HTML5 and CSS3 for Mobile Applications

Prof. Paul Krause, University of Surrey

Introducing Ruby and Rails

Objectives for today

- Explore some core Ruby syntax
  - You should take time to experiment a little with what you learn
- Introduce Classes in Ruby

Source for today’s material

- The “Pickaxe book”:
  - Programming Ruby 1.9 - The Pragmatic Programmer’s Guide
  - Dave Thomas
  - Pragmatic Bookshelf

Opinionated Software?

- Convention over configuration
- Elegance is not optional
- Do Not Repeat Yourself
- Developer motivation and productivity are primary factors in project success

Let’s take a look at Ruby

Key Features of Ruby

- Easy syntax
- Fully OO
- Highly dynamic
- Strong Web frameworks
    - Ruby on Rails

Let’s Try it!

- Open an Interactive Ruby shell

    > irb

    > quit *or* exit *will end the session*

“Hello World” in Java

```
class HelloWorld {

        public static void main(String[] args) {

            System.out.println(“Hello World!”)

        }

    }
```

and in Ruby:

```
puts “Hello World!”
```

Even Better

Enter: 3.times {puts “Hello World!”}

Ruby’s Objects

- Enter: 4 (return)

- Enter: 4.567.round

- Find out the class of something you have entered...

Ruby is Dynamically Typed

- Enter: n=1
- Enter: n.class
- Enter: n=”fishpaste”
- Enter: n.class
- How about: n+4; n.size; n.methods

Regular Expressions

- Regular expressions are written between slashes
- Enter: regex = /better/
- Find out what class it is.
- Enter: “Mine is bigger” =~ regex
- Enter: “Mine is better” =~ regex

Containers

- Enter: stack = [1, 2, 3]
- Enter: stack.push “cat”
- Enter: stack.pop
- Enter: stack

Arrays and Hashes

```
a = [1, ‘cat’, 3.14]    # array with 3 elements

puts “The first element is #{a[0]}”

# set the third element

a[2] = nil

puts “The array is now #{a.inspect}”
```

Note that *nil* is an object – it just represents nothing

Short cut to arrays with words

```
a = [ ‘ant’, ‘bee’, ‘cat’, ‘dog’, ‘fox’]
```

Try a[0], a[1] in the irb

Alternative is

```
a = %w{ ant bee cat dog fox }
```

Try the above again.

Hashes

- Basically a list of key, value pairs separated by “=>”
- Each Key in a particular Hash must be unique

```
inst_section = {
  ‘cello’ => ‘string’,
  ‘clarinet’ => ‘woodwind’,
  ‘drum’ => ‘percussion’,
  ‘oboe’ => ‘woodwind’,
  ‘trumpet’ => ‘brass’,
  ‘violin’ => ‘string’
}
```

- Try accessing with `p inst_section[‘KEY’]`
  - (what happens if you use a key that is not yet defined?)

Control Structures

```
if count > 10
   puts “Try Again”
elseif tries == 3
   puts “You loose”
else
   puts “Enter a number”
end
```

```
while weight < 100 and num_pallets <= 30
   pallet = next_pallet()
   weight += pallet.weight
   num_pallets += 1
end
```

Statements as conditions

`gets` returns `nil` when the end of file is reached, and

`nil` is treated as “false” in conditions, so

```
while line = gets
  puts line.downcase
end
```

will terminate cleanly when the end of file is reached.

Statement modifiers

- Useful if the body of an if or while statement is just a single expression

```
if radiation > 1000
   puts “I suggest you leave now!”
end
```

- Can be rewritten as

```
puts “I suggest you leave now!” if radiation > 1000
```

- Also

```
square = 2
square = square*square while square < 1000
```

Regular Expressions

- To match a string containing either Perl or Python use:

```
/Perl|Python/ or
/P(erl|ython)/
```

- Repetition – one a, followed by one or more b’s and finish with one c:

```
/ab+c/
```

- For zero or more b’s use “*”:

```
/ab*c/
```

- Character classes

`\s` – matches any white space character
\w – matches characters that may appear in words [A-Z,a-z,0-9]
\d – matches any digit
. – matches (almost) any character

Using Regular Expressions

```
if line =~ /Perl|Python/
  puts “Scripting language mentioned: #{line}”
end
```

- Changing history:

```
line.sub(/Perl/, ‘Ruby’)     # Replace first ‘Perl’ with ‘Ruby’
line.gsub(/Python/, ‘Ruby’) # Replace every ‘Python’ with
                             # ‘Ruby’

line.gsub(/Perl|Python/, ‘Ruby’) # Total dominance
```

Blocks and iterators

- Two kinds of delimiter for code blocks

```
{ puts “Hello” }
```

- Or

```
do
   club.enroll(person)
   person.socialize
 end
```

Yield

- What can you do with a block?

- You can associate it with a call to a method

```
greet { puts “Hi” }
```

  - The method (‘greet’ in the above case) can then invoke the block using the Ruby `yield` statement

  - Try it out...

Blocks and yield

- Inter this into a Ruby file:

```
def call_block
  puts "Start of Block"
  yield
  yield
  puts "End of method"
end

call_block { puts "In the block" }
```

Passing arguments into a block

```
def who_says_what
  yield("Dave", "hello")
  yield("Simon", "goodbye")
end

who_says_what {|person, phrase| puts "#{person} says
                                               #{phrase}"}
```

Using blocks to implement iterators

- You will see this used widely in Ruby and in Rails
- Iterators return successive elements from some kind of collection. E.g.:

```
animals = %w( ant bee cat dog fox )
animals.each {|animal| puts animal}
```

- You might remember this example from the last lecture:

```
3.times {puts “Hello World!”}
```

Writing

- Ruby supports formatted writing in much the same way as C, Java and PERL
- Use `printf` as illustrated below:

```
printf(“Number: %5.2f, \nString: %s\n”, 1.23, “hello”)
```

Classes, Objects and Variables

- We will use a simple example to base this discussion around
  - Following the “Pickaxe Book”
- We want to monitor stock in a bookshop:
  - Scan books to record: Date; ISBN No.; Price
  - Enter each record into a file
  - Analyse the data to find out how many copies of each book we have, and what is the total value of the stock

Class BookInStock

- Open a Ruby Project
- Call it BookShop, or something similar
- Once the project has been created, right-click Source Files and create a new Ruby Class. Call it BookInStock.
- A file will be generated with the following skeleton:

```
class BookInStock
  def initialize

  end
end
```

Adding State

- We need to add in instance variables so that objects of class BookInStock actually contain the information we need:

```
class BookInStock
    def initialize(isbn, price)
         @isbn=isbn
         @price=Float(price)
    end
end
```

Adding State

- We need to add in instance vari of class BookInStock actually contain the information

local variables

```
class BookInStock
    def initialize(isbn, price)
        @isbn=isbn
        @price=Float(price)
    end
end
```

Adding State

- We need to add in instance vari of class BookInStock actually contain the information

local variables

```
class BookInStock
    def initialize(isbn, price)
        @isbn=isbn
        @price=Float(price)
    end
end
```

instance variables

Print out some objects

```
class BookInStock
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end
end
b1 = BookInStock.new("isbn1", 3)
p b1
b2 = BookInStock.new("isbn2", 3.14)
p b2
b1 = BookInStock.new("isbn3", “5.67”)
p b3
```

Creating a string representation

```ruby
class BookInStock
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end
  def to_s
    "ISBN: #{@isbn}, price: #{@price}"
  end
end

b1 = BookInStock.new("isbn1", 3)
puts b1
b2 = BookInStock.new("isbn2", 3.14)
puts b2
b3 = BookInStock.new("isbn1", "5.67")
puts b3
```

Accessing instance variables

```
class BookInStock
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end
  def isbn
    @isbn
  end

  # ...
end
```

attr_reader

```
class BookInStock
  attr_reader :isbn, :price
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end

  # ...
end
```

attr_reader

```
class BookInStock
  attr_reader :isbn, :price
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end

  # ...
end
```

These symbols represent names for the accessor methods and their corresponding instance variables

Writable attributes

```
class BookInStock
  attr_reader :isbn
  attr_accessor :price
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end

  # ...
end
```

Access Control

- Public methods
  - can be called by anyone (default)
- Protected methods
  - keep it in the family - only accessible by objects of the defining class and its subclasses
- Private methods
  - the receiver is *always* the current object

Specifying Access Control

```
class AccessClass
  def method1   # default is public
    # ...
  end









end
```

Specifying Access Control

```
class AccessClass
  def method1   # default is public
    # ...
  end
  protected
  def method2   # subsequent methods will be ‘protected’
    # ...
  end




end
```

Specifying Access Control

```
class AccessClass
  def method1   # default is public
     # ...
  end
  protected
  def method2   # subsequent methods will be ‘protected’
    # ...
  end
  private
  def method3   # subsequent methods will be ‘private’
    # ...
  end
end
```

Variables

- Remember, a Variable is a *reference* to an object

```
person1 = "Tim"
person2 = person1

person1[0] = 'J'

puts "person1 is #{person1}"
puts "person2 is #{person2}"
```

Contrast with

- Duplicating an object

```
person1 = "Tim"
person2 = person1.dup

person1[0] = 'J'

puts "person1 is #{person1}"
puts "person2 is #{person2}"
```

Inheritance

“the child is father of the man”

The child class inherits its parent’s methods

```
class Parent
  def sayHello
    puts "Hello from #{self}"
  end
end

class Child < Parent
end
```

Finding parents

```
class Person
  def initialize(name)
    @name = name
  end
end

puts "The superclass of Person is #{Person.superclass}"
```

Object

- Object is an ancestor of every Ruby class
- the method `to_s` is defined in Object
- Hence, every Ruby object has access to a `to_s` method
- But you will normally need to override it

Writing objects to strings

```
class Person
  def initialize(name)
    @name = name
  end
end

p = Person.new(“Brian”)
puts p
```

produces:

```
#<Person:0x18b1bc>
```

Overriding `to_s`

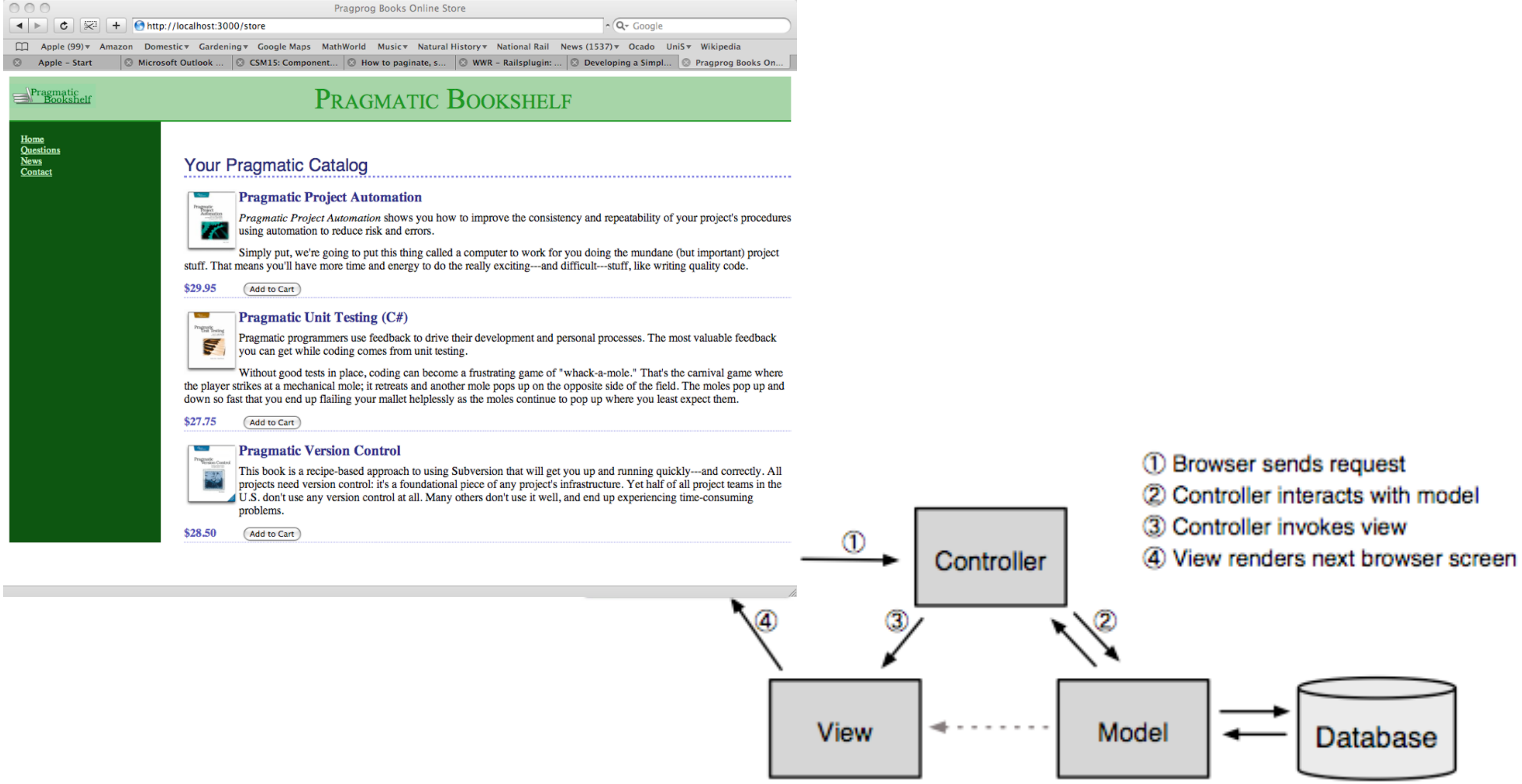
```
class Person
  def initialize(name)
    @name = name
  end
  def to_s
   "Person named #{@name}"
  end
end

p = Person.new(“Brian”)
puts p
produces:
Person named Brian
```

That’s very nice, but

- Ruby was released by its designer *Yukihiro Matsumoto* in 1995
- Why is it only relatively recently, that it has been generating so much interest?
- Let’s explore Rails a little bit ...

The Rails Framework


Pragprog Books Online Store
http://localhost:3000/store
Google
Apple (99) Amazon Domestic Gardening Google Maps MathWorld Music Natural History National Rail News (1537) Ocado UniS Wikipedia
Apple – Start
Microsoft Outlook ...
CSM15: Component...
How to paginate, s...
WWR – Railsplugin: ...
Developing a Simpl...
Pragprog Books On...
Pragmatic Bookshelf

Pragmatic Bookshelf

Home
Questions
News
Contact

Your Pragmatic Catalog

Pragmatic Project Automation

Pragmatic Project Automation shows you how to improve the consistency and repeatability of your project's procedures using automation to reduce risk and errors.

Simply put, we're going to put this thing called a computer to work for you doing the mundane (but important) project stuff. That means you'll have more time and energy to do the really exciting---and difficult---stuff, like writing quality code.

$29.95 Add to Cart

Pragmatic Unit Testing (C#)

Pragmatic programmers use feedback to drive their development and personal processes. The most valuable feedback you can get while coding comes from unit testing.

Without good tests in place, coding can become a frustrating game of "whack-a-mole." That's the carnival game where the player strikes at a mechanical mole; it retreats and another mole pops up on the opposite side of the field. The moles pop up and down so fast that you end up flailing your mallet helplessly as the moles continue to pop up where you least expect them.

$27.75 Add to Cart

Pragmatic Version Control

This book is a recipe-based approach to using Subversion that will get you up and running quickly---and correctly. All projects need version control: it's a foundational piece of any project's infrastructure. Yet half of all project teams in the U.S. don't use any version control at all. Many others don't use it well, and end up experiencing time-consuming problems.

$28.50 Add to Cart

① Browser sends request
② Controller interacts with model
③ Controller invokes view
④ View renders next browser screen

Recap and Next

- We have delved a little deeper into Ruby basics
- Next time we will
    - explore classes in a little more detail
    - start to explore Rails