learn enough

RUBY

TO BE DANGEROUS

MICHAEL HARTL

Praise for Learn Enough Tutorials

"Just started the #100DaysOfCode journey. Today marks day 1. I have completed @mhartl's great Ruby tutorial at @LearnEnough and am looking forward to starting on Ruby on Rails from tomorrow. Onwards and upwards."
—Optimize Prime (@_optimize), Twitter post

"Ruby and Sinatra and Heroku, oh my! Almost done with this live web application. It may be a simple palindrome app, but it's also simply exciting! #100DaysOfCode #ruby @LearnEnough #ABC #AlwaysBeCoding #sinatra #heroku"
—Tonia Del Priore (@toninjaa), Twitter post
Software Engineer for a FinTech Startup for 3+ years

"I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!"
—Michael King

"I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read."
—Pedro Iatzky

```
>> now = Time.now.utc
=> 2018-08-15 02:20:31 UTC
```

Finally, **`Time`** instances can be subtracted from each other:

```
>> now - moon_landing
=> 1548482571.0
```

The result here is the number of seconds since the day and time of the first Moon landing (Figure 4.2).[5] (Your results, of course, will vary, because time marches on, and your value for **`Time.now`** will differ.)

You may have noticed that the month and day are returned as *unit-offset* values, which differs from the zero-offset indexing used for arrays (Section 3.2). For example, in the eighth month (August), the return value of **`now.month`** is **`8`** rather than **`7`** (as it would be if months were being treated like indices of a zero-offset array). There is one important value that is returned as a zero-offset index, though:

```
>> moon_landing.wday        # wday = weekday
=> 0
```

Here **`wday`** is the "weekday" method, and the **`0`** index indicates that the Moon landing happened on the "zeroth" (first) day of the week.

Even though the official international standard is that Monday is the first day, Ruby follows the American convention of using Sunday instead. We can get the name of the day by making an array of strings for the days of the week (assigned to an ALL CAPS identifier, indicating a constant), and then using **`wday`** as an index in the array with the square bracket notation (Section 3.1):

```
>> DAYNAMES = ["Sunday", "Monday", "Tuesday", "Wednesday",
?>             "Thursday", "Friday", "Saturday"]
>> DAYNAMES[moon_landing.wday]
=> "Sunday"
>> DAYNAMES[Time.now.wday]
=> "Tuesday"
```

[5] Image courtesy of Castleski/Shutterstock.

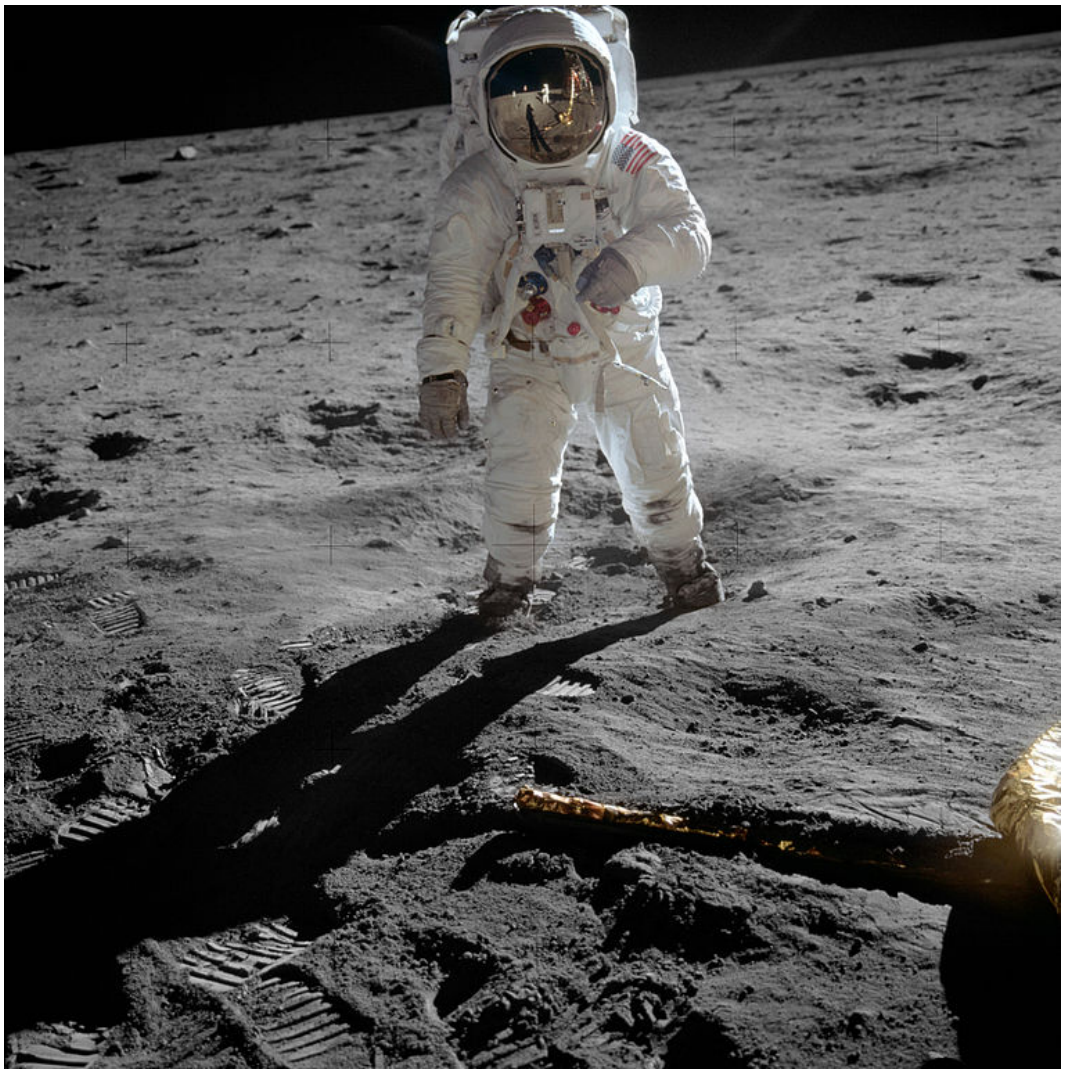

Figure 4.2: Buzz Aldrin and Neil Armstrong somehow got to the Moon (and back!) without Ruby.

(These day names are actually available as part of the `Date` class via `Date::DAYNAMES`. You just have to run `require 'date'` to load the class.) Your results for the last line will vary, of course, unless you happen to be reading this on a Tuesday.

As a final exercise, let's update our Sinatra hello app from Listing 1.8 with a greeting including the day of the week. The code appears in Listing 4.2, with the result as shown in Figure 4.3.

Listing 4.2: Adding a greeting customized to the day of the week.
hello_app.rb

```
require 'sinatra'

get '/' do
  DAYNAMES = ["Sunday", "Monday", "Tuesday", "Wednesday",
              "Thursday", "Friday", "Saturday"]
  dayname = DAYNAMES[Time.now.wday]
  "Hello, world! Happy #{dayname}."
end
```

Note that we've switched the greeting in Listing 4.2 from a single- to a double-quoted string so that we can interpolate the `dayname` value into the greeting.

4.2.1 Exercises

1. Use Ruby to calculate how many seconds after the Moon landing you were born. (Or maybe you were even born *before* the Moon landing—in which case, lucky you! I hope you got to watch it on TV.)

2. Show that Listing 4.2 works even if you pull `DAYNAMES` out of the `get`, as shown in Listing 4.3. (Remember to restart the server.) Then use the `Date` class to eliminate the variable entirely (Listing 4.4). (Note that `date` is automatically required by `sinatra`, so there's no need to include it separately.)

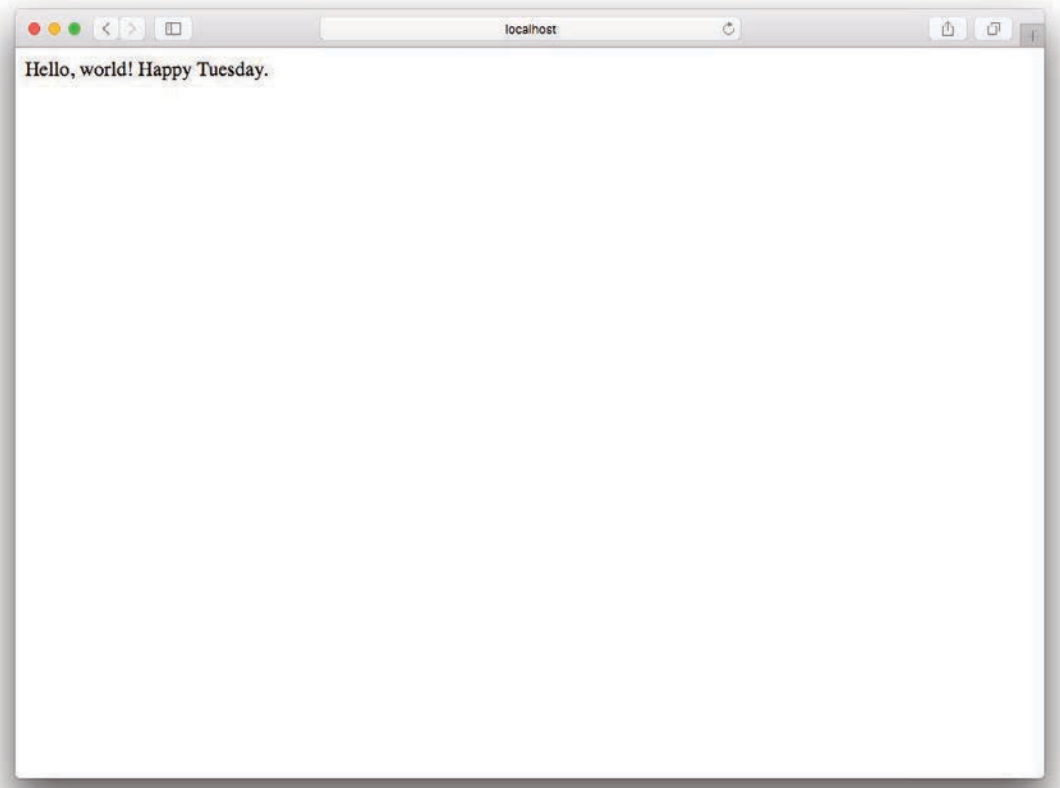

Figure 4.3: A greeting customized just for today.

Listing 4.3: Pulling `DAYNAMES` out of `get`.
hello_app.rb

```
require 'sinatra'

DAYNAMES = ["Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday"]

get '/' do
  dayname = DAYNAMES[Time.now.wday]
  "Hello, world! Happy #{dayname}."
end
```

Listing 4.4: Using the built-in `DAYNAMES`.
hello_app.rb

```
require 'sinatra'

get '/' do
  dayname = Date::DAYNAMES[Time.now.wday]
  "Hello, world! Happy #{dayname}."
end
```

4.3 Regular Expressions

Ruby has full support for *regular expressions*, often called *regexes* or *regexps* for short, which are a powerful mini-language for matching patterns in text. A full mastery of regular expressions is beyond the scope of this book (and perhaps beyond the scope of human ability), but the good news is that there are many resources available for learning about them incrementally. (Some such resources are mentioned in "Grepping" in *Learn Enough Command Line to Be Dangerous* and "Global find and replace" in *Learn Enough Text Editor to Be Dangerous*.) The most important thing to know about is the general idea of regular expressions; you can fill in the details as you go along.

Regexes are notoriously terse and error-prone; as programmer Jamie Zawinski famously said:

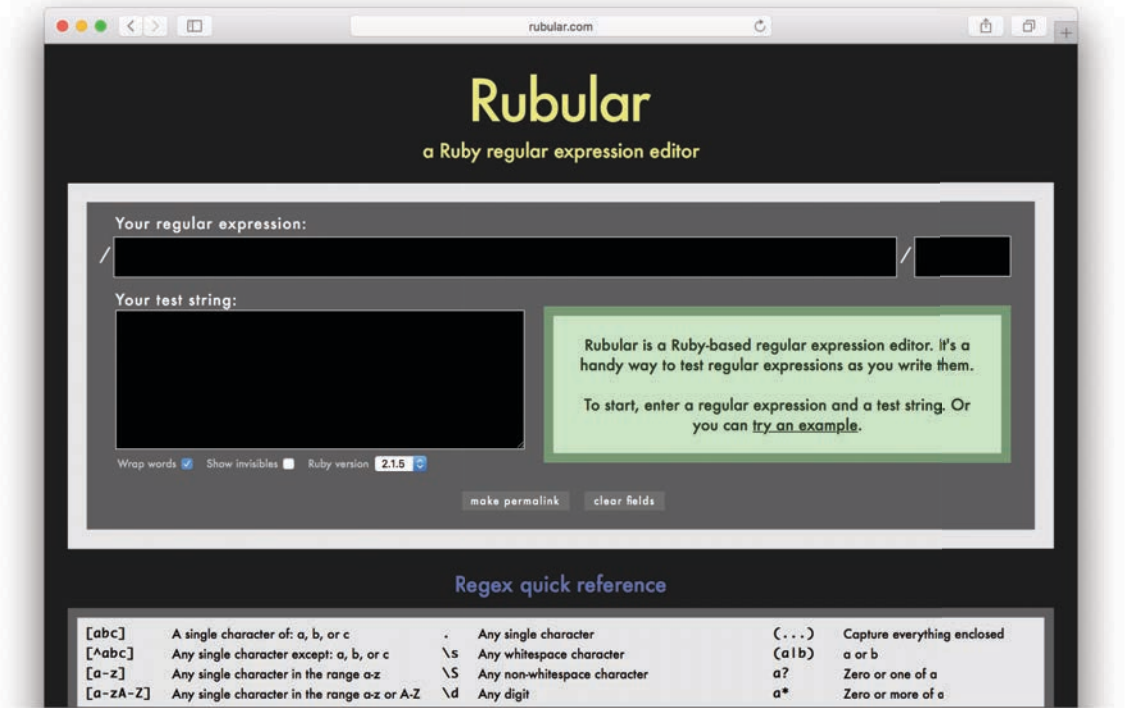

Figure 4.4: An online regex builder.

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

Luckily, this situation is greatly ameliorated by web applications like Rubular, which let us build up regexes interactively (Figure 4.4). Moreover, such resources typically include a quick reference to assist us in finding the code for matching particular patterns (Figure 4.5).

Rubular is Ruby-specific, but you can also use a site like the regex101 regex tester used in *Learn Enough JavaScript to Be Dangerous*. Regex101 doesn't have Ruby-specific support, but the "PHP" option should be good enough for most applications. In practice, languages differ little in their implementation of regular expressions, but it's wise to use the correct language-specific settings when available, and always to double-check when moving a regex to a different language.