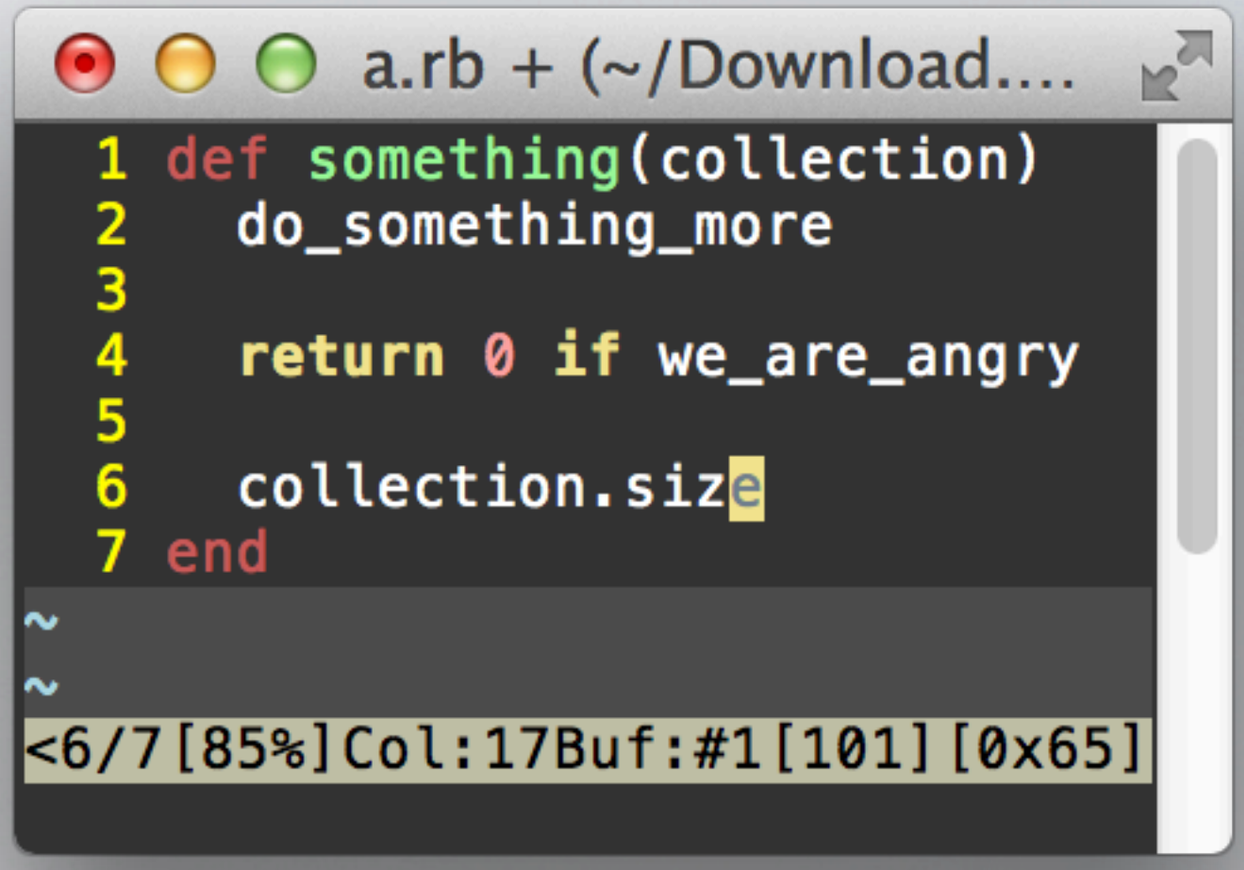
Tricks and differential treats

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`)

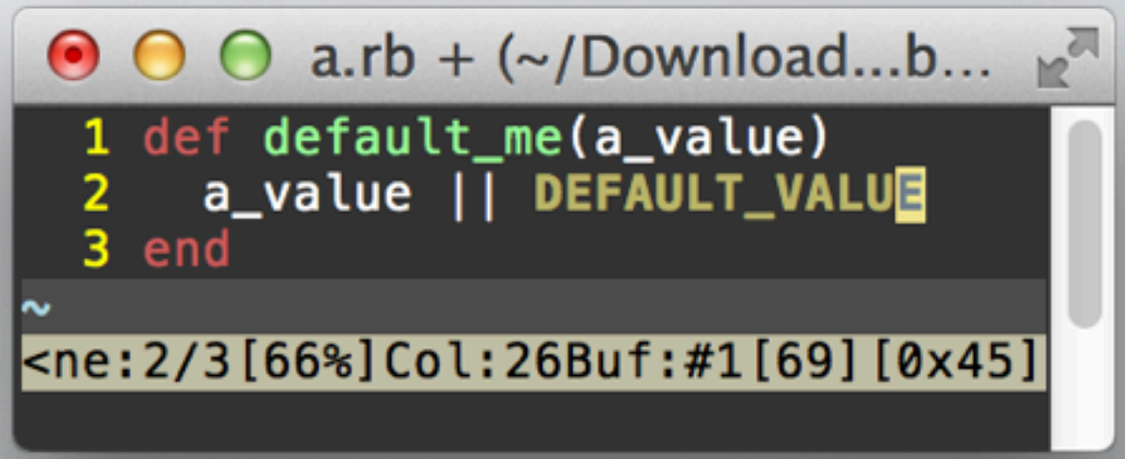
- Less and more beautiful code!



Use the ORs in assignations

- Using ORs can help us write less code too, in assignations

- For example, to provide default values to variables



Memoization

- Basically caches the result of a method

- And there is no second point

a.rb + (~/Download...by/basic-ruby) - VIM

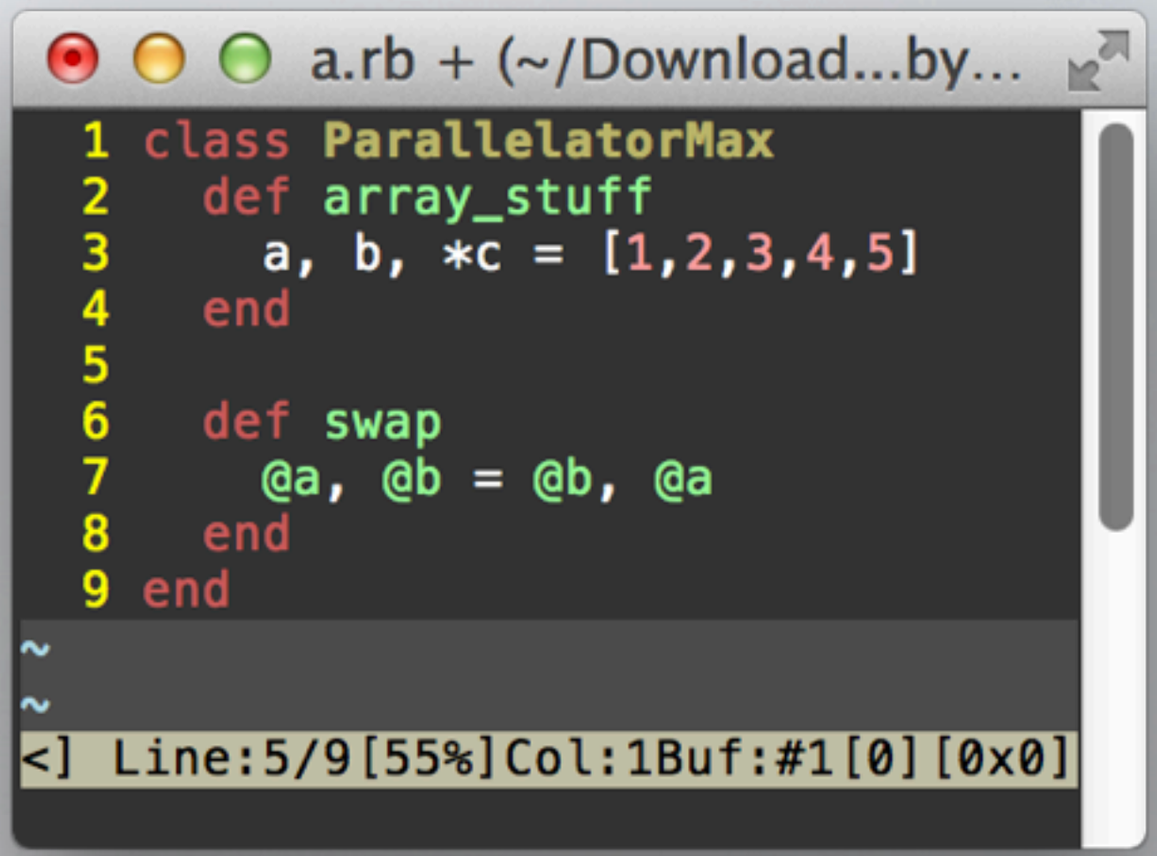
```
 1 class EfficientFolk
 2   def do_something_simple
 3     @do_something_simple ||= calculation
 4   end
 5
 6   def do_something_more_complex(value)
 7     @do_something_more_complex ||= {}
 8     @do_something_more_complex[value] ||= begin
 9                                             calculation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

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

Parallel assignation

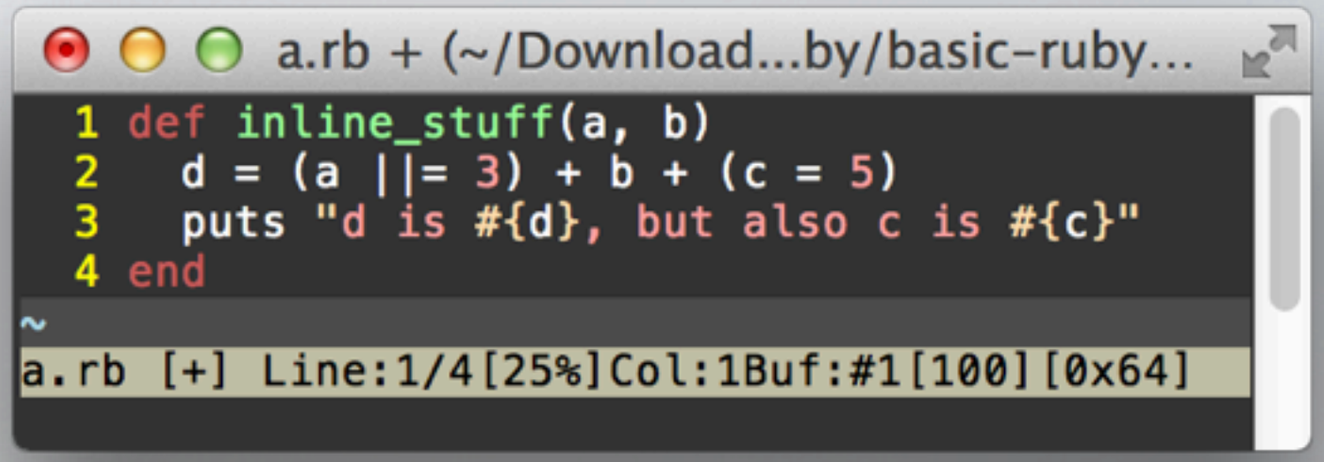
- Allows multiple assignations in one line

- Relieves us from using the damn middle variable when swapping



Inline assignation

- Consists of assigning a value to (or initializing) a variable before doing something else, in the same line

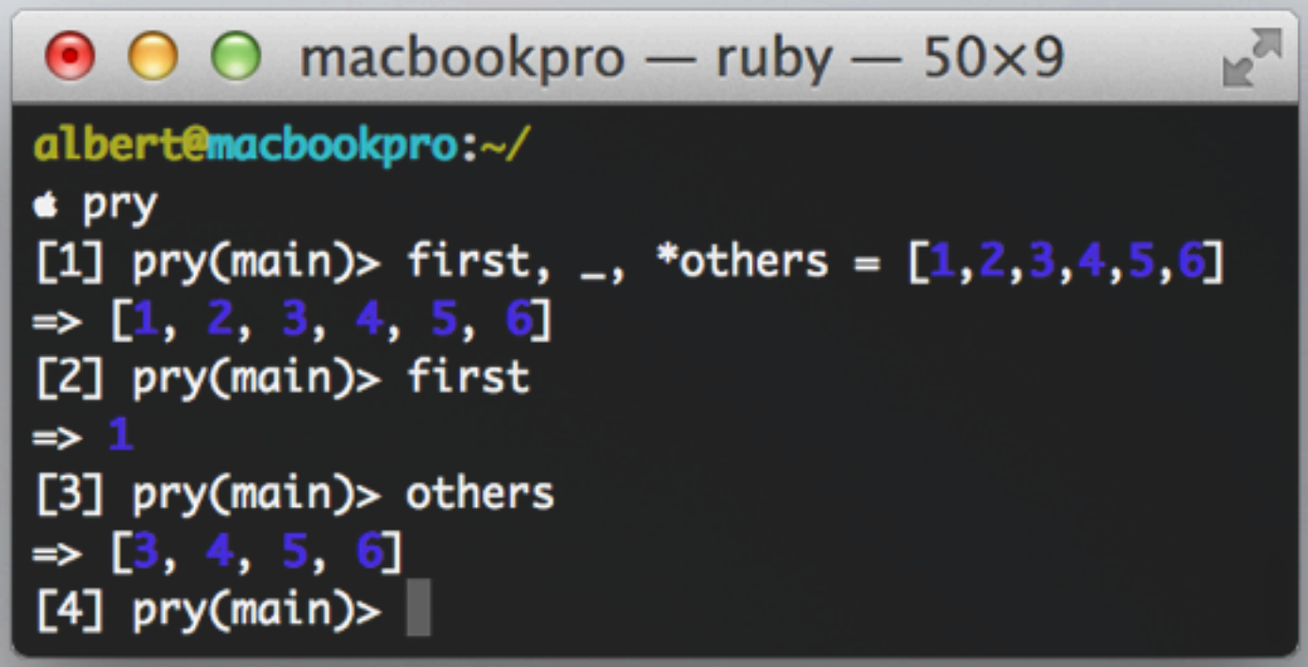


- Exploits the *everything returns something* rule

- Do not abuse it

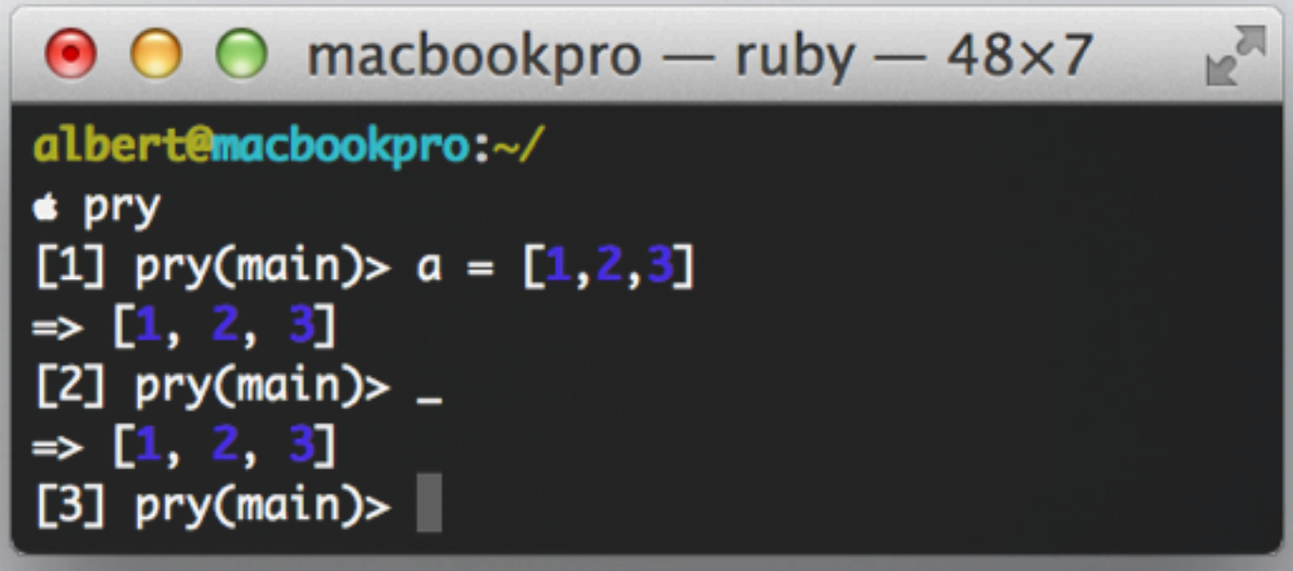
_

- A special (kind of) variable



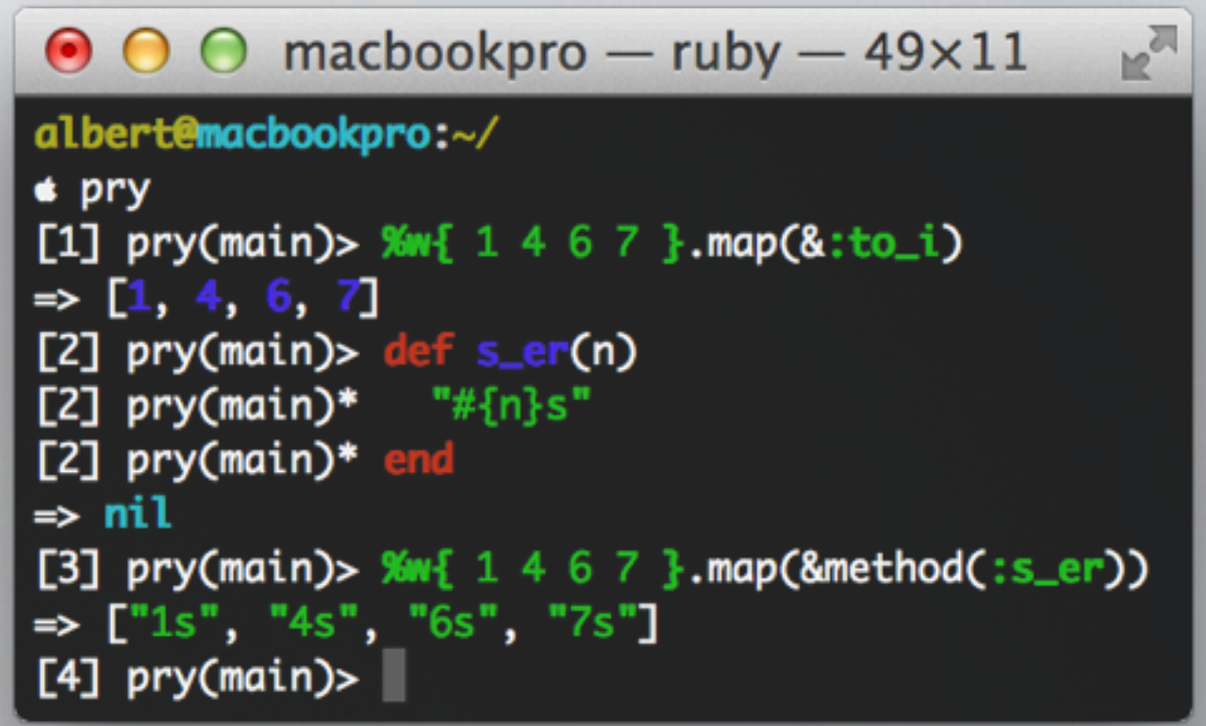
- It can be used to skip assigning an array element to a variable

- In irb or pry, it is the last calculated value



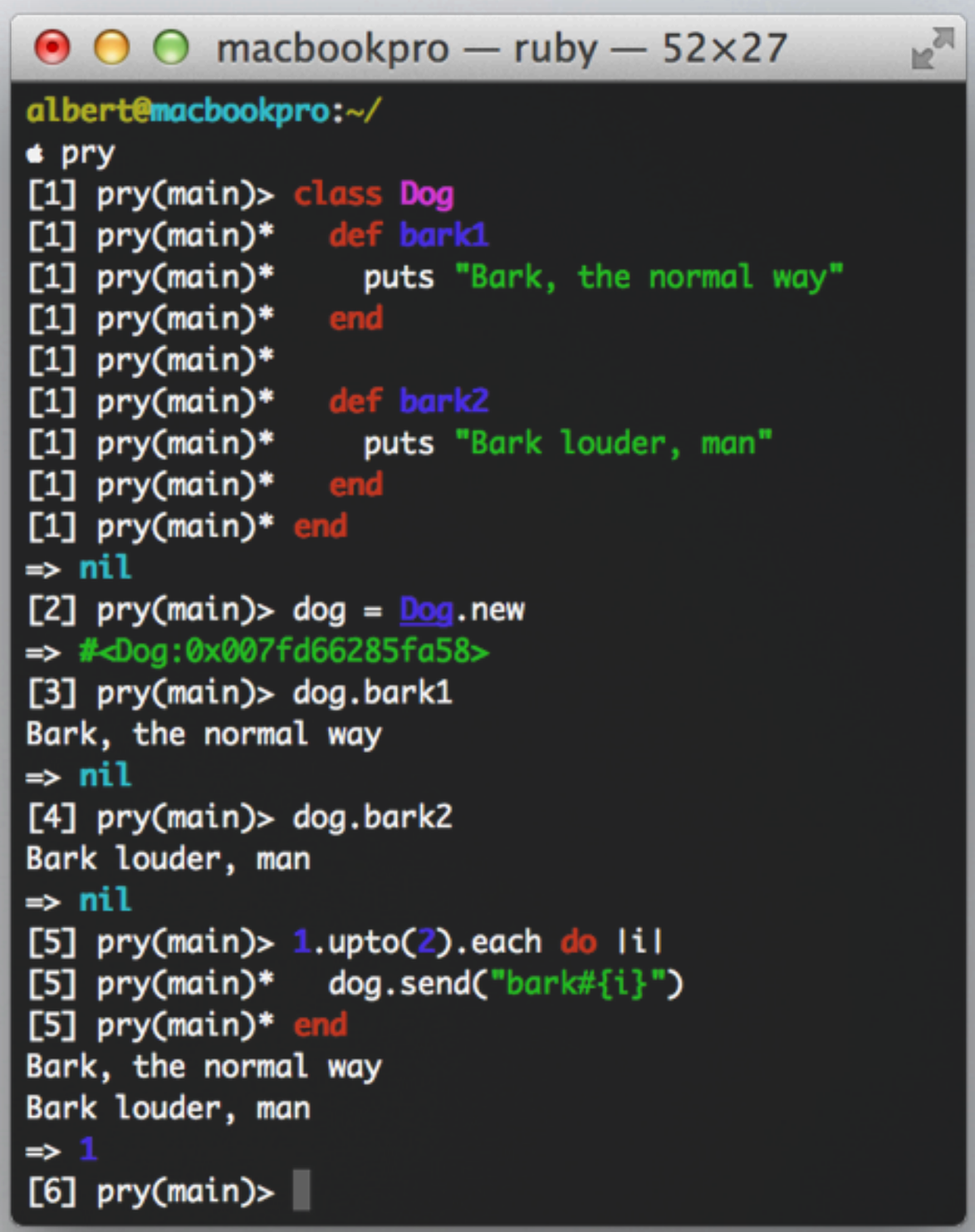
One-line iterators

- 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?*



send

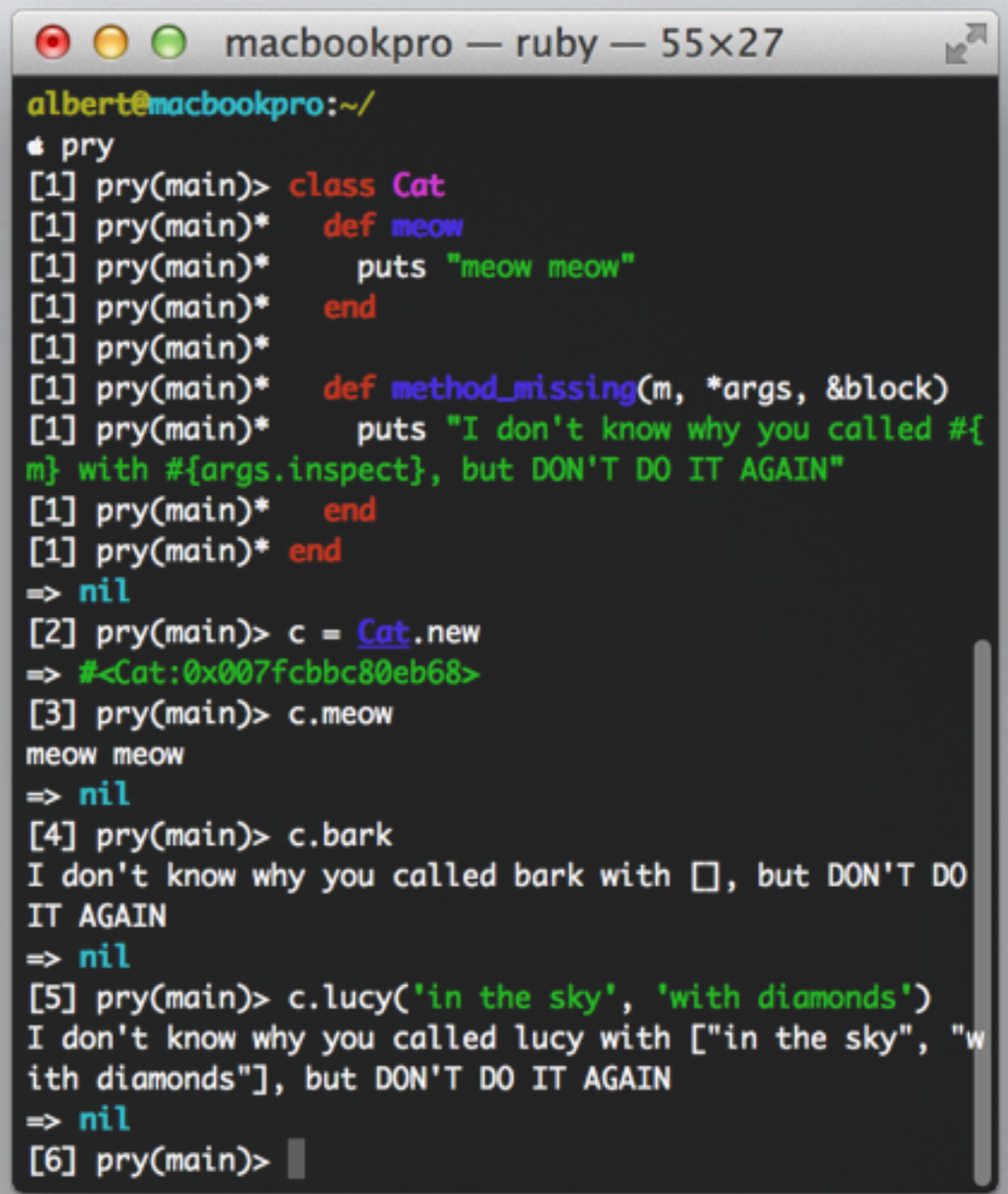
- It is like a gun
- A key to Ruby metaprogramming
- It allows us to call a method in an object, being able to specify the called method **in runtime**
- Very **powerful**



method_missing

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

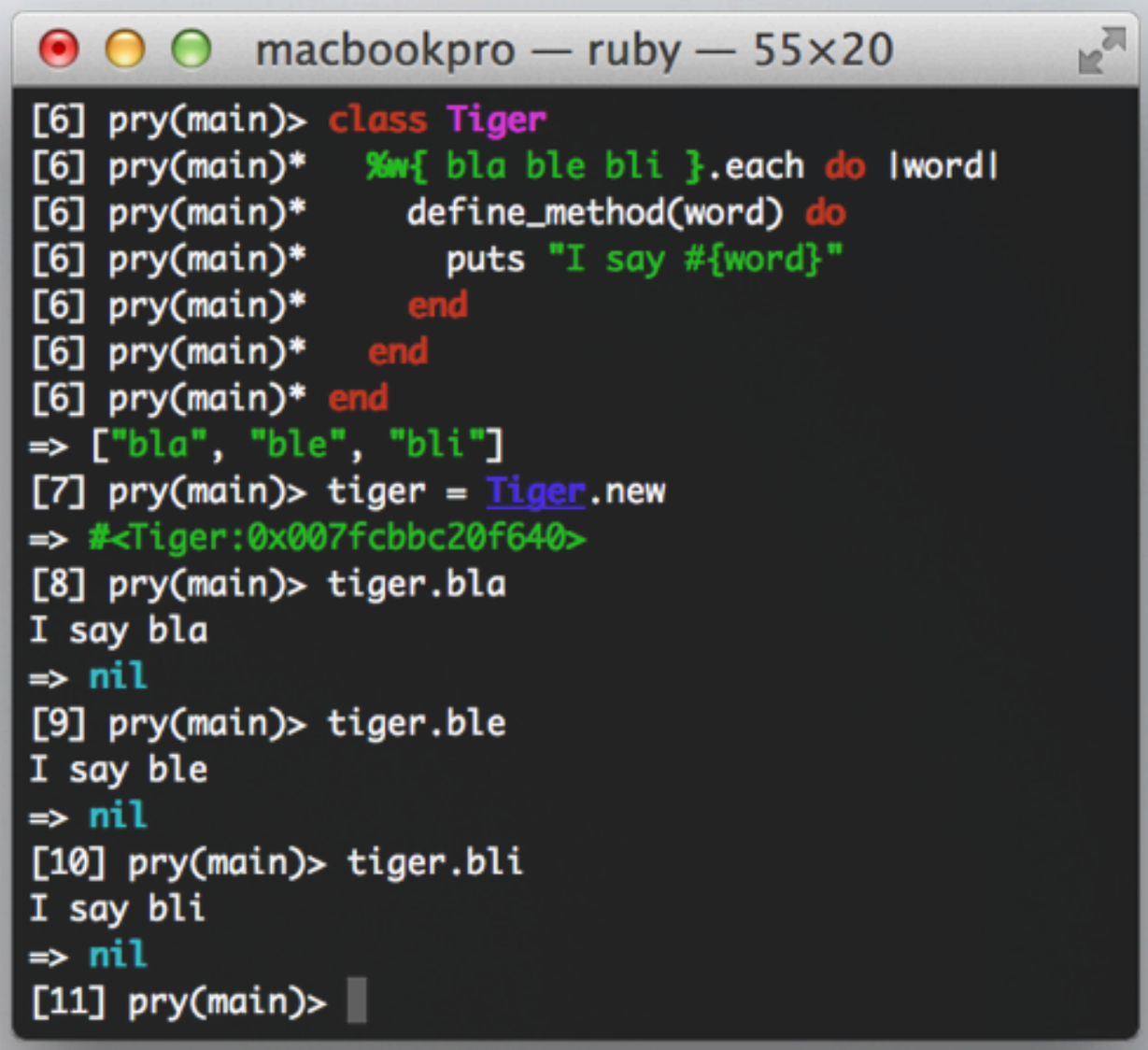
- Also key for metaprogramming



define_method

- Allows us to define methods in runtime

- This way we can reduce similar definitions, and avoid duplicating code



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