Ruby Fundies: Guide to Teaching

This document provides details into the course philosophy and how it should be taught

Publication Date: March 1, 2017

Author: Ethan Puzarne - epuzarne@gmail.com

Table of Contents

Technology

IDE: Local vs Online

- **Programming is hard:** even without the challenges of local development
- **Prefer Online:** The curriculum is designed for the Repl.it online IDE
- **Until students are ready:** Local development and learning to use a debugger is important, so if the students are ready for it, a guide is available here

Slack

- **Avoid Email:** Slack should be THE communication tool
- **Public group discussion:** Make sure your students are comfortable using Slack and are using the shared public channel
- **Install desktop app:** Encourage students to use desktop app instead of web app

Google Hangouts

- **Sick or Traveling students**: Use hangouts to engage students that cannot attend physically
- **Sharing in class:** Hangouts are also useful for displaying student screens rather than passing around an HDMI cord

Student Engagement

Create leaders

- **Encourage the students to lead the class**: Student teachers are much more approachable than the main instructor. If no one steps up, select a mid-range or advanced student to lead the class.
- **Let the students drive**: When students do the coding, it makes misunderstandings obvious
- **Focus on struggling students**: While the class is learning from a peer, the teacher has the time to focus on struggling students

Protect Students From Themselves

- **Address confusion:** Students worry about slowing down class and will frequently say "I'll just figure it out at home". Demonstrate that it is ok to be confused and that the class can work through that confusion together
- **Encourage note taking:** Tell students to record their Aha! Moments
- **Don't rely on homework:** Students don't do it. Teach everything in class.
- **Discourage "Perfect" code:** Students want to know the right way to do something. Tell them to make it work first

Overachievers

- **Push them to lead**: Overachievers make perfect candidates for leaders. Teaching the other students will solidify their knowledge
- **Encourage them:** They can follow the scripts at their own pace and you can provide them with additional ideas, readings, and projects
- **Teach them some style**: If a student understands the material really well, feel free to give feedback on style and how to make their code readable and good

Meta Skills

Googling

- **Let google answer the easy questions**: “is there a method for ___?”, “what does ____ mean?”, etc.

Debugging

Teach the students how to debug instead of simply answering their questions

- **Errors:** Explain the call stack, line numbers, and reading error messages
- **Puts:** use `puts object` and `puts object.inspect`
- **Pry:** When class graduates to local development, make sure they are using a real debugger. The curriculum specifically teaches Pry.

Designing

- **Talk through the practice projects:** The explicit instructions provided do not let students practice “program design”. Prior to building each practice project talk about the program and walk through the design process

Refrain from Teaching

Abstraction

Write explicit and concrete code first. The code will be ugly and repetitive, but it will also be easier to understand. Only abstract once the concrete code is understood

Example from Week 4

Bad

```
# Print the first row of the Tic Tac Toe board
# Print a space for any nil on the board

# This is not explicit and uses lots of abstractions
def print_first_row(board)
  puts board[0..2].map{ |space| space || ' ' }.join('|')
end
```

Good

```
def print_first_row(board)
  if board[0] != nil
    print board[0]
  else
    print ' '
  end

  print '|'

  if board[1] != nil
    print board[1]
  else
    print ' '
  end
```

```
  print '|'

  if board[2] != nil
    print board[2]
  else
    print ' '
  end

  puts
end
```

Truthiness

Avoid teaching truthiness and control flow syntactic sugar shortcuts until students are comfortable with the basics. Show one explicit way of accomplishing a task

Example:

Bad

```
def my_method(given_object)
  # Avoid these shortcuts
  puts "hello" if given_object

  # Avoid using "unless"
  puts "hello" unless given_object
end
```

Good

```
def my_method(given_object)
  # Prefer the explicit
  if given_object != nil
    puts "hello"
  end
end
```

Shortcuts

Avoid using shortcut code

Example:

Bad

```
# Fallback Assignment
y = nil
x = y || 123

# Ternary Operator
x ? puts "I'm here" : puts 'not here'
```

Good

```
# Prefer if statement
if y
  x = y
else
  x = 123
end
```

Methods are not “Functions”

Use the word “Method”. Don’t use the word function. Ever. *Seriously. Don’t do it. Ruby has no functions and it confuses students.*

Course Modifications

A Work In Progress

This curriculum is not finished. It was designed based on feedback from a single cohort. Each new teacher will have their own ideas and each new class will have their own needs.

Ownership

This curriculum belongs to the teachers and the students. It is up to you to take ownership of improving and refining this guide.

Github

Please submit issues, changes, and PRs on **github**

Google Docs

Please leave feedback on the various google documents:

Course Description: Main course explanation

Guide to teaching: A high level guide to teaching the course (This Document

Weekly Student Instructions: This file contains high level instructions for the students for each week. Any changes to the curriculum flow and structure will need to be reflected here

Weekly Teacher Instructions: This file contains weekly instructions for the Teacher on how the class should flow