Singleton methods, symbol to proc and Ruby Equality

Singleton methods

- Can define methods on an individual instance of a class which will only apply to that particular instance of a class
- Class methods are an example of singleton methods in action

Singleton methods

```
class Dog
  attr_accessor :name
end
dog1 = Dog.new; dog1.name = "Mr. Doggy the 1st"
dog2 = Dog.new; dog2.name = "Mr. Doggy Dog the 2nd"

def dog2.informal
  "Cute doggie"
end
puts dog2.informal # => Cute doggie
# puts dog1.informal would output an error

p Dog.class # => Class
def Dog.dog_class
  "only available to the #{self} class"
end

puts Dog.dog_class # => only available to the Dog class
```

Symbol.to_proc trick

- Symbols have a `to_proc` method which converts a symbol to a `Proc` (similar to `lambda`)
- This could result in extremely concise code for methods like `Enumerable#map`
- Unfortunately, does not work if a method takes a parameter

Symbol.to_proc trick example

```
Person = Struct.new(:name, :age) do
  def old_enough?
    age > 18
  end
end

max = Person.new("Max Smith", 22)
joe = Person.new("Joe Grunt", 15)
suzy = Person.new("Suzy", 21)
ppl = [max, joe, suzy]

p ppl.select { |person| person.old_enough? }.map { |person| person.name}
# => ["Max Smith", "Suzy"]

# The same as above
p ppl.select(&:old_enough?).map(&:name) # => ["Max Smith", "Suzy"]
```

Equality in Ruby

- 4 equality methods:
  1. `equal?` - same exact object (Do not override)
  2. `==` - business equality regardless of class/type
     - Used most often
  3. `===` - Similar to `==` but lets you "compare" things like regexp and Ranges in case statements
  4. `eql?` – same as `==` but makes sure the class/type is the same as well

Equality in Ruby example

```
a = 1
b = 1.0
c = 1

puts a == b # => true
puts a.eql? b # => false
puts a.eql?(c) # => true
puts "one".equal? "one" #=> false

# Implicitly uses '==='
case a
  when /\w{3}/ then puts "Bingo!"
end # => Bingo!
```