Yalies on Rails - Week 2

Part 1: Ruby

This week, we'll cover a few more concepts in Ruby: arrays, hashes, and iterators. After you know these, you'll have about 90% of the building blocks you need to write Rails Apps.

Arrays

If you're anything like me, you loooove making lists. Shopping lists, lists of ideas for the next great startup, lists of my favorite pokeman… just kidding, I don't really know what pokeman are. :)

Arrays are just that, ordered lists of things (or as we call 'things' in Ruby, *objects*). To tell Ruby that we're making a list, we use the `[` and `]` characters for the beginning and end, and commas to separate items.

Creating Arrays

Here, I've made a few arrays, and stored them in some variables (I'll be using these variables later to refer back to these lists:)

```
shopping_list = ["Milk", "Eggs", "Bacon", "More Bacon"]

startup_ideas = ["Twitter for Geese", "CaSh4Gold for Pets", "LinkedIn for Toddlers"]

favorite_pokemon = ["Seriously, I don't really know pokemon."]
```

Now you may have noticed that all these lists stored **strings** of text, but arrays can store any type of object. Here's an array of numbers:

```
lucky_numbers = [4, 8, 15, 16, 23, 42]
```

In fact, we can put any object(s) in an array, and we can even mix-n-match types of objects in the *same* array:

```
random_stuff = [12, "carrots", 199, "a caribou on the back of a salmon"]
```

See? We can put whatever we want in arrays. Awesome!

Stuffing Ourselves (Adding more items to an array)

I just took a look at my shopping list, and I realize there's not much variety. My cardiologist would probably have a heart attack himself if he saw it. Let's fix that by adding another item to the list.

```
#First, verify the contents of the array (we don't have to; this is just a reminder for you).
puts shopping_list
# ["Milk", "Eggs", "Bacon", "More Bacon"]

#OK now let's add another item:
shopping_list << "Canadian Bacon"

#That should make him happy. Let's check the result:
puts shopping_list
# ["Milk", "Eggs", "Bacon", "More Bacon", "Canadian Bacon"]
```

The `<<` is really just a method, but just like `+` and `-`, Ruby lets us use this method in a way that is easier for us to read and write. Adding an item to the end of an array like this is sometimes called *pushing* an item onto the array.

Fetch Lassie! (Retrieving from Arrays)

I don't know much about dogs, but if the TV commercials for *Beggin' Strips* brand dog snacks are true, they loooove bacon. I want to make sure I have bacon as the 3rd item in my shopping_list, so that I don't forget it.

There are a couple of ways to get items out of an array, but I'll only show you one right now. (We'll see another in the later section on **Iterators**).

Arrays are ordered lists, so we can get items out by using their order in the list, or **index**. In programming, we always start counting with **0**, not **1**. So the 'first' item in the list is at index **0**. The second item is at index **1**, and so on.

To get an item using its index, we use the method `[]`, passing the index as an argument.That may sound really confusing, but an example should make it clear:

```
#First item
puts shopping_list[0]
# "Milk"

#Now, let's get to the bacon
puts shopping_list[2]
# "Bacon"
```

Note that retrieving an item from an array *doesn't remove it from the array*, it's still there.

```
bacon_from_up_north = shopping_list[4]

puts bacon_from_up_north
# "Canadian Bacon"

puts shopping_list
# ["Milk", "Eggs", "Bacon", "More Bacon", "Canadian Bacon"]
```

Continuing Education

There are a lot more built-in methods to work with arrays. But instead of going over them here (really there are at least 50), I want you to get comfortable looking up documentation. So check out the documentation for array. I'd read over the following at least:

- `<<` and - (add)
- -
- compact
- delete_at
- first (and last)
- join
- length
- reverse

Arrays are great for lists where the items are ordered numerically, or where the order doesn't matter (it's not uncommon to ignore the indices of an array and work with the objects in other ways). But sometimes, you just want a little structure. That's where **hashes** come in.

Hashes

Hashes are like arrays in that they store collections of things. But while an array stores a list of ordered objects, hashes store objects in pairs. In each pair, we have a key and a value. I like to think of these pairs like entries in the dictionary. The key would be the word we're looking up, and the value is the block of text that defines the meaning of the word.

We usually create a hash using curly brackets `{` and `}`. Here's an example:

```
my_pets = { "Teeka" => "Cat", "Spaetzle" => "Cat", "Nigel" => "Hamster"}
```

The `=>` is used to indicate the relationship between the keys and values. This hash has three keys: "Teeka", "Spaetzle", and "Nigel". Each key is associated with one value, which in this case are "Cat", "Cat", and "Hamster".

Adding and Retrieving Pairs

To retrieve a value from a hash, we use a similar syntax as we do for arrays, replacing the index with the key:

```
puts my_pets["Teeka"]
# "Cat"

puts my_pets["Nigel"]
# "Hamster"
```

To add items to a hash, we also use a similar syntax as we did for arrays:

```
my_pets["Bob"] = "Turtle"

puts my_pets["Bob"]
# "Turtle"
```

If it fits, it sits (aka what can we put in these?)

The great thing about hashs (and arrays) is that we can store any object in them (including more arrays and hashes). With hashes, this is true for both the key and the value.

So far, the only objects we've shown you are strings, numbers, arrays, and hashes, but there are many others, and once you start creating your own objects, you'll probably want to store them in an array or hash as well, so keep that in mind!

Continuing Education

Again, there's lots more to the hash class than I've shown you. I suggest you check out the documentation for the Ruby Hash class.

Some useful methods are:

- keys
- values
- include?
- to_a

Iterators

Hashes and arrays are united by the fact that they are both collections of objects. While programming, we often have a collection of objects, which we'd like to work with one-by-one.

Iterating Over Arrays

For example, let's consider my shopping list. What if I wanted to print out a copy of my shopping list, but with each item in UPPERCASE so I can read it easier. I could do this with standard while loops, but it would be very messy:

```
#NOT IDEAL!
current_item_number = 0 #to keep track of what index we're on
shopping_list_size = shopping_list.size #In this case, 5
while item_number < shopping_list_size
    #Print the current item
    puts shopping_list[current_item_number].upcase
    #Increase our index, so
    current_item_number = current_item_number + 1
end
```

This produces the following output:

```
MILK
EGGS
BACON
MORE BACON
CANADIAN BACON
```

Thankfully, with Ruby, there's a better way, the `each` method. Here's that same program, rewritten with `each`:

```
shopping_list.each do |item|
    puts item.upcase
end
```

Much better, right? You may be wondering what the `do`, `end`, and `|` all mean. When we use the `each` method, Ruby expects a chunk of code to reapeat each time it gives us a new item in the array/hash. The `do` and `end` tell Ruby where that chunk (we call it a **block** of code) starts and ends.

The `|item|` is a way to reference the current item (out of all the items) in our block of code. The `item` part is just a variable we're creating on-the-fly, it could be called anything we want; we just have to make sure we refer to it using the same name in our block of code.

For example, this does exactly the same thing, even though it's a bit silly:

```
shopping_list.each do |chicken_nugget_sauce|
    puts chicken_nugget_sauce.upcase
end
```

Run it for yourself if you don't believe me. To Ruby, it doesn't matter that our variable is called `chicken_nugget_sauce`, it happily lets us use that variable name to refer to each item in the list.

Iterating Over Hashes

The concept is exactly the same for hashes, only we get two variables each time we iterate, one for the key, and one for the corresponding value.

Here's an example using `my_pets`:

```
my_pets.each do |name, species|
    puts "My pet " + name + " is a " + species
end
```

Gives us:

```
My pet Spaetzle is a Cat
My pet Nigel is a Hamster
My pet Teeka is a Cat
My pet Bob is a Turtle
```