Chapter 4 Rails-flavored Ruby

Create a separate topic branch to keep our changes.

- $ git checkout -b rails-flavored-ruby

Build-in helper

Use the built-in Rails function stylesheet_link_tag to include application.css for all media types:
<%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>

Custom helpers

- In addition to coming equipped with a large number of built-in functions for use in the views, Rails also allows the creation of new ones. Such functions are called helpers.
- full_title helper to solve the problem of a missing page title (L4.2)

Strings and methods

Rails console - a command-line program for interacting with Rails applications.

Include recommended irb configuration parameters:

- $ nano ~/.irbrc #open a file
- Fill it the the contents:
  IRB.conf[:PROMPT_MODE] = :SIMPLE
  IRB.conf[:AUTO_INDENT_MODE] = false # ~/.irbrc
- Exit nano and save changes: Ctrl+X

Star/Exit console:

- $ rails console
- Ctrl+C, Ctrl+D exit console.

Comments starts with the pound sign: #

Strings

- String literals are created using the ": “foo”
- Concatenate strings with the +: “foo” + “bar” # also works with single quoted strings: ‘foo’ + ‘bar’
- Another way to build up strings is via interpolation using the special syntax #{}: fn=”Michael”; “#{fn} Hartl” #but not works with single quoted strings
- Printing cmd: puts # returns literally nothing: nil
- nil is a special Ruby value for “nothing at all”, nil object is special, in that it is the only Ruby object that is false in a boolean context, apart from false itself.
- print command prints the raw string without the extra line: print “foo”
- Ruby won’t interpolate into single-quoted strings
- Benefits of single quoted strings: they are truly literal, containing exactly the characters you type

Objects and message passing

- Everything in Ruby, including strings and even nil, is an object.
- String methods: .length; .empty?; .nil; .include?(“string”); .downcase; .upcase
- &&, ||, !
- if…else…end; if…elsif…elsif…end
- convert virtually any object to a string: nil.to_s
- unless keyword:
  >> string = "foobar"

>> puts "The string '#{string}' is nonempty." unless string.empty?

Methods

```
def method_name(para_name='')        #contains a default argument
  ...
end
```

- Ruby functions have an implicit return, meaning they return the last statement evaluated
- Ruby also has an explicit return option

Arrays and ranges

- Split a string into an array: "foo bar baz".split; "fooxbarxbazx".split('x')
- Ruby uses square brackets for array access. The first element of an array in the array has index 0. Indices can even be negative: a = [42, 8, 17]; a[0] = 42; a[-1] = 17;
- Ruby offers synonyms for some commonly accessed elements: a.first = 42; a.last = 17
- Array methods (a will remain the same): a.length; a.empty?; a.include?(42); a.sort; a.reverse; a.shuffle
- To mutate an array, use the corresponding "bang" method: a.sort!
- Add to arrays: a.push(6); a << 7; a << "foo" << "bar"
- Ruby arrays can contain a mixture of different types
- Join an array: a.join; a.join(',')
- Use %w to make a string array: sa = %w[foo bar baz quux]
- Ranges to array, work with numbers and characters: (0..9).to_a; ('a'..'e').to_a
- Ranges are useful for pulling out array elements: sa[0..2] => ["foo", "bar", "baz"]
- ('a'..'z').to_a.reverse

Blocks

- Both arrays and ranges respond to a host of methods that accept blocks.
- The vertical bars around the variable name in |i| are Ruby syntax for a block variable
- Use curly braces only for short one-line blocks and the do..end syntax for longer one-liners and for multi-line blocks
- The *map* method returns the result of applying the given block to each element in the array or range: >> %w[a b c].map { |char| char.upcase } => ["A", "B", "C"]
- The block inside map involves calling a particular method on the block variable, and in this case there's a commonly used shorthand called "symbol-to-proc": >> %w[A B C].map(&:downcase) => ["a", "b", "c"]

Hashes and symbols

- Hashes are essentially arrays that aren't limited to integer indices. Hash indices, or keys, can be almost any object.
- Hashes are indicated with curly braces containing key-value pairs; a pair of braces with no key-value pairs—i.e., {}—is an empty hash.
- Although hashes resemble arrays, one important difference is that hashes don't generally guarantee keeping their elements in a particular order. If order matters, use an array.
- It's easy to use a literal representation with keys and values separated by =>, called a "hashrocket"
  >> user = { "first_name" => "Michael", "last_name" => "Hartl" }
- Symbols look kind of like strings, but prefixed with a colon instead of surrounded by quotes. :symbol_name
- Hashes-of-hashes, or nested hashes: params[:user] = { name: "Michael Hartl", email: "mhartl@example.com" }
- each method: a hash iterates through the hash one key-value pair at a time
- inspect method: returns a string with a literal representation of the object it's called on

CSS revisited

- Parentheses on function calls are optional
- When hashes are the last argument in a function call, the curly braces are optional.
- <%= %> for inserting results

Ruby classes

- Constructors
- trace back the class hierarchy: s.class.superclass

Modifying build-in classes

- Ruby classes can be opened and modified, like the String class

To finish:

- $ git add -A
- $ git commit -am "Add a full_title helper"
- $ git checkout master
- $ git merge rails-flavored-ruby
- $ rails test
- $ git push
- $ git push heroku