



www.inotechne.com
Copyright ©2012 Inotechne. All Rights Reserved.

Richard Piacentini
Chief Technical Officer & Founder

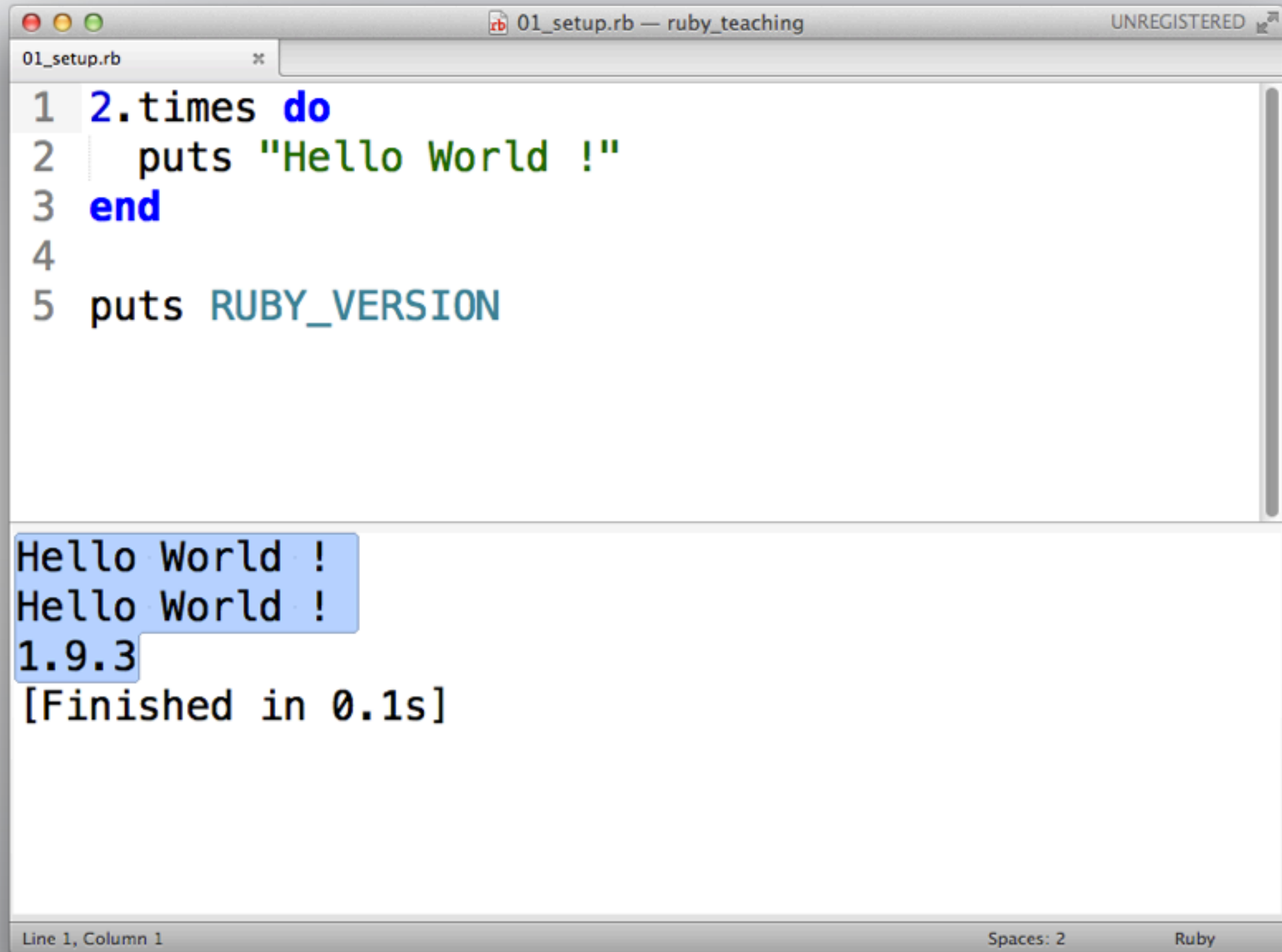
Pratompol Jiravanitchakorn
Ruby Developer

Monsiree Thassanakathitum
Ruby Developer

Virudson Thitilertdecha
Ruby Developer

SETUP

- Create a new directory on your Desktop: **ruby_teaching**
- Launch **Sublime Text Editor**
- Create a new file **ruby_teaching\code.rb**
- Save the file (CTRL + S)



```
01_setup.rb
1 2.times do
2    puts "Hello World !"
3  end
4
5 puts RUBY_VERSION
```

Hello World !
Hello World !
1.9.3
[Finished in 0.1s]

Line 1, Column 1 Spaces: 2 Ruby

INTRODUCTION TO RUBY



Kasetsart University, Bangkok
July 19th, 2012

TABLE OF CONTENTS

- Introduction
- Variables
- Methods
- Classes
- More about Classes
- Modules
- Constants

INTRODUCTION

puts "Sawasdee Krub !"

puts "Sawasdee Krub !"
Sawasdee Krub !

WHAT IS RUBY ?

WHAT IS RUBY ?

- Dynamic **Object Oriented** Programming Language

WHAT IS RUBY ?

- Dynamic **Object Oriented** Programming Language
- Inspired by **Perl** and **Smalltalk**

WHAT IS RUBY ?

- Dynamic **Object Oriented** Programming Language
- Inspired by **Perl** and **Smalltalk**
- Influenced by **Eiffel** and **Lisp**

WHAT IS RUBY ?

WHAT IS RUBY ?

- Created in Japan by **Yukihiro “Matz” Matsumoto**

WHAT IS RUBY ?

- Created in Japan by **Yukihiro “Matz” Matsumoto**
- First release on **December 21, 1995**

WHAT IS RUBY ?

- Created in Japan by **Yukihiro “Matz” Matsumoto**
- First release on **December 21, 1995**
- Standard implementation is called **“MRI”**, and is written in **C**

WHAT IS RUBY ?

- Created in Japan by **Yukihiro “Matz” Matsumoto**
- First release on **December 21, 1995**
- Standard implementation is called **“MRI”**, and is written in **C**
- **JRuby** is another popular implementation, written in **Java**

ADVANTAGES OF RUBY

ADVANTAGES OF RUBY



ADVANTAGES OF RUBY

☒ Object Oriented



ADVANTAGES OF RUBY

 Object Oriented



 Managed memory

ADVANTAGES OF RUBY

☒ Object Oriented



☒ Managed memory

☒ General purpose

ADVANTAGES OF RUBY

☒ Object Oriented



☒ Managed memory

☒ General purpose

☒ Builtin REPL

ADVANTAGES OF RUBY

☒ Object Oriented



☒ Managed memory

☒ General purpose

☒ Simple and elegant

☒ Builtin REPL

ADVANTAGES OF RUBY

☒ Object Oriented



☒ Interpreted

☒ Managed memory

☒ General purpose

☒ Simple and elegant

☒ Builtin REPL

ADVANTAGES OF RUBY

☒ Object Oriented

☒ Dynamic

☒ Interpreted

☒ Simple and elegant



☒ Managed memory

☒ General purpose

☒ Builtin REPL

PHILOSOPHY OF RUBY ?

PHILOSOPHY OF RUBY ?



PHILOSOPHY OF RUBY

PHILOSOPHY OF RUBY

- Designed for **programmer's productivity**

PHILOSOPHY OF RUBY

- Designed for **programmer's productivity**
- Follow the **Principle of least surprise** (POLS)

PHILOSOPHY OF RUBY

- Designed for **programmer's productivity**
- Follow the **Principle of least surprise** (POLS)
- Focus on the **programmer**, not the computer !

FEATURES OF RUBY

FEATURES OF RUBY

- Truly Object Oriented...**Everything is an object**

FEATURES OF RUBY

- Truly Object Oriented...**Everything is an object**
- **Compact** and **flexible** syntax

FEATURES OF RUBY

- Truly Object Oriented...**Everything is an object**
- **Compact** and **flexible** syntax
- **Memory** is **managed** by interpreter through GC

FEATURES OF RUBY

- Truly Object Oriented...**Everything is an object**
- **Compact** and **flexible** syntax
- **Memory** is **managed** by interpreter through GC
- **Dynamic Typing** (type checking performed at run-time)

FEATURES OF RUBY

- Truly Object Oriented...**Everything is an object**
- **Compact** and **flexible** syntax
- **Memory** is **managed** by interpreter through GC
- **Dynamic Typing** (type checking performed at run-time)
- **Duck Typing** (what you can do is more important than who you are !)

EVERYTHING IS OBJECT

EVERYTHING IS OBJECT

- 1.class

EVERYTHING IS OBJECT

- `1.class`
- `(1.5).class`

EVERYTHING IS OBJECT

- `1.class`
- `(1.5).class`
- `(2*5.0).class`

EVERYTHING IS OBJECT

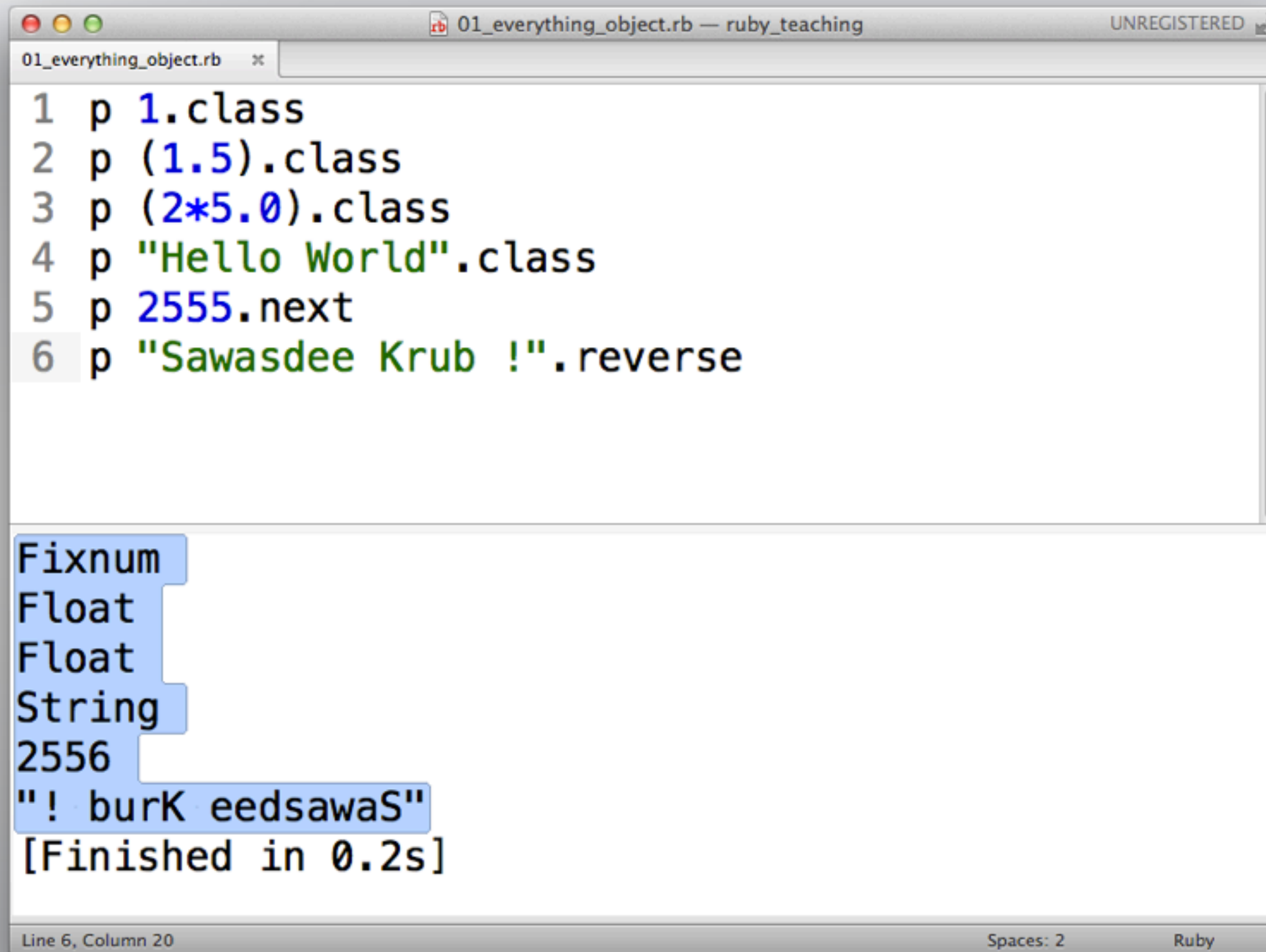
- `1.class`
- `(1.5).class`
- `(2*5.0).class`
- `"Hello World".class`

EVERYTHING IS OBJECT

- `1.class`
- `(1.5).class`
- `(2*5.0).class`
- `"Hello World".class`
- `2555.next`

EVERYTHING IS OBJECT

- `1.class`
- `(1.5).class`
- `(2*5.0).class`
- `"Hello World".class`
- `2555.next`
- `"Sawasdee Krub !".reverse`



The screenshot shows a Ruby IDE window titled "01_everything_object.rb — ruby_teaching" with a tab for "01_everything_object.rb". The script contains six lines of code. The output area shows the results of these lines: Fixnum, Float, Float, String, 2556, and the reversed string "! burK eedsawaS". The status bar at the bottom indicates "Line 6, Column 20", "Spaces: 2", and "Ruby".

```
1 p 1.class
2 p (1.5).class
3 p (2*5.0).class
4 p "Hello World".class
5 p 2555.next
6 p "Sawasdee Krub !".reverse
```

Fixnum
Float
Float
String
2556
"! burK eedsawaS"
[Finished in 0.2s]

Line 6, Column 20 Spaces: 2 Ruby

CODING STYLE

CODING STYLE

- Use **SPACES** to **indent** your code

CODING STYLE

- Use **SPACES** to **indent** your code
- Never use **TABS**

CODING STYLE

- Use **SPACES** to **indent** your code
- Never use **TABS**
- The rule is to use 2 spaces per **indent**

CODING STYLE

- Use **SPACES** to **indent** your code
- Never use **TABS**
- The rule is to use 2 spaces per **indent**
- but **why** ??

CODING STYLE

- Use **SPACES** to **indent** your code
- Never use **TABS**
- The rule is to use 2 spaces per **indent**
- but **why** ??
- Because it's the **smallest** indentation that you can **easily see** !

VARIABLES

LOCAL VARIABLES

LOCAL VARIABLES

- Start with a **lower case letter** or an **underscore _**

LOCAL VARIABLES

- Start with a **lower case letter** or an **underscore _**
- Example: *results, first_name, _ary*

LOCAL VARIABLES

- Start with a **lower case letter** or an **underscore _**
- Example: *results, first_name, _ary*
- Ruby variables **naming convention** called: **snake_case**

LOCAL VARIABLES

- Start with a **lower case letter** or an **underscore _**
- Example: *results*, *first_name*, *_ary*
- Ruby variables **naming convention** called: **snake_case**
- Multiple words should be separated by an underscore

LOCAL VARIABLES

- Start with a **lower case letter** or an **underscore _**
- Example: *results*, *first_name*, *_ary*
- Ruby variables **naming convention** called: **snake_case**
- Multiple words should be separated by an underscore
- **Never** use a **capital letter** to start a local variable name !

INSTANCE VARIABLES

INSTANCE VARIABLES

- Used to store information per **object instance**

INSTANCE VARIABLES

- Used to store information per **object instance**
- Must start with an @ sign

INSTANCE VARIABLES

- Used to store information per **object instance**
- Must start with an @ sign
- Example: *@first_name*, *@results*

INSTANCE VARIABLES

- Used to store information per **object instance**
- Must start with an @ sign
- Example: *@first_name*, *@results*
- Can start with an uppercase letter: *@First_name* ...

INSTANCE VARIABLES

- Used to store information per **object instance**
- Must start with an **@** sign
- Example: *@first_name*, *@results*
- Can start with an uppercase letter: *@First_name* ...
- ... But you should **avoid** it

CLASS VARIABLES

CLASS VARIABLES

- Used to store information per **class hierarchy**

CLASS VARIABLES

- Used to store information per **class hierarchy**
- Must start with a double @@ sign

CLASS VARIABLES

- Used to store information per **class hierarchy**
- Must start with a double @@ sign
- Example: @@counter, @@total_results

CLASS VARIABLES

- Used to store information per **class hierarchy**
- Must start with a double @@ sign
- Example: @@counter, @@total_results
- Can start with an uppercase letter: @@Count ...

CLASS VARIABLES

- Used to store information per **class hierarchy**
- Must start with a double @@ sign
- Example: @@counter, @@total_results
- Can start with an uppercase letter: @@Count ...
- ... But, here also, you should **avoid** it

GLOBAL VARIABLES

GLOBAL VARIABLES

- Easily identifiable by their leading **\$** sign

GLOBAL VARIABLES

- Easily identifiable by their leading **\$** sign
- Example: *\$LOAD_PATH*, *\$:*, *\$/*

GLOBAL VARIABLES

- Easily identifiable by their leading **\$** sign
- Example: *\$LOAD_PATH*, *\$:*, *\$/*
- **But avoid creating new global variables !!**

GLOBAL VARIABLES

- Easily identifiable by their leading **\$** sign
- Example: *\$LOAD_PATH*, *\$:*, *\$/*
- **But avoid creating new global variables !!**
- It's (very) **BAD practice**

REMEMBER THIS

REMEMBER THIS

- Ruby variables type is determined by the **first character(s)** of their name

REMEMBER THIS

- Ruby variables type is determined by the **first character(s)** of their name

\$GLOBAL_VARIABLE	\$LOAD_PATH, \$bad_practice
@@class_variable	@@total_count, @@TOTAL_COUNT
@instance_variable	@results, @first_name
CONSTANT	VERSION, PI
local_variable	x, _my_string

METHODS

METHOD DEFINITION

METHOD DEFINITION

- Follow the **same naming rules** as **local variables**

METHOD DEFINITION

- Follow the **same naming rules** as **local variables**
- Start with keyword *def* followed by **lower case letter** or an **underscore** _

METHOD DEFINITION

- Follow the **same naming rules** as **local variables**
- Start with keyword *def* followed by **lower case letter** or an **underscore** _
- End by keyword *end*

METHOD DEFINITION

- Follow the **same naming rules** as **local variables**
- Start with keyword *def* followed by **lower case letter** or an **underscore** _
- End by keyword *end*
- Method **name** can end with **?, ! or =**

METHOD DEFINITION

- Follow the **same naming rules** as **local variables**
- Start with keyword *def* followed by **lower case letter** or an **underscore** _
- End by keyword *end*
- Method **name** can end with **?, ! or =**
- Example: *create, _get_user, is_active?, replace!, result=(value)*

NAMING CONVENTION

NAMING CONVENTION

- Methods that returns *true* or *false* should end by *?*

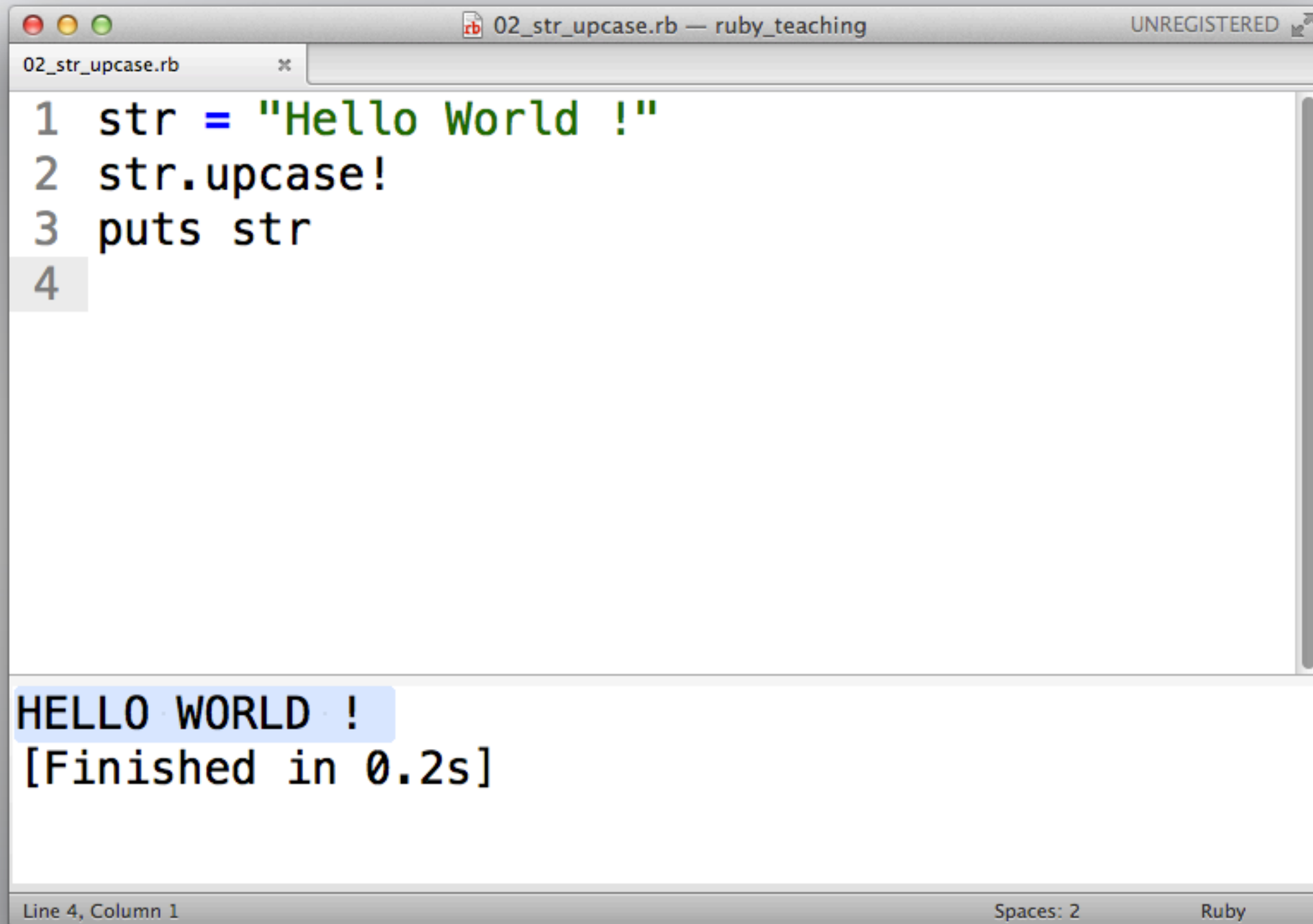
NAMING CONVENTION

- Methods that returns *true* or *false* should end by *?*
- Methods that *modify* object *in place* should end by *!*

NAMING CONVENTION

- Methods that returns *true* or *false* should end by *?*
- Methods that *modify* object *in place* should end by *!*

```
str = "Hello World !"  
str.upcase!  
puts str
```



```
02_str_upcase.rb
1 str = "Hello World !"
2 str.upcase!
3 puts str
4
```

HELLO WORLD !
[Finished in 0.2s]

Line 4, Column 1 Spaces: 2 Ruby

METHODS EXAMPLE

METHODS EXAMPLE

```
def say(message)
  puts "Ruby says: #{message}"
end
```

METHODS EXAMPLE

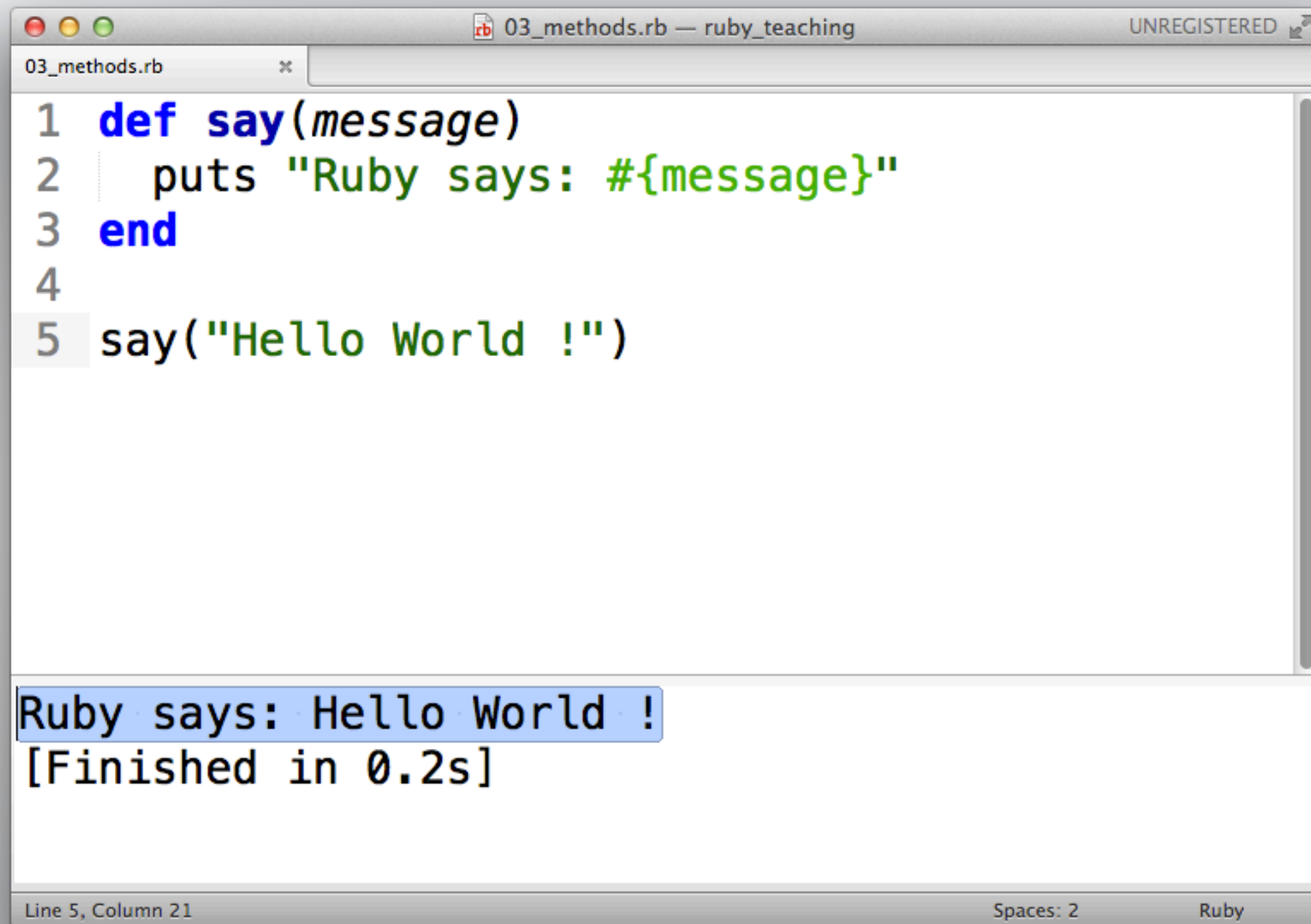
```
def say(message)
  puts "Ruby says: #{message}"
end
```

```
def exclude?(str, value)
  !str.include?(value)
end
```

METHODS EXAMPLE

```
def say(message)
  puts "Ruby says: #{message}"
end
```

```
def exclude?(str, value) # no explicit return needed, last evaluated
  !str.include?(value)   expression is returned by default.
end
```

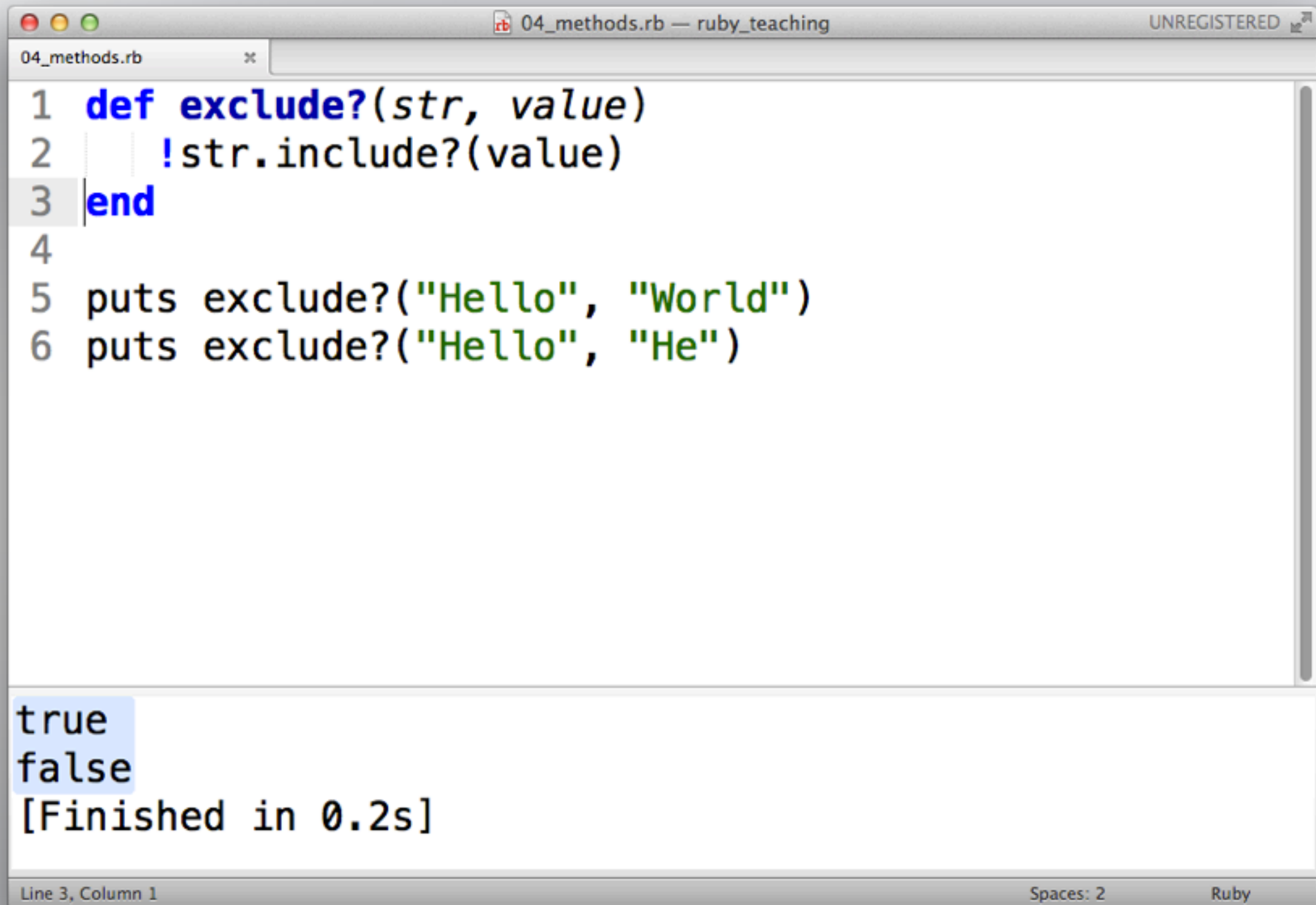



The image shows a screenshot of a Ruby IDE window titled "03_methods.rb — ruby_teaching". The window has a tab labeled "03_methods.rb" and a status bar at the bottom indicating "Line 5, Column 21", "Spaces: 2", and "Ruby". The main editor area contains the following Ruby code:

```
1 def say(message)
2   puts "Ruby says: #{message}"
3 end
4
5 say("Hello World !")
```

Below the code editor, the output of the script is displayed in a separate pane:

```
Ruby says: Hello World !
[Finished in 0.2s]
```



```
1 def exclude?(str, value)
2   !str.include?(value)
3 end
4
5 puts exclude?("Hello", "World")
6 puts exclude?("Hello", "He")
```

true
false
[Finished in 0.2s]

Line 3, Column 1 Spaces: 2 Ruby

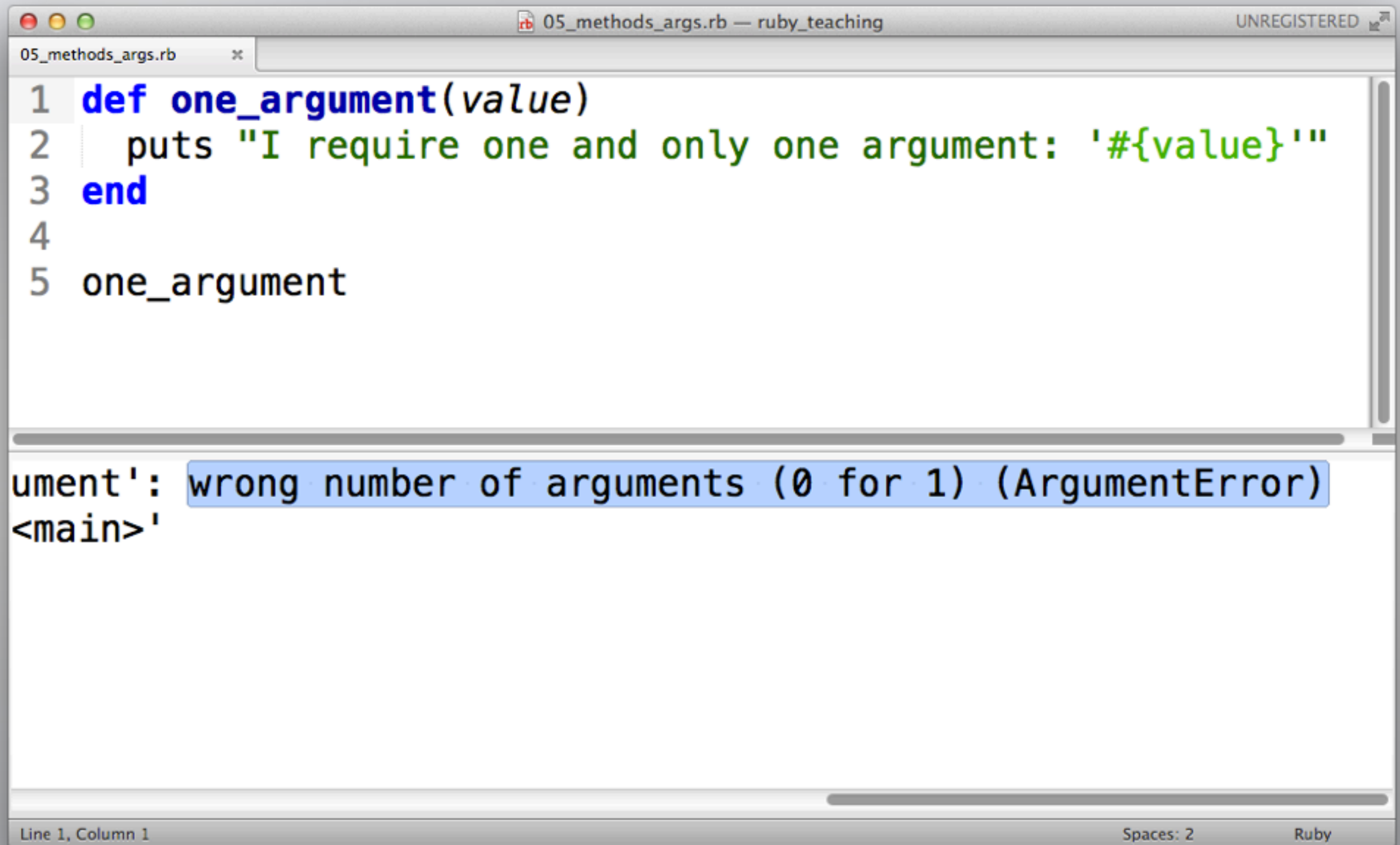
METHOD ARGUMENTS #1

- You have to supply the **correct number** of **arguments**

METHOD ARGUMENTS #1

- You have to supply the **correct number** of **arguments**

```
def one_argument(value)  
  puts "I require one and only one argument: '#{value}'"  
end
```



The screenshot shows a Ruby IDE window titled "05_methods_args.rb — ruby_teaching" with a status bar indicating "UNREGISTERED". The editor contains the following Ruby code:

```
1 def one_argument(value)
2   puts "I require one and only one argument: '#{value}'"
3 end
4
5 one_argument
```

Below the editor, the console displays an error message:

```
ArgumentError: wrong number of arguments (0 for 1) (ArgumentError)
<main>
```

The status bar at the bottom shows "Line 1, Column 1", "Spaces: 2", and "Ruby".

06_methods_args.rb

```
1 def one_argument(value)
2   puts "I require one and only one argument: '#{value}'"
3 end
4
5 one_argument(1,2,3)
```

```
ArgumentError: wrong number of arguments (3 for 1) (ArgumentError)
main>
```

Line 5, Column 20

Spaces: 2

Ruby

07_methods_args.rb

```
1 def one_argument(value)
2   puts "I require one and only one argument: '#{value}'"
3 end
4
5 one_argument("Hello World !")
```

```
I require one and only one argument: 'Hello World !'
[Finished in 0.2s]
```

Line 5, Column 28

Spaces: 2

Ruby

MULTIPLE VALUES

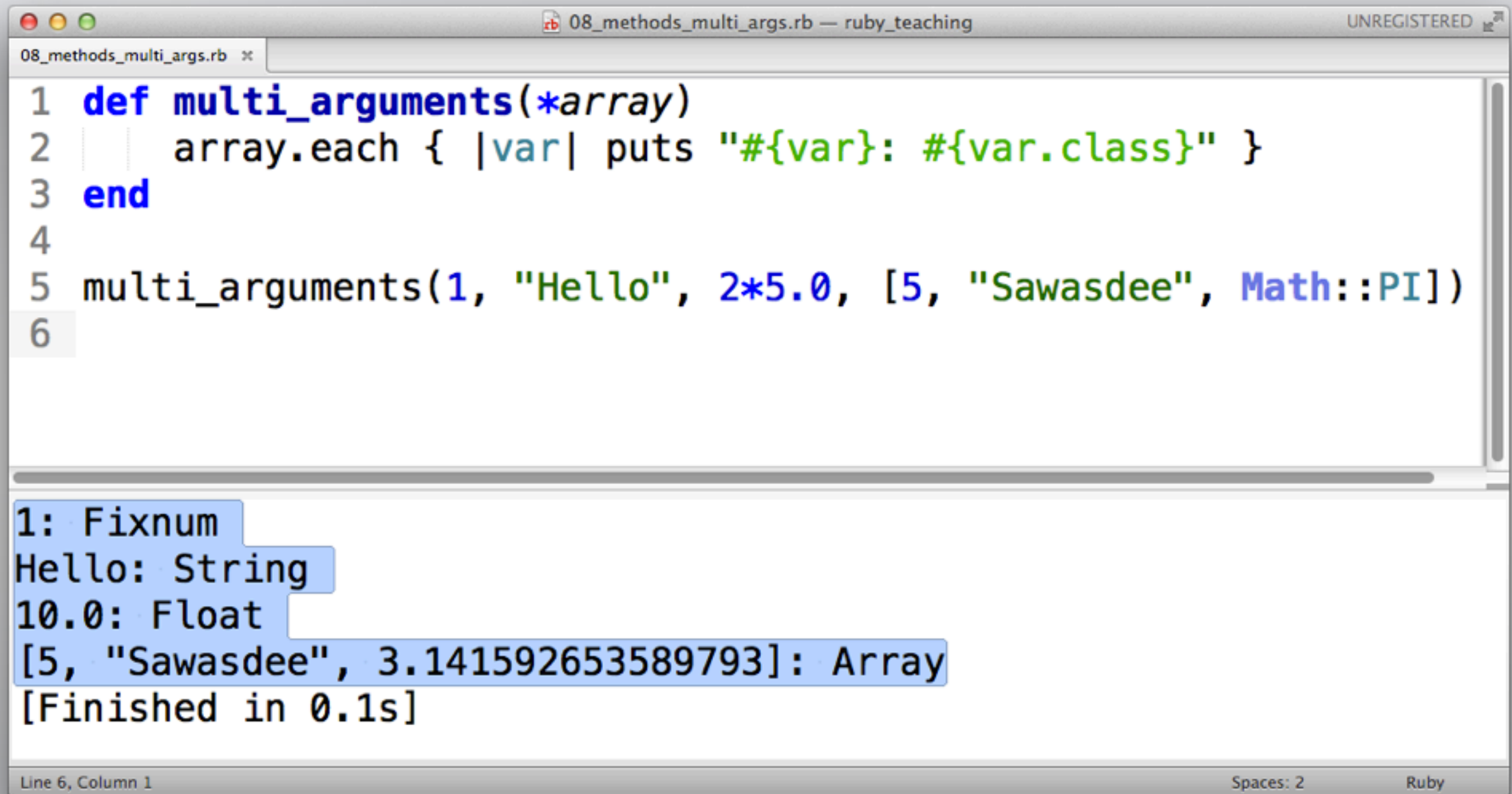
MULTIPLE VALUES

- You can define **methods** that allows **any number** of **arguments**

MULTIPLE VALUES

- You can define **methods** that allows **any number** of **arguments**

```
def multi_arguments(*array)
  array.each { |var| puts "#{var}: #{var.class}" }
end
```



The screenshot shows a Ruby IDE window titled "08_methods_multi_args.rb — ruby_teaching" with a tab for "08_methods_multi_args.rb". The code defines a method `multi_arguments` that takes a variable number of arguments (`*array`) and prints each argument along with its class. The method is then called with arguments: `1`, `"Hello"`, `2*5.0`, and an array `[5, "Sawasdee", Math::PI]`. The output shows the class of each argument: `1: Fixnum`, `Hello: String`, `10.0: Float`, and `[5, "Sawasdee", 3.141592653589793]: Array`. The execution finished in 0.1s. The status bar at the bottom indicates "Line 6, Column 1", "Spaces: 2", and "Ruby".

```
1 def multi_arguments(*array)
2   array.each { |var| puts "#{var}: #{var.class}" }
3 end
4
5 multi_arguments(1, "Hello", 2*5.0, [5, "Sawasdee", Math::PI])
6
```

```
1: Fixnum
Hello: String
10.0: Float
[5, "Sawasdee", 3.141592653589793]: Array
[Finished in 0.1s]
```

Line 6, Column 1 Spaces: 2 Ruby

DEFAULT VALUES

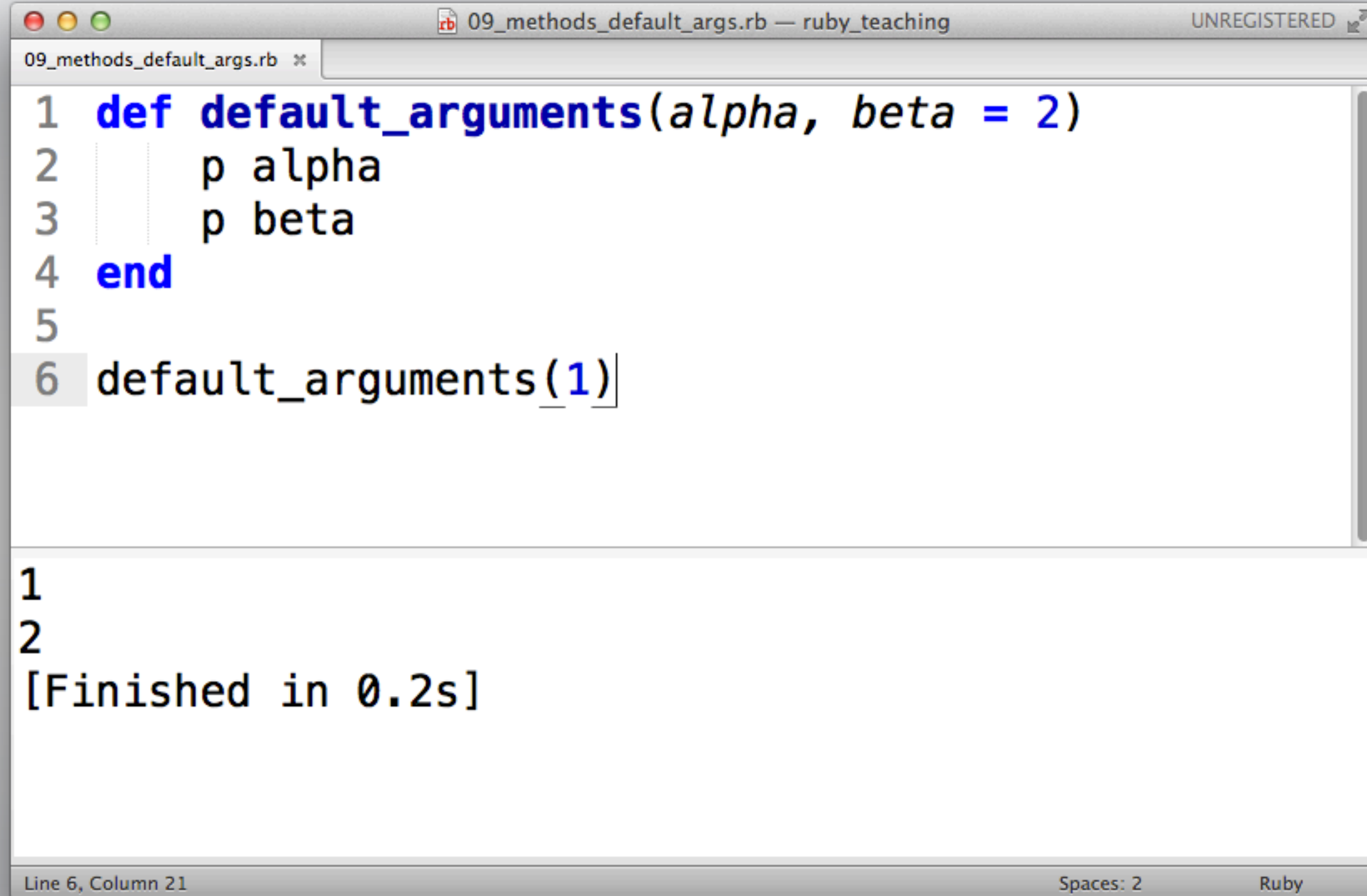
DEFAULT VALUES

- You can define **methods** with **default value** for **arguments**

DEFAULT VALUES

- You can define **methods** with **default value** for **arguments**

```
def default_arguments(alpha, beta = 2)
  p alpha
  p beta
end
```



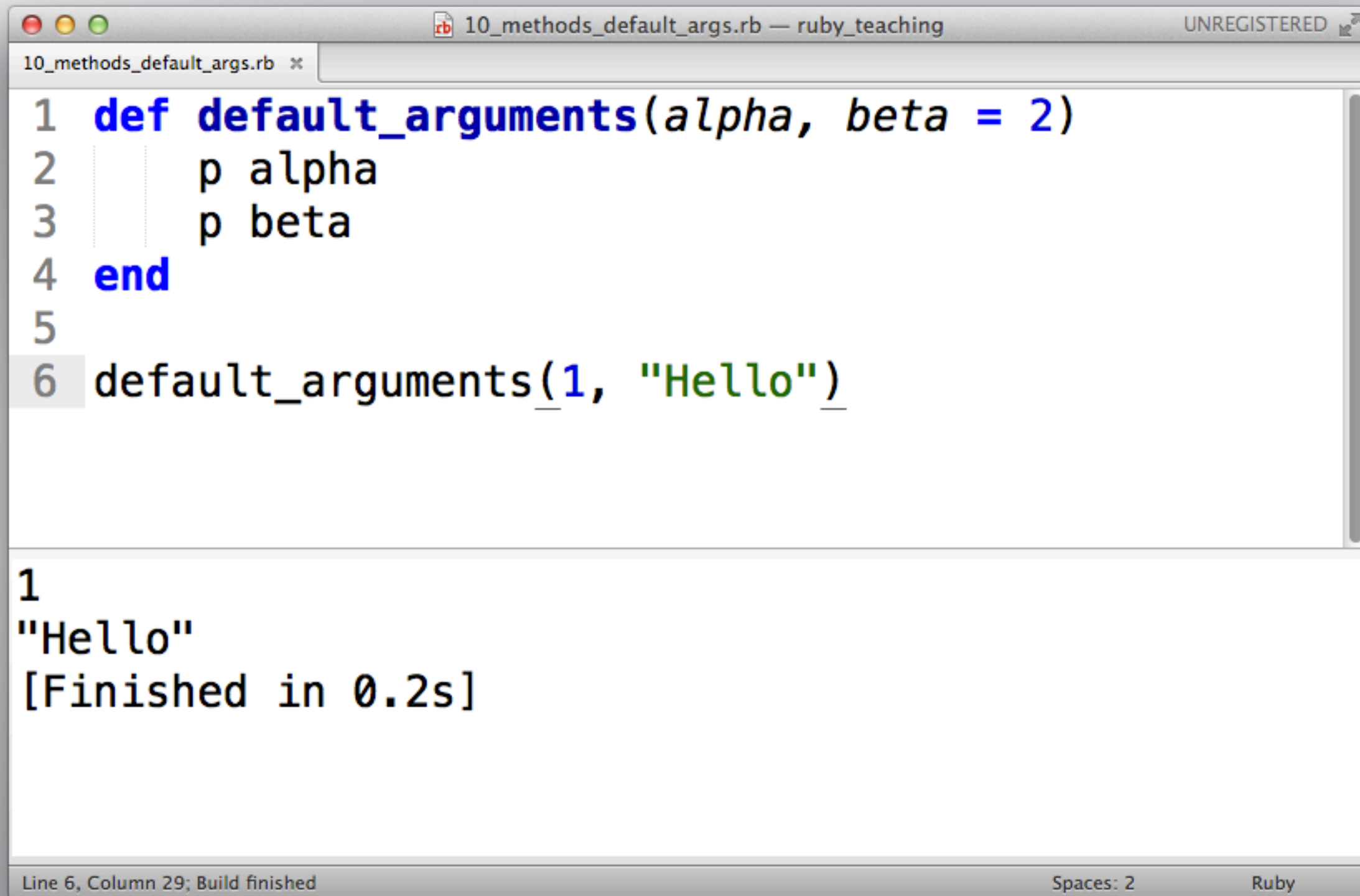
The screenshot shows a Ruby IDE window titled "09_methods_default_args.rb — ruby_teaching" with a status bar indicating "UNREGISTERED". The editor contains the following Ruby code:

```
1 def default_arguments(alpha, beta = 2)
2   |   p alpha
3   |   p beta
4 end
5
6 default_arguments(1)
```

The output pane shows the execution results:

```
1
2
[Finished in 0.2s]
```

The status bar at the bottom indicates "Line 6, Column 21", "Spaces: 2", and "Ruby".



The image shows a screenshot of a Ruby IDE window titled "10_methods_default_args.rb — ruby_teaching". The window contains a Ruby script defining a method `default_arguments` with two parameters, `alpha` and `beta`, where `beta` has a default value of 2. The method prints the values of `alpha` and `beta`. Below the script, the output of the method call `default_arguments(1, "Hello")` is shown, displaying the values "1" and "Hello", followed by a completion message "[Finished in 0.2s]". The status bar at the bottom indicates "Line 6, Column 29; Build finished", "Spaces: 2", and the language is "Ruby".

```
1 def default_arguments(alpha, beta = 2)
2   |   p alpha
3   |   p beta
4 end
5
6 default_arguments(1, "Hello")
```

```
1
"Hello"
[Finished in 0.2s]
```

Line 6, Column 29; Build finished Spaces: 2 Ruby

REMEMBER THIS

REMEMBER THIS

- Method definition **start** by the **keyword** *def*

REMEMBER THIS

- Method definition **start** by the **keyword** *def*
- Method definition **end** by the **keyword** *end*

REMEMBER THIS

- Method definition **start** by the **keyword** *def*
- Method definition **end** by the **keyword** *end*
- Method **name** start by **lower case letter** or an **underscore** _

REMEMBER THIS

- Method definition **start** by the **keyword** *def*
- Method definition **end** by the **keyword** *end*
- Method **name** start by **lower case letter** or an **underscore** _
- Method **name** can end with **?, ! or =**

CLASSES

CLASS DEFINITION

CLASS DEFINITION

- A **class** is a construct used to **create instances** of itself...

CLASS DEFINITION

- A **class** is a construct used to **create instances** of itself...
- ...referred to as *class instances*, *instance objects* or simply **objects**

CLASS DEFINITION

- A **class** is a construct used to **create instances** of itself...
- ... referred to as *class instances*, *instance objects* or simply **objects**
- Each **object** can have his **own state** and **behavior**

CLASS DEFINITION

- A **class** is a construct used to **create instances** of itself...
- ... referred to as *class instances*, *instance objects* or simply **objects**
- Each **object** can have his **own state** and **behavior**
- **Instance variables** enable an object to **maintain** his **state**

CLASS DEFINITION

- A **class** is a construct used to **create instances** of itself...
- ... referred to as *class instances*, *instance objects* or simply **objects**
- Each **object** can have his **own state** and **behavior**
- **Instance variables** enable an object to **maintain** his **state**
- **Methods** enable the **behavior** of **objects**

RUBY CLASS

RUBY CLASS

- Start with keyword *class* followed by an **uppercase letter**

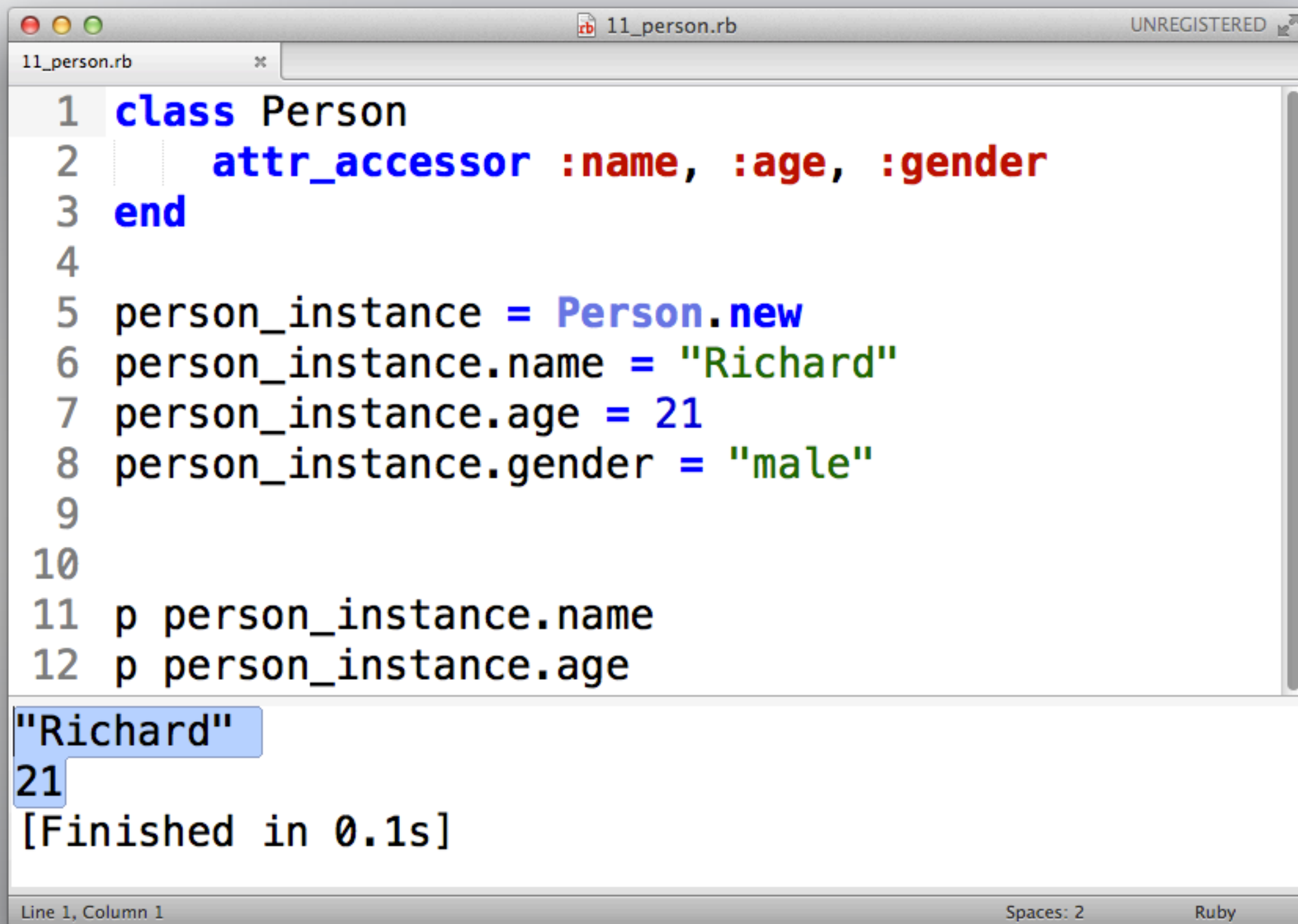
RUBY CLASS

- Start with keyword *class* followed by an **uppercase letter**
- End by keyword *end*

CLASS EXAMPLE

```
class Person
  attr_accessor :name, :age, :gender
end
```

```
person_instance = Person.new
person_instance.name = "Richard"
person_instance.age = 21
person_instance.gender = "male"
```

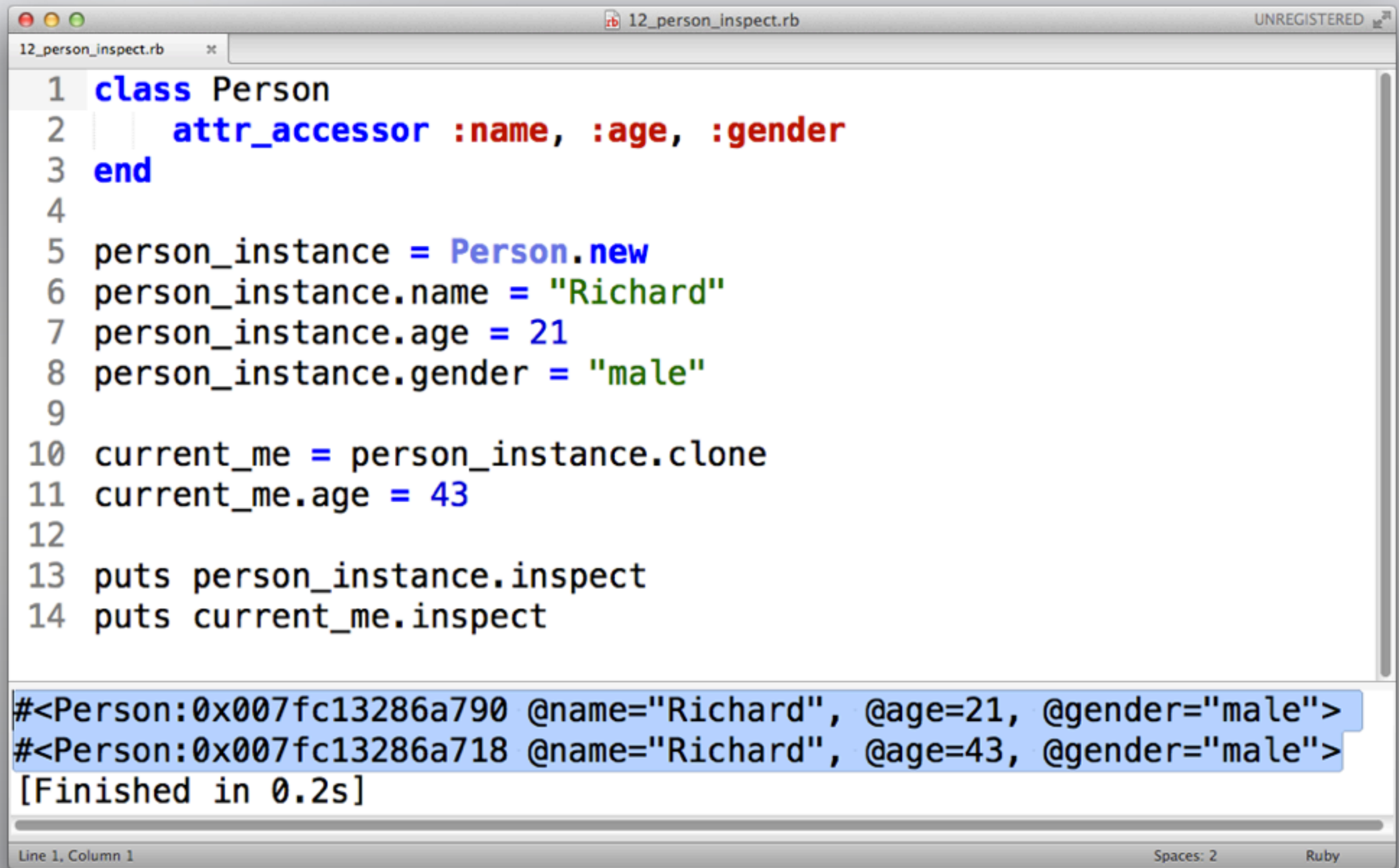



The image shows a screenshot of a Ruby IDE window titled "11_person.rb". The window has a tab labeled "11_person.rb" and a status bar at the bottom indicating "Line 1, Column 1", "Spaces: 2", and "Ruby". The code in the editor is as follows:

```
1 class Person
2   attr_accessor :name, :age, :gender
3 end
4
5 person_instance = Person.new
6 person_instance.name = "Richard"
7 person_instance.age = 21
8 person_instance.gender = "male"
9
10
11 p person_instance.name
12 p person_instance.age
```

Below the code editor, the output of the program is displayed:

```
"Richard"
21
[Finished in 0.1s]
```

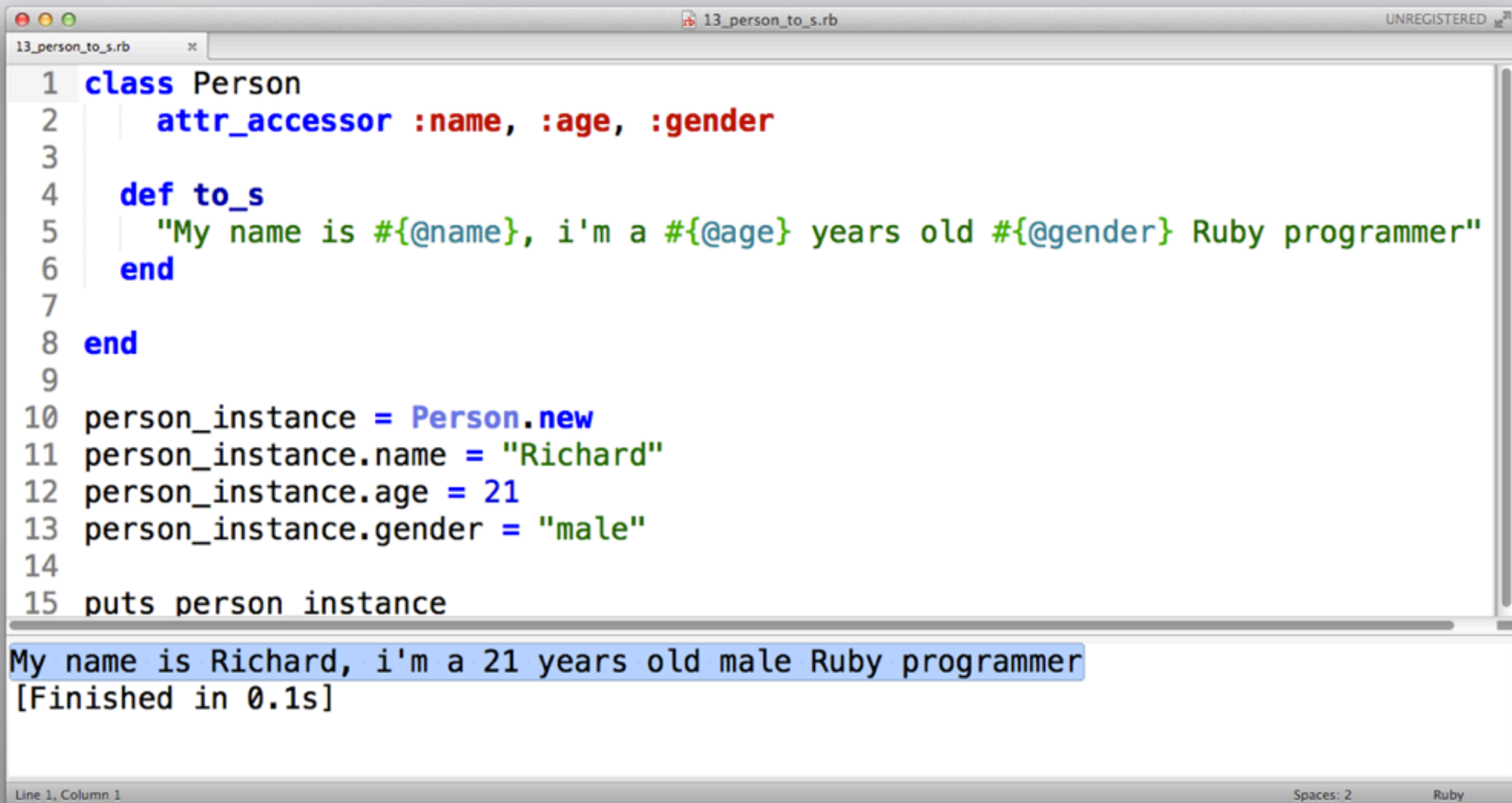


The screenshot shows a Ruby IDE window titled "12_person_inspect.rb" with a tab labeled "12_person_inspect.rb". The code defines a `Person` class with `attr_accessor` for `:name`, `:age`, and `:gender`. It then creates a `person_instance` with name "Richard", age 21, and gender "male". A clone of this instance, `current_me`, is created and its age is changed to 43. Finally, `puts` statements are used to inspect both objects. The output at the bottom shows the inspect results for both objects, highlighting that they are distinct instances with the same name and gender but different ages. The execution time is noted as 0.2s. The status bar at the bottom indicates "Line 1, Column 1", "Spaces: 2", and "Ruby".

```
1 class Person
2   attr_accessor :name, :age, :gender
3 end
4
5 person_instance = Person.new
6 person_instance.name = "Richard"
7 person_instance.age = 21
8 person_instance.gender = "male"
9
10 current_me = person_instance.clone
11 current_me.age = 43
12
13 puts person_instance.inspect
14 puts current_me.inspect
```

```
#<Person:0x007fc13286a790 @name="Richard", @age=21, @gender="male">
#<Person:0x007fc13286a718 @name="Richard", @age=43, @gender="male">
[Finished in 0.2s]
```

Line 1, Column 1 Spaces: 2 Ruby



The image shows a screenshot of a Ruby IDE window titled "13_person_to_s.rb". The window contains a Ruby class definition for "Person" with attributes :name, :age, and :gender. A method "to_s" is defined to return a string describing the person. Below the class definition, an instance of "Person" is created and its attributes are set. The output of the program is displayed at the bottom of the window.

```
1 class Person
2   attr_accessor :name, :age, :gender
3
4   def to_s
5     "My name is #{@name}, i'm a #{@age} years old #{@gender} Ruby programmer"
6   end
7
8 end
9
10 person_instance = Person.new
11 person_instance.name = "Richard"
12 person_instance.age = 21
13 person_instance.gender = "male"
14
15 puts person_instance
```

My name is Richard, i'm a 21 years old male Ruby programmer
[Finished in 0.1s]

Line 1, Column 1 Spaces: 2 Ruby

MORE ABOUT CLASSES

CLASSES IN DETAIL

CLASSES IN DETAIL

- Ruby classes are **executable code**

CLASSES IN DETAIL

- Ruby classes are **executable code**
- The *new* method is a **constructor**

CLASSES IN DETAIL

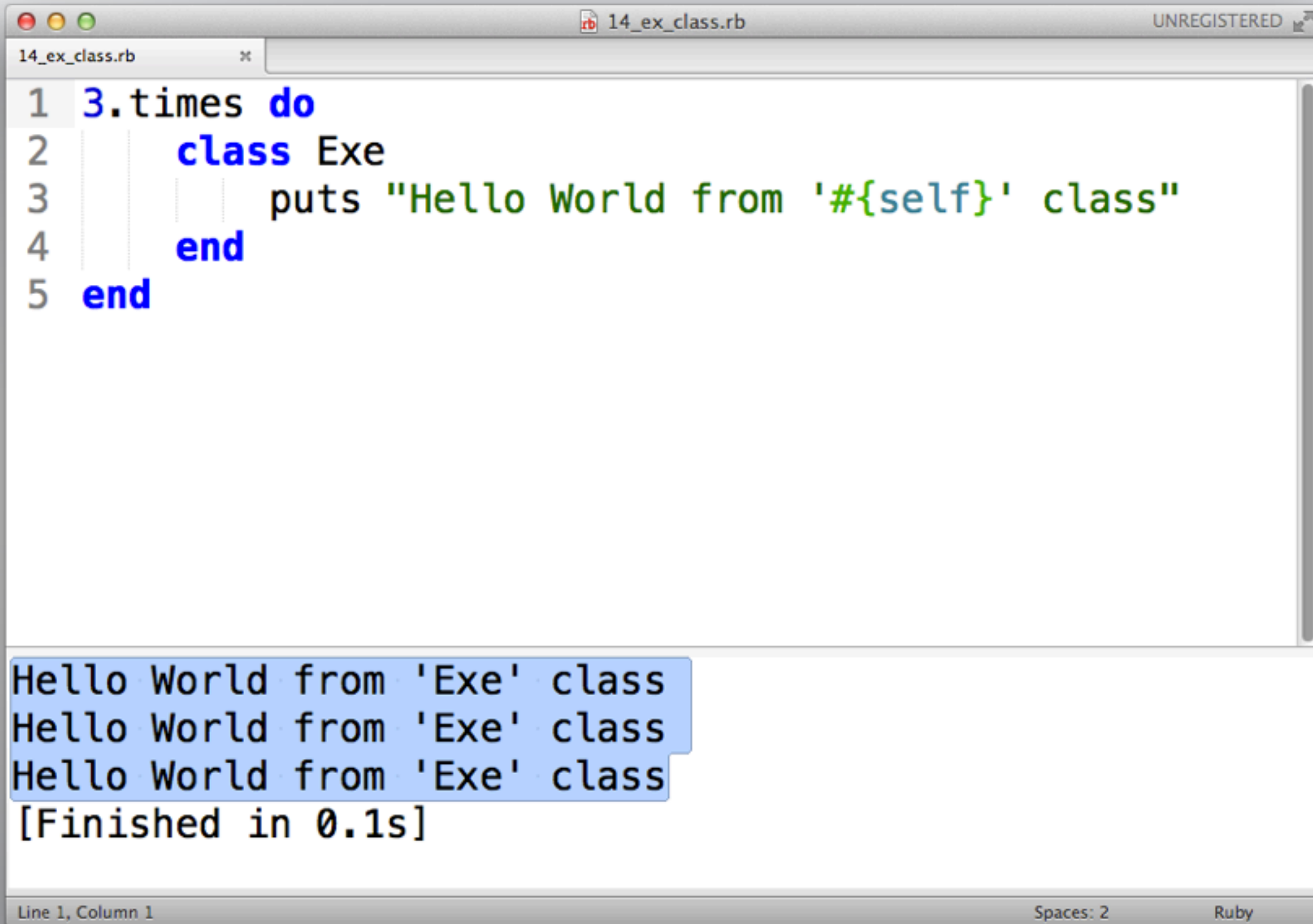
- Ruby classes are **executable code**
- The *new* method is a **constructor**
- **Classes** are **named** with **constants**

CLASSES IN DETAIL

- Ruby classes are **executable code**
- The *new* method is a **constructor**
- **Classes** are **named** with **constants**
- In Ruby **classes** are **objects** (ouch !)

CLASS IS EXECUTABLE CODE

```
3.times do
  class Exe
    puts "Hello World from '#{self}' class"
  end
end
```



```
1 3.times do
2    class Exe
3      puts "Hello World from '#{self}' class"
4    end
5 end
```

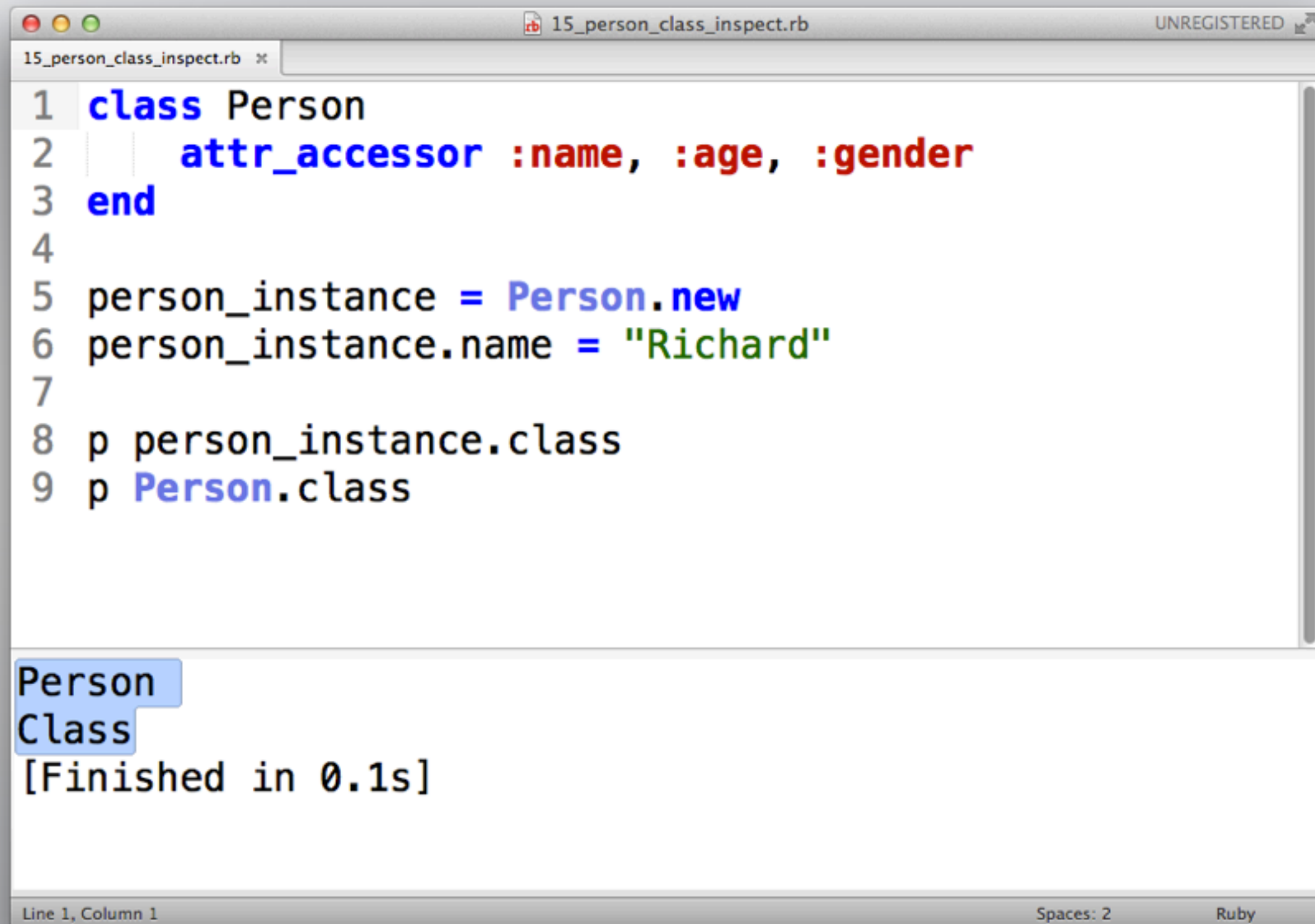
```
Hello World from 'Exe' class
Hello World from 'Exe' class
Hello World from 'Exe' class
[Finished in 0.1s]
```

Line 1, Column 1 Spaces: 2 Ruby

A CLASS IS AN OBJECT

```
class Person  
  attr_accessor :name, :age, :gender  
end
```

```
person_instance = Person.new  
person_instance.name = "Richard"
```



The image shows a screenshot of a Ruby IDE window. The window title is "15_person_class_inspect.rb" and it is marked as "UNREGISTERED". The code editor contains the following Ruby code:

```
1 class Person
2   attr_accessor :name, :age, :gender
3 end
4
5 person_instance = Person.new
6 person_instance.name = "Richard"
7
8 p person_instance.class
9 p Person.class
```

Below the code editor, the output of the program is displayed:

```
Person
Class
[Finished in 0.1s]
```

The status bar at the bottom of the window shows "Line 1, Column 1", "Spaces: 2", and "Ruby".

REOPENING CLASSES

REOPENING CLASSES

- Ruby **classes** implementation is **not closed**

REOPENING CLASSES

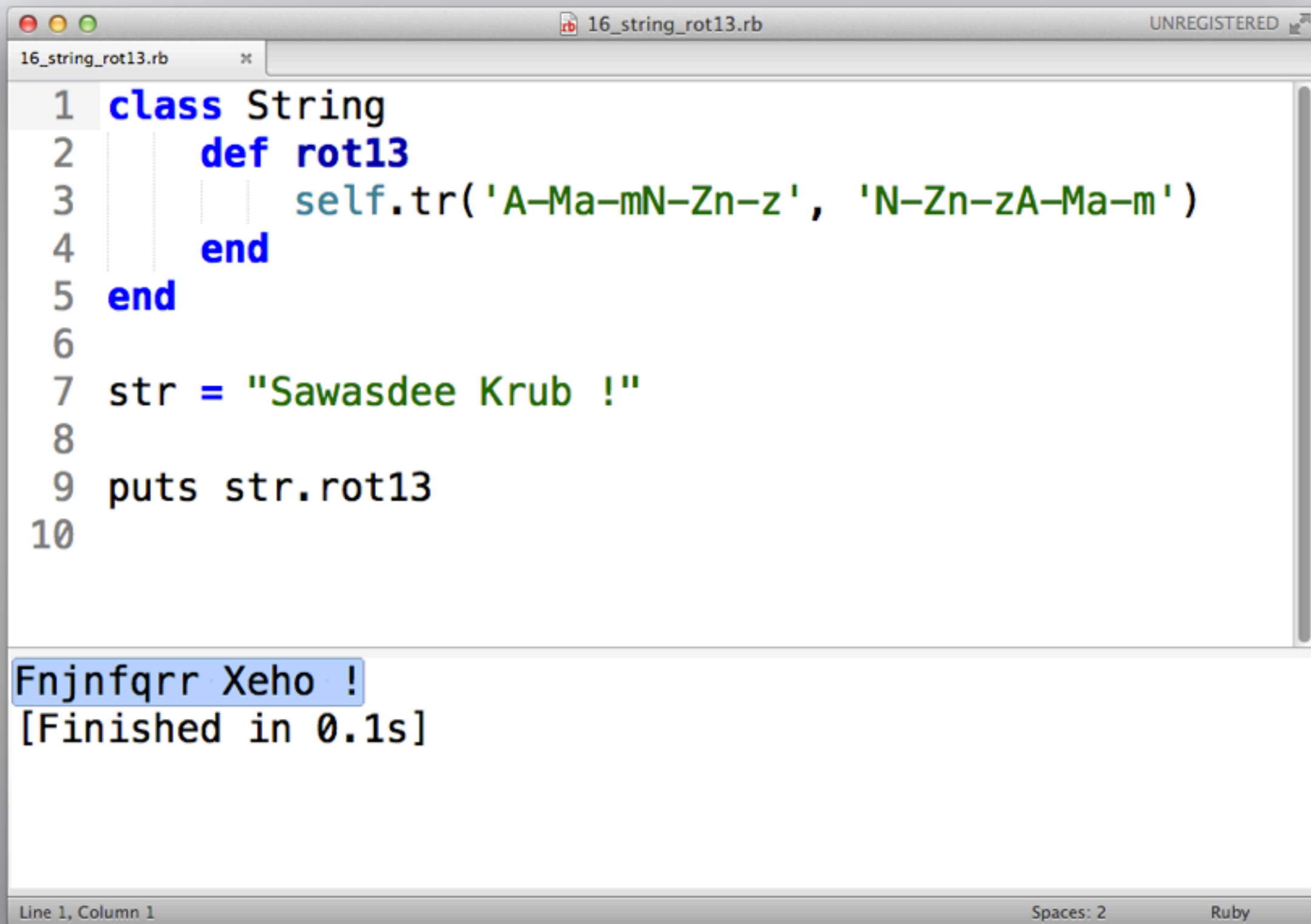
- Ruby **classes** implementation is **not closed**
- Ruby **classes** can be **reopened** and **modified**

REOPENING CLASSES

- Ruby **classes** implementation is **not closed**
- Ruby **classes** can be **reopened** and **modified**
- **Any class** can be reopened !

REOPENING CLASSES

- Ruby **classes** implementation is **not closed**
- Ruby **classes** can be **reopened** and **modified**
- **Any class** can be reopened !
- This **feature** make Ruby **incredibly flexible**



The image shows a screenshot of a Ruby IDE window titled "16_string_rot13.rb" with a status bar indicating "UNREGISTERED". The code defines a `String` class with a `rot13` method that uses `tr` to rotate characters. It then creates a string "Sawasdee Krub !" and prints its `rot13` result. The output "Fnjnfqrr Xeho !" is displayed in the console, highlighted with a blue selection box. The console also shows "[Finished in 0.1s]". The status bar at the bottom indicates "Line 1, Column 1", "Spaces: 2", and "Ruby".

```
1 class String
2   def rot13
3     self.tr('A-Ma-mN-Zn-z', 'N-Zn-zA-Ma-m')
4   end
5 end
6
7 str = "Sawasdee Krub !"
8
9 puts str.rot13
10
```

Fnjnfqrr Xeho !
[Finished in 0.1s]

Line 1, Column 1 Spaces: 2 Ruby

EXERCISE

EXERCISE

- Add the **exclude?** method to all **String** objects

EXERCISE

- Add the **exclude?** method to all **String objects**
- Original implementation below

EXERCISE

- Add the **exclude?** method to all **String objects**
- Original implementation below

```
def exclude?(str, value)  
  !str.include?(value)  
end
```

COMMAND LINE SWITCHES

Switch	Description	Example of usage
-c	Check the syntax of a program file without executing the program	<code>ruby -c c2f.rb</code>
-w	Give warning messages during program execution	<code>ruby -w c2f.rb</code>
-e	Execute the code provided in quotation marks on the command line	<code>ruby -e 'puts "Code demo!"'</code>
-v	Show Ruby version information, and execute the program in verbose mode	<code>ruby -v</code>
-l	Line mode: print a newline after every line of output	<code>ruby -le 'print "+ newline!"'</code>
-rname	Require the named feature	<code>ruby -rprofile</code>
--version	Show Ruby version information	<code>ruby --version</code>



Creative Intelligence for Business



www.inotechne.com

Copyright ©2012 Inotechne. All Rights Reserved.