# Phase 1 Quick References

Ruby, Git and GitHub, Rspec, Regular expressions, Markdown, SQL. This is what you need for phase 1. Updates, feel free to reach out to any of the **SF-Dragonflies-2015**

**G**ot an update, correction, let me know:

[mikefarr@mac.com](mailto:mikefarr@mac.com)

michaelmfarr@gmail.com

# Ruby

Everything is an Object, even basic datatypes. methods of an object are called using dot syntax.

Every method returns an object. Therefore methods and method-results can be chained.

25.**methods.sort**

### Numbers

1.even?

1.odd?

to\_s to\_sym to\_f round between chr

div next pred even ceil chr divmod

### Operators

**+ - \* / = == != > < >= <= [] \*\***

**. ::**

### 70’s style user input

**name = gets.chomp** **#read user input and remove the \n**

### Arrays

**ar=[] #empty array**

**ar = Array.new(8){Array.new(8)} #use block to init multi-dimensional arrays**

**words = ["foo", "bar", "baz"]**

**numbers = [1,2,3,4,5]**

**words[i]**  **#return ith element**

**words[-5]**  #**return 5th from end**

**words[0..1]**  **#return array with first two items**

**.first**

**.last**

**words << 'woot'**  **#append new element , push**

**words + sentences #concatenate arrays**

**words.concat(sentences) #same but modifies words**

Useful methods:

**.size**

**.length #number of elements**

.**pop #pops off last element, deleting it**

.**push(item) #pushes at end**

**.shift #returns first element, modifies array**

**.delete(obj)**  **#delete every element that matches. Note that 2**

**.delete(obj){"rtn this if not found"} #obj's can have different IDs but same contents**

.**delete\_if{|obj| obj.size < 4 } #delete obj based on property, i.e. obj.size < 4**

.**join (delim)**  **#into string, ["hello", "there"].join(' ')**

.**include? (element) #does the array contain element**

**.index(element) #index of where the element is stored**

**.insert(idx, obj, obj2…)** **#insert obj before element idx**

.**map { |el| el + 1 }**  **#rtn array w/ 1 added to each element**

Iteration: do something on each element of the array. It looks like this:

**my\_ar.each { |element| puts element }**

or

**my\_ar.each do |element| #side effects only.(puts element)**

**puts element #the array is returned not changed.**

**#but can assign in loop:**

**end #new\_ar << element + 5 within do**

**.each\_with\_index {|element, index | puts element + index }**

**#access to element and its index.**

.**any?( {|element| condition})**

.**all?( {|element| condition})**

**.none?( {|element| condition})#return appropriate boolean**

**.select {|num| num % 2 == 0}** **#keeps only even**

**.select do |number| … end** #alt format

**.delete\_if {|num| num % 2 == 0}** **#keeps only odd**

**.flatten** **#multi-dim array to linear array**

**.sort**  **#return an new sorted array**

**.sort!** **#modify the original array**

### Splat

the splat operator turns arrays into a parameter list, and a param list into an array.

**def** divide(numerator, denominator)

numerator / denominator

**end**

divide(\*[4,2]) **#array into params**

values = [5, 6, 7, 8]

**def** min\_max(\*values) **#params into array**

[values.min, values.max]

**end**

min, max = min\_max(3,5,2) **#min # => 2,max # => 5**

first, \*rest = [1, 2, 3] **#using splat to slurp**

first **# => 1**

rest **# => [2, 3]**

triples = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] **#array to params in block**

triples.each { |(first, second, third)| puts second } **#note multiple parameters**

triples.map { |(first, \*rest)| rest.join(' ') } **# => ["2 3", "5 6", "8 9"]**

### Strings

**str = "str variable"**

**'' and " " # single quote doesn't allow string interpolation inside**

**puts "this is a #{str}" #=> this is a string variable**

**puts 'this is a #{str}' #=> this is a #{str}**

**text = %q{This is a test #create multi line string**

**of multi-line text}**

**“The var is: #{variable}.” #print string and value**

**'Red' << ‘Ruby' #=> "RedRuby" String append (shovel operator)**

**'Red' + ‘Ruby' #=> "RedRuby" String append (+ operator)**

**.each\_char**

**.each\_line**

**.length**

**.index ’c’ #return the index of first ‘c’**

**.starts\_with?(string)**

**.end\_with?(string)**

**.split(’ ‘) #split into an array at each ‘ ‘**

**.to\_f #convert to float**

**.to\_i #convert to integer**

**.next #inc as int or string, stays a string**

**.sub('I', 'We')**

**.gsub('I', 'We')**

**.gsub(/[aeiou]/,'ff') #sub ‘ff’ for each & every vowel**

**.match(/ ./,4) #match “ .” at pos 4 or after**

**65.chr #returns the ascii char for a number**

**'c'.getbyte(0) #returns the letter for a 1 byte ascii**

Logical Operators

|| && !

### Hash: Key/Value pairs.

**kid\_ages = { "Jack" =>** 10**,"Jill" =>** 12**} #string key, rocket form**

**kid\_ages["Jack"] =** 11

**kid\_ages =** {**:Jack** => 10, **:Jill** => 12} **#symbol key**

**kid\_ages[√] =** 11

**kid\_ages =** {**Jack:** 10, **Jill:** 12} **#short cut form**

**kid\_ages[:Jack] =** 11

**a =** [**:punch**, 0] **#alternative defining of a Hash**

**b** = [**:kick**, 72]

**key\_value\_pairs** = [a,b];

**chuck\_norris** = Hash[key\_value\_pairs]

**kid\_ages = Hash.new("brown") #default value for missing keys**

**kid\_ages.keys #retrieving keys**

**.keys**

**.values**

**.clear**

**.delete(key) #deletes the k,v pair, returns v**

**.delete\_if { |k, v| v < 12 } #delete any for which block rtns true**

**.has\_value?(v)**

**.has\_key?(k)**

**.each { |k, v| puts "#{k} and #{v}" }**

**.delete\_if {**|k, v| k == 4 && v <=11**}**

**.select {**|k, v| k == 4 && v <=11**}**

hash = {:a => 1, :b => 2, :c => 3} **#hash values to variables**

a, b = hash.values\_at(:a, :b)

### Conditionals

**if** *condition*

*stmt*

**elsif** *condition*

*stmt*

**else**

*stmt*

**end**

**end**

stmt

**end**

**unless** can be used instead of **if**

*cond* **?** *stmt1* : *stmt2* **#ternary: if cond then do stmt1 else to stmt2**

*expr* **if** *condition* **#used for 1 line conditional**

*expr* **unless** *condition*

**case** my\_var

**when** *5*

*stmt*

**when** *(6..10)*

*stmt*

**else**

*stmt*

**end**

**case #without a variable, when can be full conditional**

**when** *x < 25 & y > 18*

*stmt*

**when** *condition*

*stmt*

**else**

*comp*

**end**

### Loops

**loop do**

**break [conditiona]**

**end**

**while** *condition*

**next if condition**  **#skip to the next loop iteration**

**end**

statement **while** *condition*

**until** *condition*

…

end

**loop** **do**

**break if** *condition*

**end**

5.**times** do |i|

**end**

5.**downto(1)** do |i|

**end**

**for element in** array do

**end**

**for element in** (a..b) do

**end**

**array.each** do |element|

**end**

**array.each\_with\_index** do |element, index|

**end**

**hash.each** {|key,value| puts key + value }

**string.each\_char** do |c|

**end**

(3..6).**each** do |el|

**end**

**for i in** (0..5) do

1.**upto(5)**1**)** do |number|

Sometimes frowned upon:

$i = 0

$num = 5

**begin**

puts("Inside the loop i = #$i" )

$i +=1

**end while** $i < $num

### Classes

**class** Rectangle

**attr\_accessor** :var1, :var2, :var3

**def** initialize(length, breadth)

**@**length = length

**@**breadth = breadth

**end**

**def** perimeter

2 \* (@length + @breadth)

**end**

**end**

Over-riding a Method

**class MyClass**

**def initialize**

**@code = ['A', 'B', 'C', 'D']**

**end**

**def [](i)**

**@code[i]**

**end**

**end**

**mine = MyClass.new**

**puts mine[3]**

### Blocks & Procs

A proc is just a block you give a name so you could call it from more than one place.

Create like this: **my\_p** = Proc.new {|a, b| a + b}

Call it like this: **my\_p**.call(a, b)

#Whenever you pass a Proc to a method you have to do it as you would a variable, within the method's( )s. #You pass block outside the ().

def calculation(a, b, my\_p) #define a method that takes a proc

my\_p.call(a, b)

**end**

addition = lambda {|a,b| a+b } #lambdas are like proc but also check number of args

puts calculation(5,4,addition)

With the addition of the & you can pass a raw block outside when you call the method:

**def** takes\_a\_block(a, b, &operation)

operation.call(a, b)

**end**

puts takes\_a\_block(7,4){|a,b| a + b}

#In this case to pass a lambda or proc you must have a & in the call to turn it back into a block

puts takes\_a\_block(5, 14, &my\_p)

def takes\_a\_block(a, b, &my\_p) #Using yeild you call the block

yield(a, b)

**en**

puts takes\_a\_block(1,4){|a,b| a + b}

or implicit:

def calculation(a, b) #don't use this, it sucks you can't see the block

yield(a, b)

**end**

puts calculation(-1,-4){|a,b| a + b}

—-

### Files

string = File.read("movie-times.txt")

array = File.readlines("movie-times.txt")

f = File.new("todos")

f.each {|line| puts "#{f.lineno}: #{line}" }

File.open("cool-things.txt", "w") do |f| #will create if file doesn’t exist

f.puts "Race cars"

f.puts “Lasers”

end

f.close

$stdin.each do |input| #line is basic unit

puts "I was given: #{ input }”

if \_\_FILE\_\_ == $PROGRAM\_NAME #if run directly from bash (not pry) do this stuff

### ARGV

$ ruby my\_cat\_counter.rb list\_of\_cats.txt

if !ARGV.length==0

puts ARGV[0]

### Exceptions

**class BadCommand < StandardError  
end** puts **"hi there"** loop **do  
 begin** *input* **= ask\_user  
 break  
 rescue BadCommand** *=>* e  
 puts e.message  
 puts **"Try again: "** *#puts e.backtrace.inspect* **end  
 end  
  
end  
  
def ask\_user** puts **"enter a command: "** command**=** gets.chomp  
 *raise* **BadCommand**.new **"Not a valid command" \**

**unless** [**"jump"**, **"roll-over"**, **"sit"**].include?(command)  
 command**end**

# Regular Expressions

Regular expressions match patterns in strings. For example, the \* character is often used as a "wild card" character that matches one or more of any character, so **\*apple** matches both **apple** and **crabapple.** if you use UNIX you'll know that

**ls \*.rb**

willlist all of the ruby files in the directory, those whose name ends in ".**rb**". Regexes can be useful to match legal forms of phone numbers or email addresses and not invalid ones, or to substitute a replacement string for a matched one (using **string.gsub**).

Regular expressions are virtually unreadable, mind-numbing, tedious but powerful. A regex command string is usually placed between slash characters: / and /

In Ruby regexes are used with the methods **string.match** and **string.gsub.**

Each code from the summary below can be used to match one or more characters in a string. To include a special character like '{' or '(' in a string (e.g. a phone number) use a backslash before the special character to "escape" it.

Encosing a regex pattern in (..) will capture the string it matches which can then be used in the replacement string of a gsub call. Each (…) group can be referenced by number, for example '\1' is the first such group. See the example below.

[rubular.com](http://rubular.com) is a website for testing regexes.

Examples:

/[aeiou]/ **match any single vowel**

/\d/ **match any single digit**

/\d{3} **match any string of three digits**

/\d{3}.? **match any 3 digits followed by zero or one other char**

\d{3}\)?.\d{3}.?\d{4} **match any three digits, a possible ')', then three digits,**

**and then 4 digits with possible separators**.

/\A[+-]?\d+\Z/ **match only integers**

Here's an example that removes parenthesis and other chars from a phone number. The phone digits are captured in (…) and used as '\1' . Easier to use Ruby's string.select, no?

string = **"(415) 297-3277"**

**puts string.gsub**(/\(?(\d{3})\)?.?(\d{3}).?(\d{4})\)?/, '\1''\2''\3')

#=> 4152973277

### Regular Expression Matchers

[abc] **A single character of: a, b, or c**

[^abc] **Any single character except: a, b, or c**

[a-z] **Any single character in the range a-z**

[a-zA-Z] **Any single character in the range a-z or A-Z**

^ **Start of line**

$ **End of line**

\A **Start of string**

\z **End of string**

. **Any single character**

\s **Any whitespace character**

\S **Any non-whitespace character**

\d **Any digit**

\D **Any non-digit**

\w **Any word character (letter, number, underscore)**

\W **Any non-word character**

\b **Any word boundary**

(...) **Capture everything enclosed, see groups in**

(a|b) **a or b**

a? **Zero or one of a**

a\* **Zero or more of a**

a+ **One or more of a**

a{3} **Exactly 3 of a**

a{3,} **3 or more of a**

a{3,6} **Between 3 and 6 of** a

\ **putting a \ before a special char like ) or } means treat is as just a char**

# Git and GitHub

### Where is my Phase-N Guide?

* + - <https://github.com/sf-bumblebees-2015> #make this a bookmark
    - search for "guide" if you don't immediately see your phase guide

### How do I: Fork a repository from DevBootCamp to my own github account?

* Fork will copy and break the connection to the parent repository. You can't git push to the parent.
* Go to [GitHub.com](http://GitHub.com) and login to your account.
* Go to the GitHub repository page you want.
* If you see the following graphic, it's a repository you can fork. Click it.

clone a repository from DevBootCamp to your own computer

It will ask you where to fork it, and you can click on your GitHub picture.

### How do I clone a DBC repository to my Mac?

Go to the repository you want (see above)

Look for this:

If you don't see this, you are not on a repository page. Click the little clipboard icon to copy that URL. Make sure it says "https" since this is a protected directory.

Go to your computer and navigate in terminal to a folder (e.g. ~/DevBootCamp) where you want the cloned directory to be placed. I had folders for each phase and each week inside the folder DevBootCamp. Then from the terminal type git clone and paste in the URL you just copied above:

git clone <https://github.com/sf-bumblebees-2015/phase-1-guide.git>

### OK, I've cloned to my machine. How do I make a branch?

Make sure you in the repository's main directory. If you are not sure do an

**ls -la**

and you should see the **.git** directory.

drwxr-xr-x@ 14 mikefarr staff 476B Jun 4 22:08 **.git**

Now, if you are on your own computer:

**git checkout** -b *mybranchname* **master**

If you are at DBC:

**weare** *git-hub-name*

or if pairing:

**weare** *git-hub-name,other-git-hub-name*

**pair-branch**

This will do the checkout -b for you. Do this whenever you clone a repository and need to make your own branch. When you leave be sure to do a

**weare out**

to log out of the DBC computer.

### OK, how do I push my own branch to DBC?

Make sure you are not on the master branch. Make sure you have committed all files. A common mistake is that you forgot to add modified or new files. Do a:

git status

git commit -m "commit message:

git push origin *mybranchname*

If you are at DBC, this branch should start with "pair-" You can type

git push origin pair<tab>

Git should fill in the name of the branch. If it doesn't auto-complete you are likely not in the correct branch.

### How do I push my local git repository up to my own GitHub account?

OK, so you've used git init or git clone to create a repository locally. Now you want to push it up to your GitHub account. I did this all the time to save a copy of all my work. First, go to your github account in the browser. You bookmarked it, right?

Click on the Repositories tab. Now create a new empty repository so you have somewhere to push to: click on:

or



Add an MIT license to it.

Copy the URL using the clipboard icon again:

Go back to the terminal window and create a

readme if you don't have one yet:

echo "# Shortest description">> README.md

Make sure you have done a git status. Commit anything you need to.

git status

git commit -m "first commit"

Now type the following to add a new remote name for this repository, pasting in the URL you copied from GitHub.

git remote add upstream <https://github.com/username/repo_name.git>

git push -u upstream master

Remember that **both** parameters to a git push, the "origin master", "origin branch name" or "upstream master" refer to the remote. It's push current branch to *remote\_repo remote\_branch*

# RSpec Cheat Sheet

require\_relative 'grocery\_list'

describe GroceryList do

let(:groceries) { GroceryList.new } **#**let **is reset at every** it **clause**

it **"has a list array"** do

expect(groceries.list).to be\_a(Array)

end

context '#add' do

item = “lettuce” **#item defined inside context only**

it "adds an item to the list array" do

groceries.add(item)

expect(groceries.list[0]).to eq("Lettuce")

end

end

it "removes an item from the list array" do

groceries.add("Yogurt")

groceries.add("Gogurt")

groceries.remove("Yogurt")

expect(groceries.list[0]).to eq("Gogurt")

end

end

**#Checking inheritence example. See matchers below:**

describe "Dog" do

describe "inheritance" do

it "inherits from Pet" do

expect(Dog < Pet).to be true

end

end

## Built-in Rspec matchers

### Equivalence

expect(actual).to eq(expected) # passes if actual == expected

expect(actual).to eql(expected) # passes if actual.eql?(expected)

expect(actual).not\_to eql(not\_expected) # passes if not(actual.eql?(expected))

### Identity

expect(actual).to be(expected) # passes if actual.equal?(expected)

expect(actual).to equal(expected) # passes if actual.equal?(expected)

Comparisons

expect(actual).to be > expected

expect(actual).to be >= expected

expect(actual).to be <= expected

expect(actual).to be < expected

expect(actual).to be\_within(delta).of(expected)

### Regular expressions

expect(actual).to match(/expression/)

### Types/classes

expect(actual).to be\_an\_instance\_of(expected) # passes if actual.class == expected

expect(actual).to be\_a(expected) # passes if actual.kind\_of?(expected)

expect(actual).to be\_an(expected) # an alias for be\_a

expect(actual).to be\_a\_kind\_of(expected) # another alias

### Truthiness

expect(actual).to be\_truthy # passes if actual is truthy (not nil or false)

expect(actual).to be true # passes if actual == true

expect(actual).to be\_falsy # passes if actual is falsy (nil or false)

expect(actual).to be false # passes if actual == false

expect(actual).to be\_nil # passes if actual is nil

expect(actual).to\_not be\_nil # passes if actual is not nil

### Expecting errors

expect { ... }.to raise\_error

expect { ... }.to raise\_error(ErrorClass)

expect { ... }.to raise\_error("message")

expect { ... }.to raise\_error(ErrorClass, "message")

### Expecting throws

expect { ... }.to throw\_symbol

expect { ... }.to throw\_symbol(:symbol)

expect { ... }.to throw\_symbol(:symbol, 'value')

### Yielding

expect { |b| 5.tap(&b) }.to yield\_control # passes regardless of yielded args

expect { |b| yield\_if\_true(true, &b) }.to yield\_with\_no\_args # passes only if no args are yielded

expect { |b| 5.tap(&b) }.to yield\_with\_args(5)

expect { |b| 5.tap(&b) }.to yield\_with\_args(Fixnum)

expect { |b| "a string".tap(&b) }.to yield\_with\_args(/str/)

expect { |b| [1, 2, 3].each(&b) }.to yield\_successive\_args(1, 2, 3)

expect { |b| { :a => 1, :b => 2 }.each(&b) }.to yield\_successive\_args([:a, 1], [:b, 2])

### Ranges

expect(1..10).to cover(3)

### Collection membership

expect(actual).to include(expected)

expect(actual).to start\_with(expected)

expect(actual).to end\_with(expected)

expect(actual).to contain\_exactly(individual, items)

# ActiveRecord -

1. Active Record is a collection of Ruby classes that wraps an SQL database. Using the AR classes is easier than using SQL.
2. Using Active Record like this:
3. Is there a default AR directory structure setup? I.e. new AR app?
   1. Use SQL designer and create a schema
   2. Create some Ruby AR "migration" classes that define/create the database
   3. Create Ruby "Base" classes that represent the entities in your database (the Model)
   4. Use objects from those base classes instead of the database
4. AR applications have a rather consistent directory structure:

<http://guides.rubyonrails.org/active_record_querying.html>

Dog.all

Dog.where(age: 1)

The SQL executed was SELECT "dogs".\* FROM "dogs" WHERE "dogs"."age" = 1.

Dog.where("age = ? and name like ?", 1, '%Te%')

Dog.order(age: :desc)

Dog.limit(2)

Dog.count

Dog.pluck(:name, :age)

Dog.first

Dog.find(1)

Dog.order(name: :asc).where(age: 1).limit(1)

Dog.create([{name: “Spot”, age: 1}, {name: “Cosmo”}])

Dog.find\_or\_initialize\_by(name: “Tenley”, license: “OH-9384764”)

Dog.find\_or\_create\_by() # will try to save a new obj

#Once you have an instance, AR creates getter and setter methods for all attributes

henley = Dog.find\_by(name: “Henley”)

henley

henley

ActiveRecord::Base.connection.tables #the table in the DB

class Rating < ActiveRecord::Base

belongs\_to :dog

end

gives us rating.dog and rating.dog =

#build\_dog, #create\_dog, and #create\_dog!

In ratings:

belongs\_to :dog, { :class\_name => "Dog", :foreign\_key => :dog\_id }

belongs\_to :judge, { :class\_name => "People", :foreign\_key => :judge\_id }