Code of Art 2

Beta Version

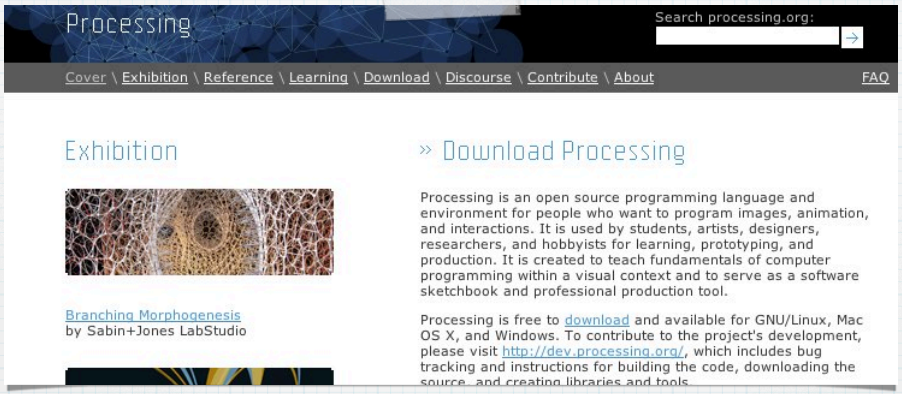
integrating ruby-processing with your Ruby and Rails applications

`github.com/jcasimir/code_of_art2`

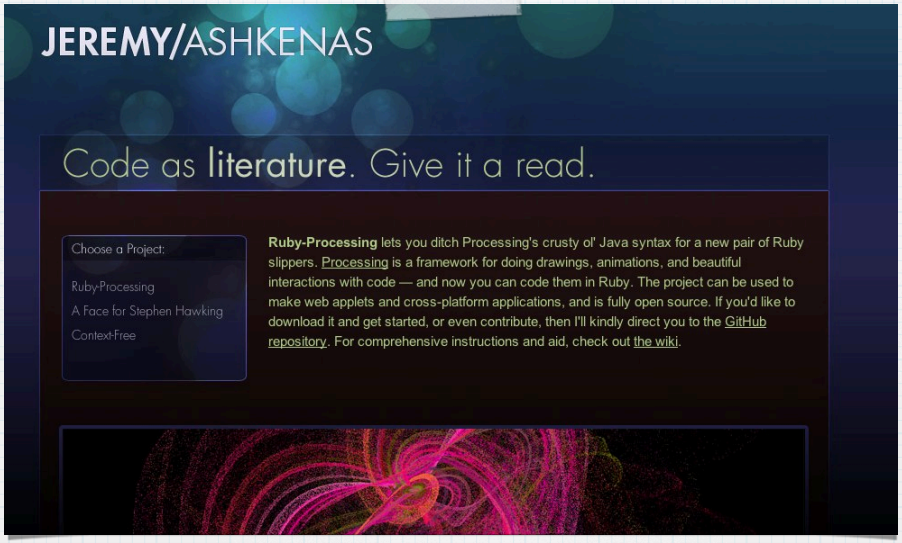
The Plan

* What is Processing?
* Creating Sample Content
* Automation & Messaging
* Next Steps

Introduction to Processing



What is **Processing**?



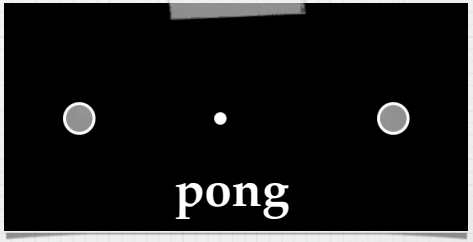
Ruby-Processing

Art in 2D

Art in 3D

Audio/Video

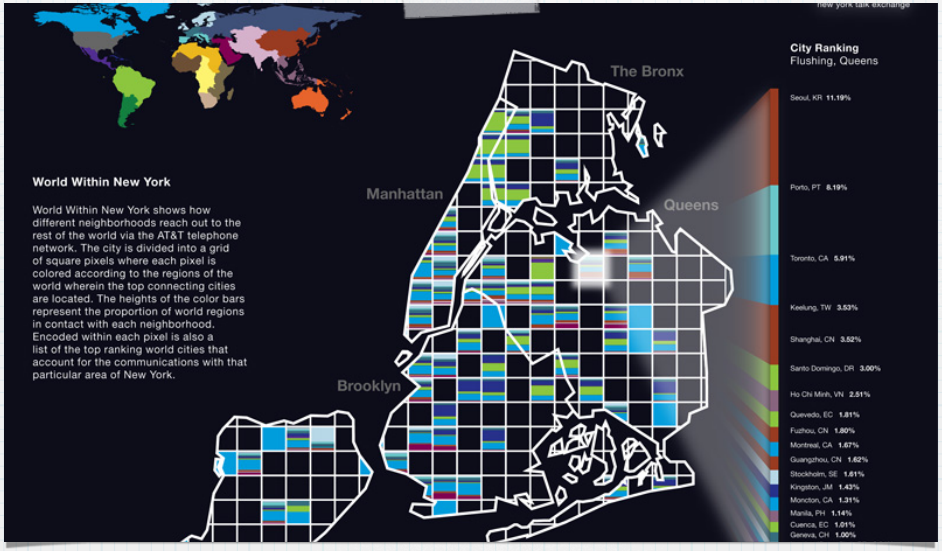
Interaction



The Good...

...and the

Not So Good


World Within New York
World Within New York shows how different neighborhoods reach out to the rest of the world via the AT&T telephone network. The city is divided into a grid of square pixels where each pixel is colored according to the regions of the world wherein the top connecting cities are located. The heights of the color bars represent the proportion of world regions in contact with each neighborhood. Encoded within each pixel is also a list of the top ranking world cities that account for the communications with that particular area of New York.
The Bronx
Manhattan
Queens
Brooklyn
City Ranking
Flushing, Queens
Seoul, KR 11.19%
Porto, PT 8.19%
Toronto, CA 5.91%
Keelung, TW 3.53%
Shanghai, CN 3.52%
Santo Domingo, DR 3.00%
Ho Chi Minh, VN 2.51%
Quevedo, EC 1.81%
Fuzhou, CN 1.80%
Montreal, CA 1.67%
Guangzhou, CN 1.62%
Stockholm, SE 1.61%
Kingston, JM 1.43%
Moncton, CA 1.31%
Manila, PH 1.14%
Cuenca, EC 1.01%
Geneva, CH 1.00%

Creativity

Ease of Development

Performance

Extremely Large Data

And where does that get you?

How I got Here

What does this icon make you *think*?

Slowdowns Ahead!

Annoying Interface

Who uses desktop apps?

Excel Feature Envy

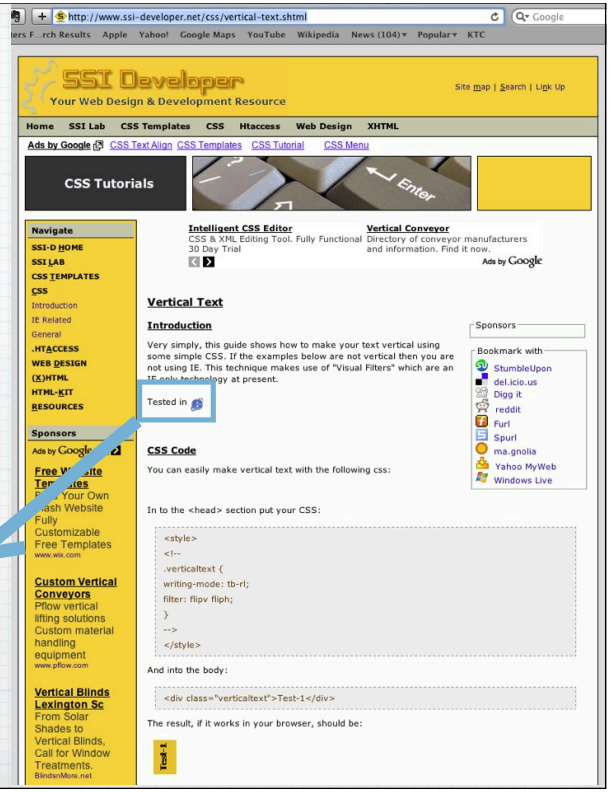
	A	B	C	D	E
1	Last Name	First Name	Week 6 Quiz	Week 7 Review Sheet	Week 8 Quiz
2	Dane	John	5	4	5
3	Goins	Sarah	4	4	5
4	Lancaster	Robert	4	2	3
5	Osprey	Karen	3	4	3
6	Samuels	Luke	2	3	4

Vertical Table Headers

Is that So Much to Ask?

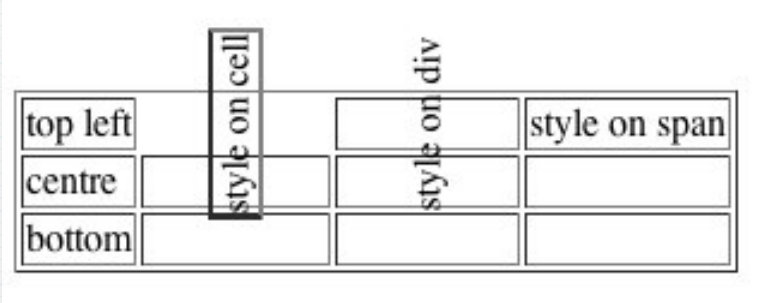
CSS?

Tested in



Is that So Much to Ask?

CSS Transforms?



Is that
So Much
to Ask?

CSS

Javascript

SVG

Canvas

Conclusion:

I need to dynamically
generate images
based on data
from my Rails application.

Creating Images with Ruby-Processing

Step 1: Proof of Concept

rp_sample.rb

```
1  def setup
2    @x_max = 100
3    @y_max = 200
4    size @x_max, @y_max
5  end
6
7  def draw
8    background 255
9    fill rand(255),rand(255),rand(255)
10   oval 50,100,50, 50
11   save "sample.png"
12   exit
13 end
```

rp5 rp_sample.rb

Step 2: Goal Setting

Provided: "Week 6 Quiz"

Create:

- Vertical Text
- Sized by Text Length
- Use Silkscreen Font
- Defaults:
  - Size 8px
  - Black text on white

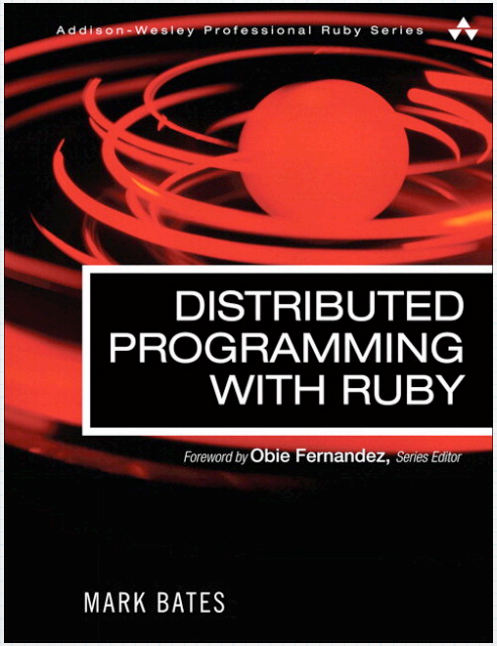
Step 3: R-P Algorithm

1. Build superstructure
2. Set options
3. Create the window
4. Move the origin to the bottom
5. Rotate the plane
6. Write the text
7. Save the file

Text Generator Demo

Parameters from Our Rails App

How can we pass messages from Rails to Ruby-Processing?



- Mark Bates
- Published 11/09
- Best tutorial for dRB, Rinda, etc

Beanstalkd

"Beanstalk is a simple, fast workqueue service."

Installing Beanstalkd

MacOS X

```
brew install beanstalkd
sudo port install beanstalkd
```

Fedora Linux

```
yum install beanstalkd
```

From Source

http://github.com/kr/beanstalkd

Windows??

http://github.com/ghazel/beanstalkd

Running Beanstalkd

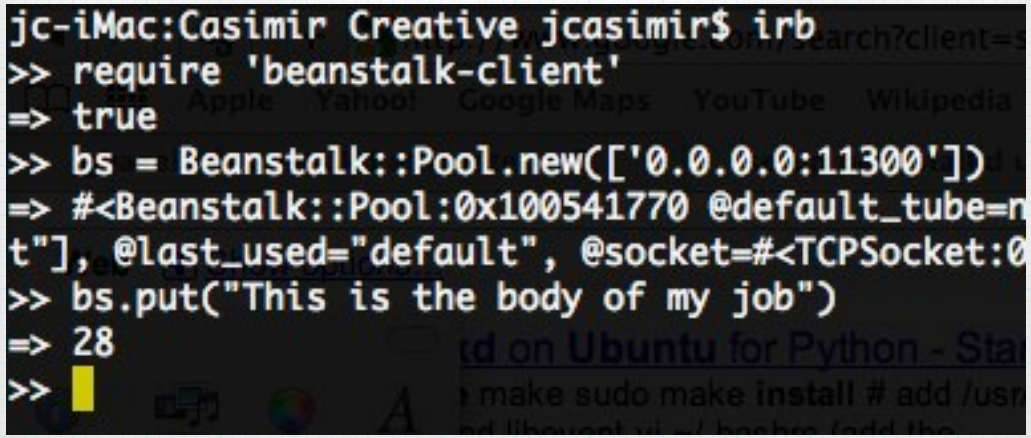
```
beanstalkd
```

(that's it!)

```
Defaults:
- Listen on 0.0.0.0:11300
- Run as current user
- Stay attached
```

Beanstalkd from Ruby

```
gem install beanstalk-client
```



Adding a Job to the Queue

Pulling a Job from the Queue

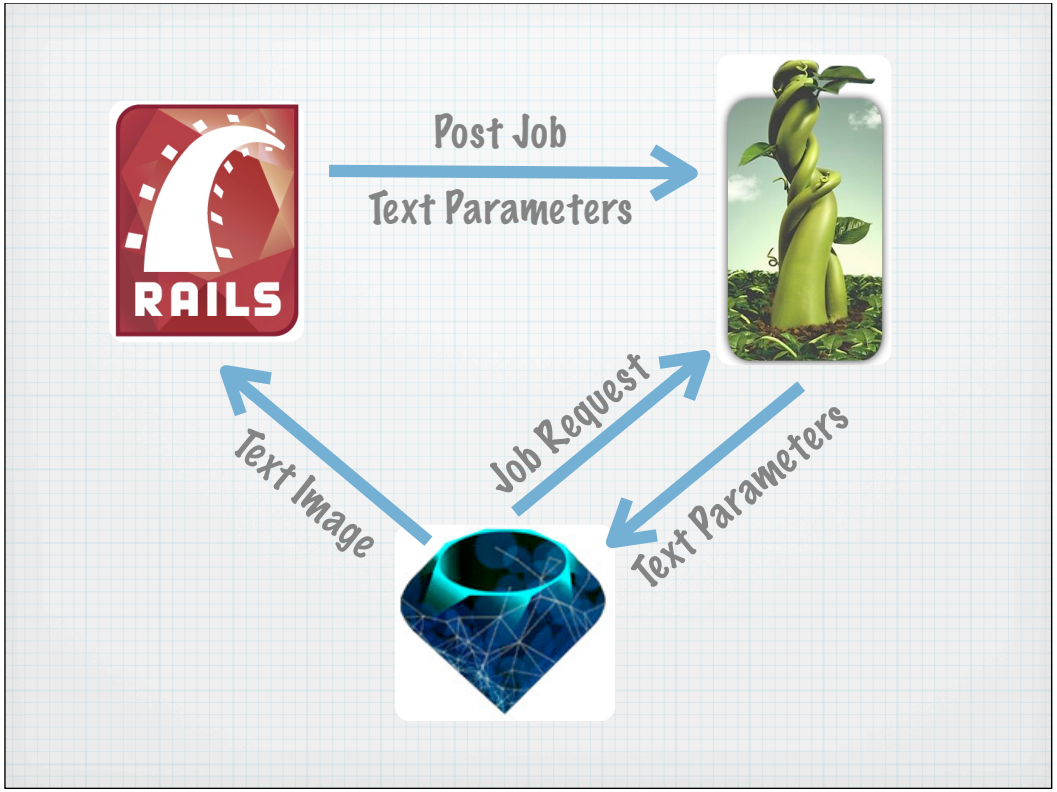
```
jc-iMac:Casimir Creative jcasimir$ irb
>> require 'beanstalk-client'
=> true
>> bs = Beanstalk::Pool.new(['0.0.0.0:11300'])
=> #<Beanstalk::Pool:0x100541770 @default_tube=
t"], @last_used="default", @socket=#<TCPSocket:
>> job = bs.reserve
=> (job server=0.0.0.0:11300 id=28 size=26)
>> puts job.body
This is the body of my job
=> nil
>> job.delete
=> true
>>
```

What if there's no Job?

```
>> job = bs.reserve
```

It sleeps!

Designing the Beanstalk Solution


RAILS
Post Job
Text Parameters
Job Request
Text Image
Text Parameters

How will this work?

Assignment Model

- When an assignment is saved...
  - Find the Beanstalk
  - Create a Job and supply the...
    - file_name
    - text

How will this work?

Ruby-Processing Worker

- Wait for a Job to hit the queue
- Load the parameters from the job body
- Create the image
- Delete the job
- Wait for the next job

Extra Challenges?

Beanstalkd works with plain "strings"

For more complex objects,
we'll need to serialize/unserialize

YAML is easy and fast

Implementing the Beanstalk Solution

Hooking into the Assignment Model

```ruby
class Assignment < ActiveRecord::Base
  attr_accessible :name
  has_many :grades

  after_save :generate_header_image
  def generate_header_image
    bs = Beanstalk::Pool.new(['0.0.0.0:11300'])
    body = {:file_name => image_file_name, :text => self.name}.to_yaml
    bs.put(body)
  end

  def image_file_name
    "assignment_id_#{self.id}"
  end
```

(that's the easy part)

R-P Can't Easily Use Gems

```
jc-iMac:code jcasimir$ mkdir vendor
jc-iMac:code jcasimir$ cd vendor
jc-iMac:vendor jcasimir$ mkdir gems
jc-iMac:vendor jcasimir$ cd gems
jc-iMac:gems jcasimir$ gem unpack beanstalk-client
Unpacked gem: '/Users/jcasimir/Dropbox/Presentations
jc-iMac:gems jcasimir$
```

Unpack the `beanstalk-client` gem into `vendor/gems/beanstalk-client-1.0.2`

Text Generator For Beanstalk

Ta-Da!

Try creating new assignments and editing existing assignments through the web interface

What Have We Done?

- Built a distributed message passing framework
- Built a Ruby-Processing powered worker process
- Rolled it all into a Rails app

So what now?

"I was promised video!"

Video in Ruby-Processing

Demo of

`video_tagger.rb`

Next Steps

On the Server Side

Xvfb

god

Going Beyond Prototypes

Replace the Workers, Keep the Process

Efficient Workers

2D Images	Textorize ImageMagick
3D Images	Ogre3D OpenGL
Audio & Video	ffmpeg Panda

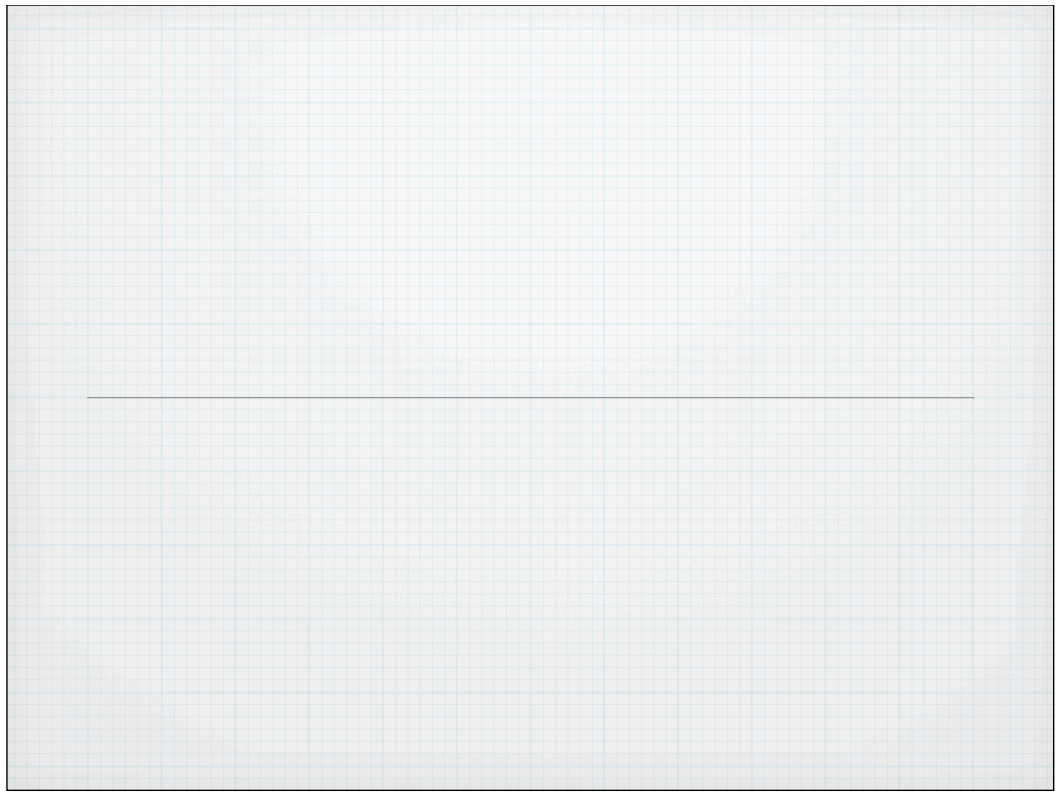
Wrapup

The Links

Presentation & Code	http://github.com/jcasimir/Code-of-Art-2
Processing	http://processing.org
Ruby-Processing	http://github.com/jashkenas/ruby-processing
Learning Processing (book)	http://rubyurl.com/3Tu6
Learning Processing (examples)	http://github.com/jashkenas/learning-processing-with-ruby
Beanstalk	http://kr.github.com/beanstalkd/
Beanstalk-Client	http://beanstalk.rubyforge.org/
Silkscreen	http://kottke.org/plus/type/silkscreen/


JumpstartLab
RubyJumpstart
Intro to Programming with Ruby
April 17 & 18
FlexJumpstart
Intro to Design with Adobe Flex
April 24 & 25
RailsJumpstart
Level 2 Rails Development
(BDD, ActiveRecord, HAML)
May 22 & 23
THE HASHROCKET WAY
Professional Rails Development
June 6 in Baltimore

Thanks!



The Following Slides are great for a Two-Screen Presentation, but are obsolete for One-Screen

Step 4: Implementation

1. Build Superstructure

rp_text_generator_step1.rb

```
1  def setup
2
3  end
4
5  def load_parameters
6
7  end
8
9  def draw
10
11 end
```

2. Set Options & 3. Create Window

rp_text_generator_step2.rb

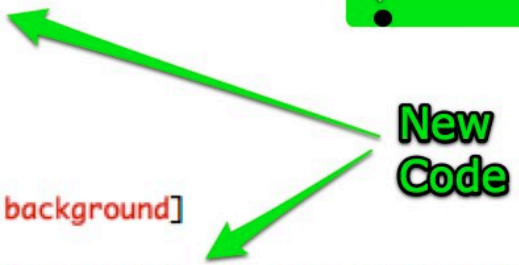
```
1  def setup
2    @params = load_parameters
3    size @params[:width], @params[:height]
4  end
5
6  def load_parameters
7    defaults = {
8      :width => 100,
9      :height => 100,
10     :background => 255
11   }
12 end
13
14 def draw
15   background @params[:background]
16 end
```

4. Move the Origin

```
def load_parameters
  defaults = {
    :width => 100,
    :height => 100,
    :background => 255,
    :line_height => 8
  }
end

def draw
  background @params[:background]

  translate @params[:line_height] -2, @params[:height]
end
```

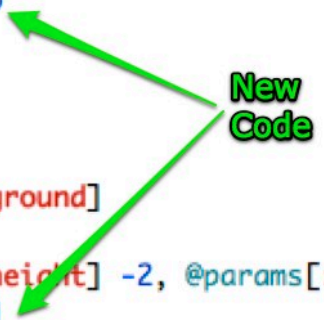


5. Rotate the Plane

```
def load_parameters
  defaults = {
    :width => 100,
    :height => 100,
    :background => 255,
    :line_height => 8,
    :rotation = radians -90
  }
end

def draw
  background @params[:background]

  translate @params[:line_height] -2, @params[:height]
  rotate @params[:rotation]
end
```



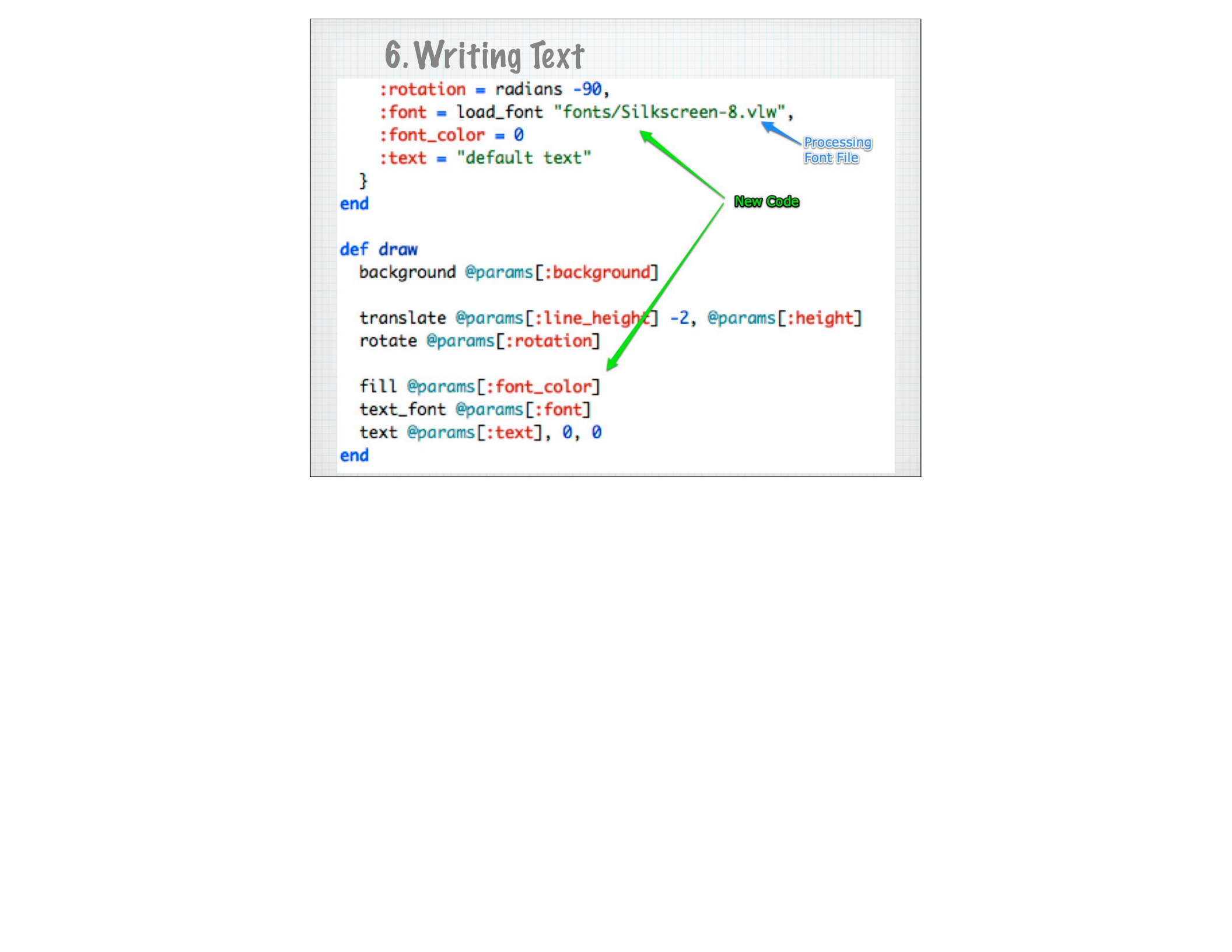
6. Writing Text

```
    :rotation = radians -90,
    :font = load_font "fonts/Silkscreen-8.vlw",
    :font_color = 0
    :text = "default text"
  }
end

def draw
  background @params[:background]

  translate @params[:line_height] -2, @params[:height]
  rotate @params[:rotation]

  fill @params[:font_color]
  text_font @params[:font]
  text @params[:text], 0, 0
end
```



7. Save the File

```
    :text => "default text",
    :file_name => "default",
    :file_type => "png"
  }
end

def draw
  background @params[:background]

  translate @params[:line_height] -2, @params[:height]
  rotate @params[:rotation]

  fill @params[:font_color]
  text_font @params[:font]
  text @params[:text], 0, 0

  save @params[:file_name] + "." + @params[:file_type]
  exit
end
```

Step 5: Refactor

1. Condense Window Size

```
def load_parameters
  defaults = {
    :background => 255,
    :line_height => 8,
    :character_width => 5,
    :rotation => radians(-90),
    :font => load_font("fonts/Silkscreen-8.vlw"),
    :font_color => 0,
    :text => "default text",
    :file_name => "default",
    :file_type => "png"
  }

  defaults[:width] = defaults[:line_height]
  defaults[:height] = (defaults[:text].length + 1) * defaults[:character_width]
  defaults
end
```

Additional Parameter

Calculated to fit

R-P Can't Easily Use Gems

```
1  Dir.glob(File.join("vendor", "gems", "*", "lib")).each do |lib|
2    $LOAD_PATH.unshift(File.expand_path(lib))
3  end
4
5  require 'beanstalk-client'
6  require 'yaml'
7
```

- Modify the $LOAD_PATH to find the new vendor folder
- Require beanstalk and yaml

Modifying the `setup`

```
def setup
  #@params = load_parameters
  #size @params[:width], @params[:height]
  size 200, 200
  @beanstalk = Beanstalk::Pool.new(['0.0.0.0:11300'])
end
```

- Set an arbitrary window size
- Find the beanstalk queue

Modifying `load_parameters`

```
def load_parameters()
  defaults = {
    :background => 255,
    :line_height => 8,
    :character_width => 5,
    :rotation => radians(-90),
    :font => load_font("fonts/Silkscreen-8.vlw"),
    :font_color => 0,
    :text => "default text",
    :file_name => "default",
    :file_type => "png"
  }
```

Leave the defaults alone

Modifying `load_parameters`

```
job = @beanstalk.reserve
values = defaults.merge(YAML.load(job.body))
job.delete
```

1. Wait for a job with `reserve`
2. When a job is found...
   a. Get the body
   b. un-YAML it
   c. merge with defaults
3. Remove the job

Modifying `load_parameters`

```
  values[:width] = values[:line_height]
  values[:height] = (values[:text].length * values[:character_width] * 1.07).to_i
  values
end
```

1. Change `defaults` in calculations to `values`
b. Changed the scaling formula slightly (not important)

Changes to `draw`

```
39 def draw
40   @params = load_parameters
41   size @params[:width], @params[:height]
42   background @params[:background]
```

1. Load `@params` each time `draw` runs instead of once in `setup`
2. Set the image `size` after each params load

Changes to `draw`

```
44  translate @params[:line_height] - 2, @params[:height]
45  rotate @params[:rotation]
46
47  fill @params[:font_color]
48  text_font @params[:font]
49  text @params[:text], 0, 0
```

No change to the text rendering

Changes to `draw`

```
51    save TARGET_DIR + @params[:file_name] + "." + @params[:file_type]
52  end
```

1. Add the `TARGET_DIR` to the `save`

2. Remove the `exit` command so Ruby-Processing keeps looping `draw`

Setting File Paths

```
8  TARGET_DIR = "./tabler/public/images/generated/"
```

- Set a target directory constant just below the requires
- Make sure the directory exists

Getting Ready to Test

1. Make sure `beanstalkd` is running
2. Start up the `tabler` (Rails) server
3. Start the image processor with

   ```
   rp5 run rp_text_generator.rb
   ```

4. Start a `script/console` session for experimenting

Testing from Console

1. Create a New Assignment

   ```
   Assignment.create(:name =>
   “My Assignment”)
   ```

2. Look for an `assignment_id_x.png` in `tabler/public/images/generated` that has the text `My Assignment`

3. Update all images with:

   ```
   Assignment.all.each{|a| a.save}
   ```

Implementing in the View

1. Open the view

   `assignments/index.html.haml`

2. Change the `%th` that outputs the assignment name to:

   `%th= image_tag(a.image_path)`

3. Add `image_path` method to `Assignment`

```
16  def image_path
17    "generated/#{image_file_name}.png"
18  end
```