Lecture 3: Containers

February 8, 2011

1 Variables so far

- String
- Integer
- Float
- Boolean

What if we have a whole bunch of numbers to carry around? Say we went to represent a whole list of exam scores, or individual card values from a blackjack hand?

In a sense, Strings are a collection of letters (characters) in sequence.

- We can remove characters
  - name.chomp
- We can find out the length
  - hometown.length
- We can check membership
  - name.include('h')

There’s an old myth that oysters are only good in months including the letter ‘r’:

```ruby
if month.include?('r')
  puts "Let’s have some oysters!"
end
```
2 Arrays

Arrays are a kind of container that can hold arbitrary values.

- Sometimes also called *lists*
- [] is an empty array
  - much like an empty tupperware container
- [1,2,3]
  - an array containing the values 1, 2, and 3
- ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
  - a list (array) of weekdays!
- Arrays have an ordering
  - we know that Monday comes before Tuesday
- How do we use them?
  - Along with this ordering comes a way to access things, with the [] syntax!
    - but they start with zero!
    - [1,2,3][1] => 2
      weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
      weekdays[0] => 'Monday'
- Arrays allow *mixed types*
  - ['Bob', 'Jones', 94]
    * Maybe Bob graduated from Tufts with the class of '94?
  - ['Bob', 'Jones', 94, 23, 91, 76]
    * Or perhaps that’s a list of homework scores we’re keeping track of?
    * What if we could separate them from the name?
  - ['Bob', 'Jones', [94, 23, 91, 76]]
    * a 3-element array. Not equivalent, but more useful?
  - [['Bob', 'Jones'], [94, 23, 91, 76]]
    * a 2-element array. Not equivalent, but perhaps more meaningful?
  - We can also substitute *variables*:
    firstname = 'Bob'
    lastname = 'Jones'
    scores = [94,23,91,76]
    record = [[firstname, lastname], scores] => [['Bob', 'Jones'], [94, 23, 91, 76]]
2.1 Array Methods

Just as Strings have some methods that deal with their elements, so do Arrays

- **include?**
  - `[1,2,3].include?(2) => true
  - `weekdays.include?('Sunday') => false`

- **length**
  - `['a', 'b', 'c'].length => 3`

- **first and last**
  - `['a', 'b', 'c'].first => 'a'
  - `['a', 'b', 'c'].last => 'c'

- **concatenation of two arrays**
  - `['parsley', 'sage'] + ['rosemary', 'thyme']
    => ['parsley', 'sage', 'rosemary', 'thyme']
  - requires two arrays
  - `[1, 2] + 3 does NOT WORK

- **destructive appending**

  ```ruby
  my_array = ['a', 'b', 'c']
  my_array << 'd'
  my_array => ['a', 'b', 'c', 'd']
  ```

Perhaps most interestingly, there is array enumeration

```ruby
weekdays = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']
weekdays.each do |day|
  puts "it's #{day}day!"
end
```

The thing inside the ‘pipes’ (`|day|` in this case) is called a block argument. It takes each element of the array in turn, and becomes a local variable to the block contained inside the `do..end`. Its name is arbitrary, but must be consistent between the block and the block argument.

Yes, this is a type of loop, and perhaps the most common type of loop you will use in Ruby programming. The syntax is a bit complicated, but understanding it will pay off soon!
2.2 Going between Arrays and Strings

We’ve talked about the types that various methods return. Let’s look at some inverse methods: \texttt{String.split} and \texttt{Array.join}

"Alpha Beta Gamma".split(’ ’) => ['Alpha', 'Beta', 'Gamma']

['Alpha', 'Beta', 'Gamma'].join(’ ’) => 'Alpha Beta Gamma'

'1,2,3,4'.split(',') => ['1','2','3','4'] # note: does not convert to Integer

[1,2,3,4].join(',') => [1,2,3,4] # note: DOES convert to String

3 Hashes

So, Arrays are a mapping from sequential numbers to any kind of value. Hashes are a mapping from any kind of key to any kind of value

- \{\} is an empty Hash
- \{'a' => 1, 'n' => 14\} is a mapping from the letters ‘a’ and ‘n’ to their positions in the alphabet

\{'Tufts' => 'Jumbos',
'Bowdoin' => 'Polar Bears',
'Amherst' => 'Lord Jeffs (are you serious!?)',
'Bates' => 'Totally lame'}
- is a mapping from schools to mascots.

- Somewhat obvious uses:
  - map from name to social security number
  - map from name to Tufts ID
  - map from social security number to an array of Tufts ID and Name

- How do we use them?
  - Indexed by key, so:
    - school_mascots['Tufts'] => 'Jumbos'
    - school_mascots['Harvard'] => nil (well, we didn’t define Harvard earlier, and a color is a lame mascot)

- Less obvious uses, nested hashes
Note that we can break lines after commas

```
{  
  991076255 => {  
                     :firstname => 'Noah',       
                     :lastname => 'Daniels',   
                     :department => 'Computer Science'  
  },  
  991076241 => {  
                     :firstname => 'Bob',       
                     :lastname => 'Jones',     
                     :department => 'Philosophy'  
  }  
}
```

What is this :firstname nonsense? It’s a symbol. It’s a lot like a string, and nice for use as a Hash key.

people[991076255][:department] => 'Computer Science'

### 3.1 Hash methods

Like Arrays, Hashes have some useful methods: - keys gives an Array of the keys - `{a => 1, n => 14}.keys => ['a', 'n']` - values gives an Array of the values - `{a => 1, n => 14}.values => [14, 1]` - What!? Order is NOT guaranteed, unlike with Arrays - You can also iterate, like with arrays:

```
people = {  
  991076255 => {  
                     :firstname => 'Noah',       
                     :lastname => 'Daniels',   
                     :department => 'Computer Science'  
  },  
  991076241 => {  
                     :firstname => 'Bob',       
                     :lastname => 'Jones',     
                     :department => 'Philosophy'  
  }  
}

people.each_pair do |tufts_id, person_record|  
  puts "person's name is #{person_record[:firstname]}"  
end
```

=>
Bob
Noah
• Again, order is not guaranteed

4 Decisions, decisions

4.1 When should you use an Array?

• When there is a natural ordering that you wish to preserve
• When you wish to access elements sequentially or numerically
• When you have a small number of elements and integers suffice as keys

4.2 When should you use a Hash?

• When the keys are most naturally represented as something other than a number
• When the keys are numbers, but they are mostly not sequential
• When the keys are numbers, and sequential, but the most important thing is *looking up an entry fast*