Blocks, Procs, Lambdas & Enumerable

Oh My

http://tinyurl.com/jaw6-blocks

What are we talking about?

```
# open_camp / app / views / notes / index.html.erb
<h1>Listing notes</h1>

<table>
  <tr>
    <th>Title</th>
    <th>Body</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @notes.each do |note| %>
  <tr>
    <td><%= note.title %></td>
    <td><%= note.body %></td>
    <td><%= link_to 'Show', note_path(note) %></td>
    <td><%= link_to 'Edit', edit_note_path(note) %></td>
    <td><%= link_to 'Destroy', note, method: :delete %></td>
  </tr>
<% end %>
</table>

<%= link_to 'New Note', new_note_path %>
```

What are we talking about?

```erb
# open_camp / app / views / notes / index.html.erb
<h1>Listing notes</h1>

<table>
  <tr>
    <th>Title</th>
    <th>Body</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @notes.each do |note| %>
  <tr>
    <td><%= note.title %></td>
    <td><%= note.body %></td>
    <td><%= link_to 'Show', note_path(note) %></td>
    <td><%= link_to 'Edit', edit_note_path(note) %></td>
    <td><%= link_to 'Destroy', note, method: :delete %></td>
  </tr>
<% end %>
</table>

<%= link_to 'New Note', new_note_path %>
```

What are we talking about?

```
# open_camp / app / views / notes / _form.html.erb
<%= form_for(@note) do |f| %>
  <% if @note.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@note.errors.count, "error") %>
          prohibited this task from being saved:</h2>
      <ul>
      <% @note.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :body %><br />
    <%= f.text_field :body %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

What are we talking about?

```
# open_camp / app / views / notes / _form.html.erb
<%= form_for(@note) do |f| %>
  <% if @note.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(@note.errors.count, "error") %>
          prohibited this task from being saved:</h2>
      <ul>
      <% @note.errors.full_messages.each do |msg| %>
        <li><%= msg %></li>
      <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </div>
  <div class="field">
    <%= f.label :body %><br />
    <%= f.text_field :body %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

What are we talking about?

```
# open_camp / app / controllers / tasks_controller.rb
class TasksController < ApplicationController
  before_filter :authenticate_user!
  # GET /tasks
  # GET /tasks.json
  def index
    @tasks = current_user.tasks

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @tasks }
    end
  end
  [...]
end
```

What are we talking about?

```
# open_camp / app / controllers / tasks_controller.rb
class TasksController < ApplicationController
  before_filter :authenticate_user!
  # GET /tasks
  # GET /tasks.json
  def index
    @tasks = current_user.tasks

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @tasks }
    end
  end
  [...]
end
```

What are we talking about?

```
# open_camp / app / controllers / tasks_controller.rb
class TasksController < ApplicationController
  before_filter :authenticate_user!
  # GET /tasks
  # GET /tasks.json
  def index
    @tasks = current_user.tasks

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @tasks }
    end
  end
  [...]
end
```

do this stuff

```
do_stuff do |some_variable|
  now_do_stuff_with(some_variable)
end

do_stuff do |user, data, admin|
  now_do_stuff_with(user, data) if admin
end

do_stuff { |variable| do_stuff(variable) }
```

do this stuff

```
some_method_name { |variable| do_stuff(variable) }
```

```
def some_method_name
  yield some_variable
end

def some_method_name(&block)
  block.call(some_variable)
end
```

What’s it for?

- allow the other method to control order: insert other operations before or after “this stuff”
- store operations for later
- separates concerns: who is responsible for which parts?
- conditional code: do this, but only sometimes
- loops with **each**

File.open

```ruby
# open a file
log = File.open("mylogfile.log")

# open a second file
archive = File.open("archive.log", "a")

results = log.readlines

# Close the log file
log.close

results.each_line do |line|
  # Copy to second file
  archive.puts("#{line}")
end

# Close the archive file
archive.close
```

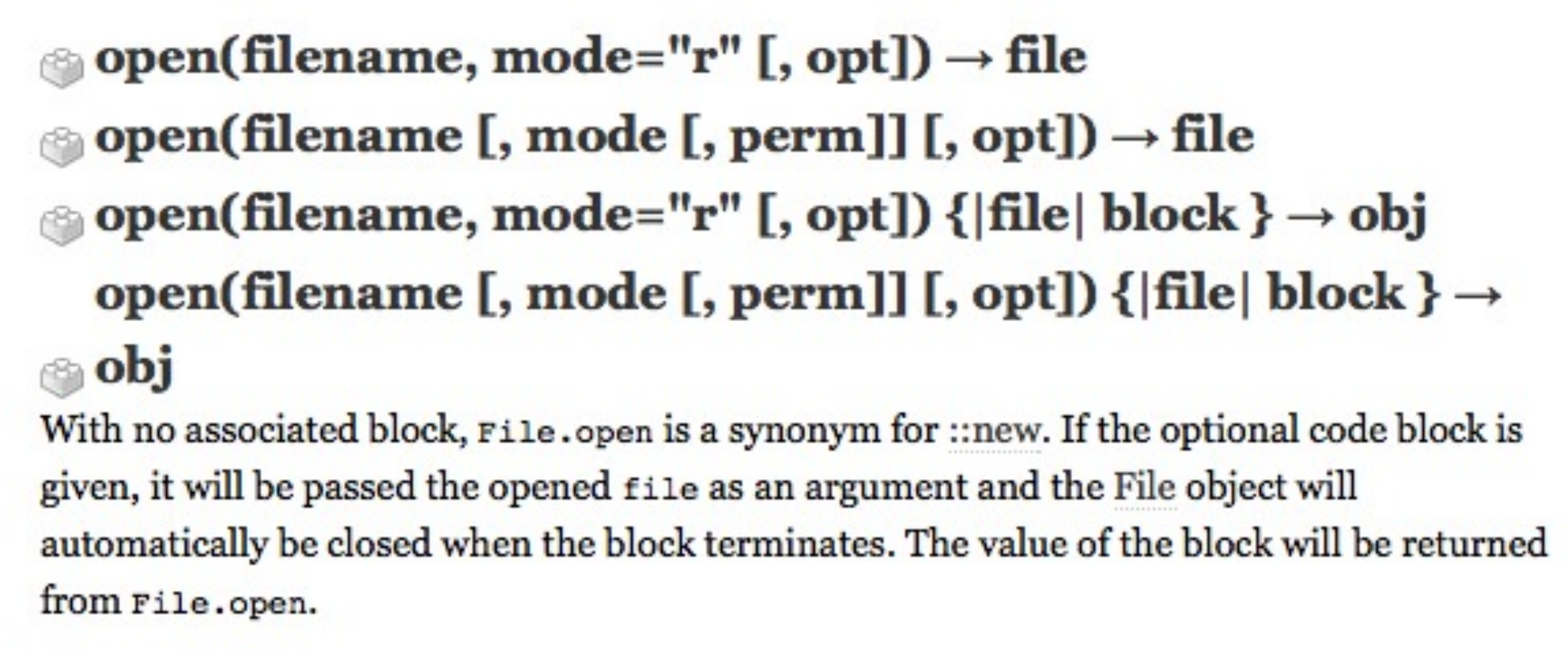
File.open

```
# open a file
log = File.open("mylogfile.log")

# open a second file
archive = File.open("archive.log", "a")

results = log.readlines

# Close the log file
log.close

results.each_line do |line|
  # Copy to second file
  archive.puts("#{line}")
end

# Close the archive file
archive.close
```

File.open

```
# open a file
log = File.open("mylogfile.log")

# open a second file
archive = File.open("archive.log", "a")

results = log.readlines

# Close the log file
log.close

results.each_line do |line|
  # Copy to second file
  archive.puts("#{line}")
end

# Close the archive file
archive.close
```

File.open



open(filename, mode="r" [, opt]) → file

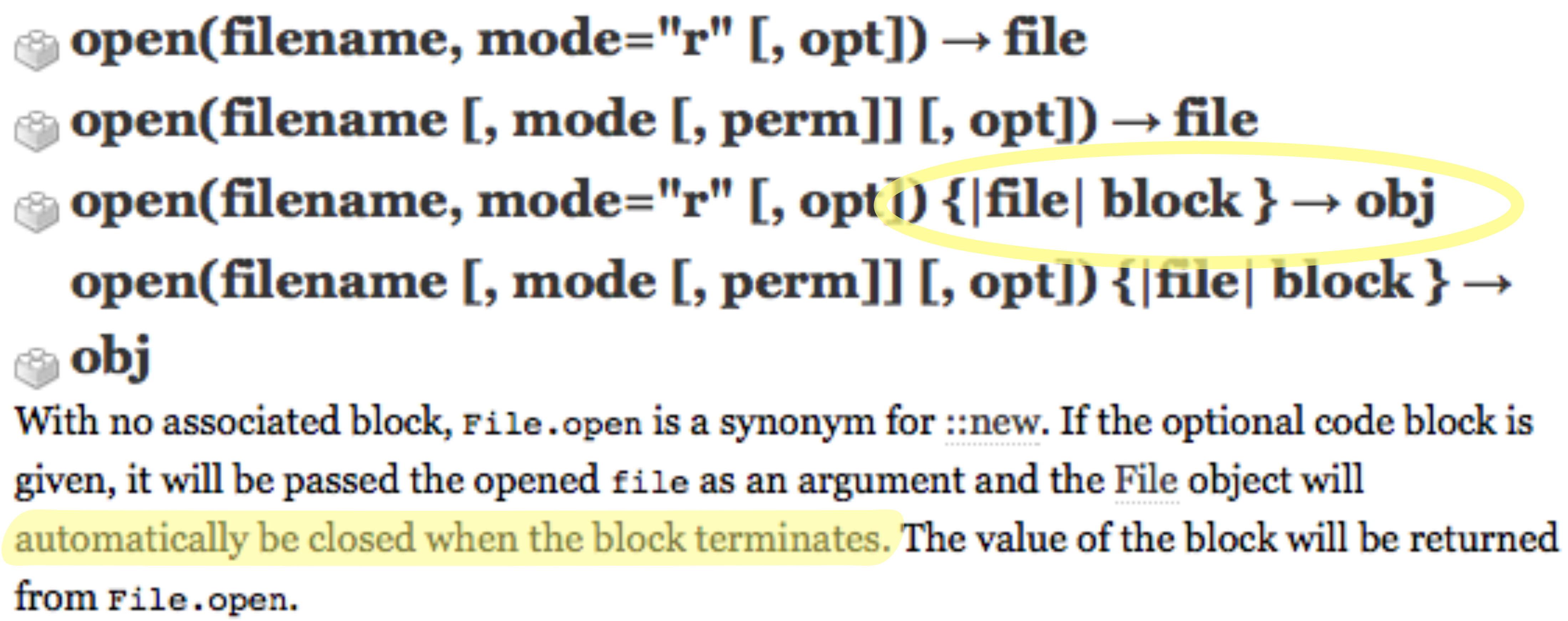
open(filename [, mode [, perm]] [, opt]) → file

open(filename, mode="r" [, opt]) {|file| block } → obj

open(filename [, mode [, perm]] [, opt]) {|file| block } → obj

With no associated block, `File.open` is a synonym for ::new. If the optional code block is given, it will be passed the opened `file` as an argument and the File object will automatically be closed when the block terminates. The value of the block will be returned from `File.open`.

File.open



open(filename, mode="r" [, opt]) → file
open(filename [, mode [, perm]] [, opt]) → file
open(filename, mode="r" [, opt]) {|file| block } → obj
open(filename [, mode [, perm]] [, opt]) {|file| block } → obj

With no associated block, `File.open` is a synonym for ::new. If the optional code block is given, it will be passed the opened `file` as an argument and the File object will automatically be closed when the block terminates. The value of the block will be returned from `File.open`.

File.open

```
# Open archive
File.open("archive.log", "a") do |archive|

  # foreach feeds one line to the block
  File.foreach("mylogfile.log") do |line|

    # Copy this line to archive
    archive.puts(line)

  end # log is automatically closed

end # archive is automatically closed
```

File.open

- allow the other method to control order: insert other operations before or after “this stuff”
- store operations for later
- separates concerns: who is responsible for which parts?
- conditional code: do this, but only sometimes
- loops with **each**

respond_to do

```
# open_camp / app / controllers / tasks_controller.rb
class TasksController < ApplicationController
  before_filter :authenticate_user!
  # GET /tasks
  # GET /tasks.json
  def index
    @tasks = current_user.tasks

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @tasks }
    end
  end
  [...]
end
```

respond_to do

```
format.json { render json: @tasks }
```

- allow the other method to control order: insert other operations before or after "this stuff"
- store operations for later
- separates concerns: who is responsible for which parts?
- conditional code: do this, but only sometimes
- loops with **each**

Blocks for Better Views

Scenario: Admin users can “masquerade” as a regular user

```
# Option #1

<% if current_user %>
  <% if current_user.is_admin? %>
    <% user = if current_user.is_masquerading?
              current_user.masquerading_as_user
              else
              current_user %>
    Currently logged in as: <%= user.name %>
  <% end %>
<% end %>
```

Scenario: Admin users can “masquerade” as a regular user

```
# Option #1

<% if current_user %>
  <% if current_user.is_admin? %>
    <% user = if current_user.is_masquerading?
                current_user.masquerading_as_user
                else
                current_user %>
    Currently logged in as: <%= user.name %>
  <% end %>
<% end %>
```

Blocks for Better Views

Scenario: Admin users can "masquerade" as a regular user

```
# Option #2

<% acting_as do |user| %>
  Currently logged in as: <%= user.name %>
<% end %>

# application_helper.rb
def acting_as
  if current_user &&
     current_user.is_admin? &&
     current_user.is_masquerading?
     yield current_user.masquerading_as
  elsif current_user
    yield current_user
  end
end
```

Blocks for Better Views

Other ideas

```
# What about this?
<% with_defaults(task) do |todo| %>
  Assigned: <%= todo.assigned_to %> # “[Not assigned]”
<% end %>

# application_helper.rb
def with_defaults(task)
  filled_in = Task.new assigned_to: task.assigned_to
  filled_in.assigned_to ||= “[Not assigned]”
  if block_given?
    yield filled_in
  else
    filled_in
  end
end
```

Blocks for Better Views

```
<% acting_as do |user| %>
```

- allow the other method to control order: insert other operations before or after “this stuff”
- store operations for later
- separates concerns: who is responsible for which parts?
- conditional code: do this, but only sometimes
- loops with **each**

each

```
# open_camp / app / views / notes / index.html.erb

<table>
  <tr>
    <th>Title</th>
    <th>Body</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @notes.each do |note| %>
  <tr>
    <td><%= note.title %></td>
    <td><%= note.body %></td>
    <td><%= link_to 'Show', note_path(note) %></td>
    <td><%= link_to 'Edit', edit_note_path(note) %></td>
    <td><%= link_to 'Destroy', note, method: :delete %></td>
  </tr>
<% end %>
</table>
```

each

```
numbers = [1,2,3,4,5,6,7,8,9]
numbers.each { |x| puts x }


# Prints:
# 1
# 2
# 3
# 4
# etc.
```

each

```
numbers = [1,2,3,4,5,6,7,8,9]
numbers.each { |x| puts “I am #{x}” }
puts numbers

# Prints:
# I am 1
# I am 2
# I am 3
# I am 4
# etc.
```

map

```
numbers = [1,2,3,4,5,6,7,8,9]
numbers.map { |x| “I am #{x}” }


# crickets
```

map

```ruby
numbers = [1,2,3,4,5,6,7,8,9]
am_i = numbers.map { |x| “I am #{x}” }
puts am_i

# [
#    “I am 1”,
#    “I am 2”,
#    “I am 3”,
#     ...
# ]
```

map

```
numbers = [1,2,3,4,5,6,7,8,9]
even = numbers.map { |x| x % 2 == 0 }
puts even

# [
#   false,
#   true,
#   false,
#   true,
#    ...
# ]
```

select

```
numbers = [1,2,3,4,5,6,7,8,9]
even = numbers.select { |x| x % 2 == 0 }
puts even

# [
#   2,
#   4,
#   6,
#   8
# ]
```

detect

```
numbers = [1,2,3,4,5,6,7,8,9]
even = numbers.detect { |x| x % 2 == 0 }
puts even

# 2
```

each_with_index

```
numbers = [1,2,3,4,5,6,7,8,9]
numbers.each_with_index do |i,x|
  puts “#{i}: #{x}”
end

# Prints:
# 0: 1
# 1: 2
# 2: 3
# ...etc.
```

Demo!

any?

```
numbers = [1,2,3,4,5,6,7,8,9]
even = numbers.any? { |x| x % 2 == 0 }
puts even

# true
```

all?

```
numbers = [1,2,3,4,5,6,7,8,9]
even = numbers.all? { |x| x % 2 == 0 }
puts even

# false
```

each_slice

```ruby
numbers = [1,2,3,4,5,6,7,8,9]
numbers.each_slice(3) { |x| puts x }

# Prints:
# [1,2,3]
# [4,5,6]
# [7,8,9]
```

Bonus!

Blocks, Procs, Lambdas?

- **`do ... end`**
  this is a **block**
- **`Proc.new { |x| do_stuff }`**
  is a Proc #duh
- `lambda { |x| do_stuff }`
  this is a **lambda**
- `def do_stuff`**`(&block) ...`**
  `&block` is now a Proc

Bonus!

Blocks, Procs, Lambdas?

- A **Block** is a **language-level** feature of **Ruby** using **do ... end**, **{ ... }** and **yield**
- **Blocks** are used when creating a **Proc** or **Lambda**
- A **Proc** is a **block** that's been **turned into a variable with &block** and you can **block.call()**
- A **Lambda** is a function that you want to store in a variable and pass around

Bonus!

Blocks, Procs, Lambdas?

- **Block:**
  - **language-level**
  - only works with **yield**
  - can't use as a variable
- **Proc:**
  - can use **yield**
  - can use **proc.call(vars)**
  - can pass around as a variable
- **Lambda:**
  - **can't** use yield
  - can use **lambda.call(x)**
  - **can** pass around as a variable
- def **method(&block)**
  - **Convert** Proc or Lambda to Block
- **method(&lambda)**
  - **Convert** Block to Proc

Demo!

Bonus!

Blocks, Procs, Lambdas?

Procs don't care about arguments, but Lambdas *do*

```ruby
def demo(callable)
  one, two = 1, 2
  callable.call(one, two)
end

demo Proc.new{|a, b, c| puts "Give me a #{a} and a #{b} and a #{c.class}"}

demo lambda{|a, b, c| puts "Give me a #{a} and a #{b} and a #{c.class}"}

# => Give me a 1 and a 2 and a NilClass
# *.rb:8: ArgumentError: wrong number of arguments (2 for 3) (ArgumentError)

# Example from: http://www.robertsosinski.com/2008/12/21/understanding-ruby-blocks-procs-and-lambdas/
```

Bonus!

Blocks, Procs, Lambdas?

Procs return from the outer method, Lambdas return from themselves

```ruby
def proc_return
  Proc.new { return "Proc.new"}.call
  return "proc_return method finished"
end

def lambda_return
  lambda { return "lambda" }.call
  return "lambda_return method finished"
end

puts proc_return
puts lambda_return

# => Proc.new
# => lambda_return method finished
# Example from http://www.robertsosinski.com/2008/12/21/understanding-
ruby-blocks-procs-and-lambdas/
```

Blocks, Procs, Lambdas & Enumerable

?

I'm **@jaw6** on twitter/github

joshua.wehner@gmail.com