Ruby Cheat Sheet

Conditionals

Ruby conditionals are pretty straigtforward:

```
if expression
  ...
elsif another_expression
  ...
else
  ...
end
```

Ruby also has unless, which is like not if:

```
unless it_is_safe
  stay_home
end
```

One nice thing in Ruby is the post-fix if which works like this:

```
go_outside if it_is_safe
```

Iteration and Enumerables

In Ruby you normally do not use a for or while loop. Instead in Ruby you use *.each* to iterate over an enumerable which is an object that yields elements one at a time.

```
[1, 2, 3].each do |element|
  puts element
end
```

You can even iterate through things like hashes with this method.

```
{foo: "bar", baz: "bang"}.each do |key, value|
  puts key
  puts value
end
```

There are of course more traditional ways to loop in Ruby. While loops get used occasionally:

```
while expression do
  do_some_work
end

loop do
  do_some_work
  break if i_want_to_quit
end
```

Functions and Classes

In Ruby the key words for creating new objects are "new" and "initialize". Then normal methods are defined with the standard syntax and class methods have a "self." in the function name. Functions that end in a question mark return a boolean by convention and methods that end in a bang alter the state of the original object.

```
class Transporter
  def initialize
    puts "starting up"
  end

  def beam_me_up
    puts "eye eye"
  end

  def self.model
    puts "Transporter2000"
  end
end
> t = Transporter.new
starting up
> t.beam_me_up
eye eye
> Transporter.model
Transporter2000
```

Arrays and Hashes

The two most common primitive objects you will work with in Ruby are Arrays and Hashes. Since Ruby is dynamically typed you can put whatever objects into the array and into the keys or values of the hash. You can even use random objects as keys in a Hash.

```
array = []
array << 3 # this is called the shovel operator
array[0]
 => 3
array.map { |x| x**2 }
 => [9]

hash = {}
hash[:foo] = "bar"
hash.merge!({1 => "one"})
 => {:foo=>"bar", 1=>"one"}
```

Strings and Symbols

In Ruby everything is an object, including Strings. Because of this every String has a different object ID and a different place in memory. Therefore Ruby also has symbols which are like strings but they are imutable and only have one instance in memory. In ruby you can also use #{} to do string iterpolation.

```
:symbol
"string"
"2 squared is #{2**2}"
 => "2 squared is 4"
```

Blocks and Yields

One of the unique aspects of Ruby is how blocks and yields work. This is what powers the ".each" method we saw with iteration but it can also be used for a lot of other things. A yield is where the method takes in a block as a sort of parameter and yields control to the block which has a lot of uses.

```
def double_and_calc(a, b)
  a = a * 2; b = b * 2
  yield(a, b)
end

double_and_calc(2, 3) { |a, b| a + b }
 => 10
double_and_calc(2, 1) do |a, b|
  a ** b # a raised to b power
end
 => 16
```

Gems and Bundling

In Ruby packages of code are called gems. They can either be installed manually or they can be listed in a project as part of a Gemfile.

```
# in terminal
gem install "pry"
# inside Gemfile
gem "pry"
```

Debugging

One of the most common ways to debug is to use a REPL (Read-Eval-Print-Loop) tool. A common tool for this is the pry gem. To debug a user puts a call to a debugger tool like pry in the middle of their code and then they can poke around in the code. One helpful rubyism is that you can call "method(:method_name).source_location" and Ruby will show you exactly where that method is defined.

```
def some_method
  binding.pry
  troublesome_method_call
end

> troublesome_method_call
 => StandardError
> method(:troublesome_method_call).source_location
 => "/user/code/app/models/something.rb#27"
```

Implicit Return

Ruby doesn't require an explicate return, which can sometimes confuse developers from other languages. These two methods are identical.

```
def square_no_return(x)
  x ** 2
end

def square_with_return(x)
  return x ** 2
end
```