Arrays and Iterators

Chapter 7

Let's write a program which asks us to type in as many words as we want (one word per line, continuing until we just press Enter on an empty line), and which then repeats the words back to us in alphabetical order. OK?

So... first we'll—uh... um... hmmm... Well, we could—er... um...

You know, I don't think we can do it. We need a way to store an unknown amount of words, and how to keep track of them all together, so they don't get mixed up with other variables. We need to put them in some sort of a list. We need arrays.

An array is just a list in your computer. Every slot in the list acts like a variable: you can see what object a particular slot points to, and you can make it point to a different object. Let's take a look at some arrays:

```
[]
[5]
['Hello', 'Goodbye']

flavor = 'vanilla'             # This is not an array, of course...
[89.9, flavor, [true, false]]  # ...but this is.
```

So first we have an empty array, then an array holding a single number, then an array holding two strings. Next, we have a simple assignment; then an array holding three objects, the last of which is the array [true, false]. Remember, variables aren't objects, so our last array is really pointing to float, a string, and an array. Even if we were to set flavor to point to something else, that wouldn't change the array.

To help us find a particular object in an array, each slot is given an index number. Programmers (and, incidentally, most mathematicians) start counting from zero, though, so the first slot in the array is slot zero. Here's how we would reference the objects in an array:

```
names = ['Ada', 'Belle', 'Chris']

puts names
puts names[0]
puts names[1]
puts names[2]
puts names[3]  # This is out of range.
```

```
Ada
Belle
Chris
Ada
```

```
Belle
Chris
```

So, we see that puts names prints each name in the array names. Then we use puts names[0] to print out the "first" name in the array, and puts names[1] to print the "second"... I'm sure this seems confusing, but you do get used to it. You just have to really start thinking that counting begins at zero, and stop using words like "first" and "second". If you go out to a five-course meal, don't talk about the "first" course; talk about course zero (and in your head, be thinking course[0]). You have five fingers on your right hand, and their numbers are 0, 1, 2, 3, and 4. My wife and I are jugglers. When we juggle six clubs, we are juggling clubs 0-5. Hopefully in the next few months, we'll be able to juggle club 6 (and thus be juggling seven clubs between us). You'll know you've got it when you start using the word "zeroth". 😃 Yes, it's a real word; ask any programmer or mathematician.

Finally, we tried puts names[3], just to see what would happen. Were you expecting an error? Sometimes when you ask a question, your question doesn't make sense (at least to your computer); that's when you get an error. Sometimes, however, you can ask a question and the answer is nothing. What's in slot three? Nothing. What is names[3]? nil: Ruby's way of saying "nothing". nil is a special object which basically means "not any other object." And when you puts nil, it prints out nothing. (Just a new line.)

If all this funny numbering of array slots is getting to you, fear not! Often, we can avoid them completely by using various array methods, like this one:

The Method each

each allows us to do something (whatever we want) to each object the array points to. So, if we want to say something nice about each language in the array below, we'd do this:

```ruby
languages = ['English', 'German', 'Ruby']

languages.each do |lang|
  puts 'I love ' + lang + '!'
  puts 'Don\'t you?'
end

puts 'And let\'s hear it for C++!'
puts '...'
```

```
I love English!
Don't you?
I love German!
Don't you?
I love Ruby!
Don't you?
And let's hear it for C++!
```

So what just happened? Well, we were able to go through every object in the array without using any numbers, so that's definitely nice. Translating into English, the above program reads something like: For each object in languages, point the variable lang to the object and then do everything I tell you to, until you come to the end. (Just so you know, C++ is another programming language. It's much harder to learn than Ruby; usually, a C++ program will be many times longer than a Ruby program which does the same thing.)

You might be thinking to yourself, "This is a lot like the loops we learned about earlier." Yep, it's similar. One important difference is that the method each is just that: a method. while and end (much like do, if, else, and all the other blue words) are not methods. They are a fundamental part of the Ruby language, just like = and parentheses; kind of like punctuation marks in English.

But not each; each is just another array method. Methods like each which "act like" loops are often called iterators.

One thing to notice about iterators is that they are always followed by do...end. while and if never had a do near them; we only use do with iterators.

Here's another cute little iterator, but it's not an array method... it's an integer method!

```
3.times do
  puts 'Hip-Hip-Hooray!'
end
```

```
Hip-Hip-Hooray!
Hip-Hip-Hooray!
Hip-Hip-Hooray!
```

More Array Methods

So we've learned each, but there are many other array methods... almost as many as there are string methods! In fact, some of them (like length, reverse, +, and *) work just like they do for strings, except that they operate on the slots of the array rather than the letters of the string. Others, like last and join, are specific to arrays. Still others, like push and pop, actually change the array. And just as with the string methods, you don't have to remember all of these, as long as you can remember where to find out about them (right here).

First, let's look at to_s and join. join works much like to_s does, except that it adds a string in between the array's objects. Let's take a look:

```
foods = ['artichoke', 'brioche', 'caramel']

puts foods
puts
puts foods.to_s
puts
puts foods.join(', ')
puts
puts foods.join('  :)  ') + '  8)'
```

```
200.times do
  puts []
end
```

```
artichoke
brioche
caramel

["artichoke", "brioche", "caramel"]

artichoke, brioche, caramel

artichoke  :)  brioche  :)  caramel  8)
```

As you can see, puts treats arrays differently from other objects: it just calls puts on each of the objects in the array. That's why putsing an empty array 200 times doesn't do anything; the array doesn't point to anything, so there's nothing to puts. (Doing nothing 200 times is still doing nothing.) Try putsing an array containing other arrays; does it do what you expected?

Also, did you notice that I left out the empty strings when I wanted to puts a blank line? It does the same thing.

Now let's take a look at push, pop, and last. The methods push and pop are sort of opposites, like + and - are. push adds an object to the end of your array, and pop removes the last object from the array (and tell you what it was). last is similar to pop in that it tells you what's at the end of the array, except that it leaves the array alone. Again, push and pop actually change the array:

```
favorites = []
favorites.push 'raindrops on roses'
favorites.push 'whiskey on kittens'

puts favorites[0]
puts favorites.last
puts favorites.length

puts favorites.pop
puts favorites
puts favorites.length
```

```
raindrops on roses
whiskey on kittens
2
whiskey on kittens
raindrops on roses
1
```

A Few Things to Try

Write the program we talked about at the very beginning of this chapter. Hint: There's a lovely array method which will give you a sorted version of an array: sort. Use it!

Try writing the above program without using the sort method. A large part of programming is solving problems, so get all the practice you can!

Rewrite your Table of Contents program (from the chapter on methods). Start the program with an array holding all of the information for your Table of Contents (chapter names, page numbers, etc.). Then print out the information from the array in a beautifully formatted Table of Contents.

So far we have learned quite a number of different methods. Now it's time to learn how to make our own.