Both arrays and ranges respond to a host of methods that accept *blocks*, which are simultaneously one of Ruby’s most powerful and most confusing features:

**>>** (1..5).each { |i| puts 2 \* i }

2

4

6

8

10

=> 1..5

This code calls the **each** method on the range **(1..5)** and passes it the block **{ |i| puts 2 \* i }**. The vertical bars around the variable name in **|i|** are Ruby syntax for a block variable, and it’s up to the method to know what to do with the block; in this case, the range’s **each** method can handle a block with a single local variable, which we’ve called **i**, and it just executes the block for each value in the range.

Curly braces are one way to indicate a block, but there is a second way as well:

**>>** (1..5).each **do** |i|

**?>**  puts 2 \* i

**>> end**

2

4

6

8

10

=> 1..5

Blocks can be more than one line, and often are. In the *Rails Tutorial* we’ll follow the common convention of using curly braces only for short one-line blocks and the **do..end** syntax for longer one-liners and for multi-line blocks:

**>>** (1..5).each **do** |number|

**?>**  puts 2 \* number

**>>**  puts '--'

**>> end**

2

--

4

--

6

--

8

--

10

--

=> 1..5

Here I’ve used **number** in place of **i** just to emphasize that any variable name will do.

Unless you already have a substantial programming background, there is no shortcut to understanding blocks; you just have to see them a lot, and eventually you’ll get used to them. Luckily, humans are quite good at making generalizations from concrete examples; here are a few more, including a couple using the **map** method:

**>>** 3.times { puts "Betelgeuse!" } *# 3.times takes a block with no variables.*

"Betelgeuse!"

"Betelgeuse!"

"Betelgeuse!"

=> 3

**>>** (1..5).map { |i| i\*\*2 } *# The \*\* notation is for 'power'.*

=> [1, 4, 9, 16, 25]

**>>** %w[a b c] *# Recall that %w makes string arrays.*

=> ["a", "b", "c"]

**>>** %w[a b c].map { |char| char.upcase }

=> ["A", "B", "C"]

As you can see, the **map** method returns the result of applying the given block to each element in the array or range.

By the way, we’re now in a position to understand the line of Ruby I threw into [Section 1.4.4](http://ruby.railstutorial.org/chapters/beginning#sec:heroku_commands) to generate random subdomains:

('a'..'z').to\_a.shuffle[0..7].join

Let’s build it up step-by-step:

**>>** ('a'..'z').to\_a *# An alphabet array*

=> ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",

"p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"]

**>>** ('a'..'z').to\_a.shuffle *# Shuffle it.*

=> ["c", "g", "l", "k", "h", "z", "s", "i", "n", "d", "y", "u", "t", "j", "q",

"b", "r", "o", "f", "e", "w", "v", "m", "a", "x", "p"]

**>>** ('a'..'z').to\_a.shuffle[0..7] *# Pull out the first eight elements.*

=> ["f", "w", "i", "a", "h", "p", "c", "x"]

**>>** ('a'..'z').to\_a.shuffle[0..7].join *# Join them together to make one string.*

=> "mznpybuj"