# RUBY – RUBY – RUBY – RUBY – RUBY

Everything in Ruby is an OBJECT

Command line

Irb to test lines in the terminal (just like JS console)

ruby file.rd to run the file

Qui / exit / ctlr+d

Ruby –v for current version

Pry

Can’t execute script inside pry though - quit

## Logics and Basics

### Data types

Sth.class -> tell me what it is

Convert data types: '5'.to\_i => string to integer

(1..3).to\_a => [1,2,3]

[1,2,3].map { |i| i.to\_s } => [‘1’,’2’,’3’]

!! ‘hello’.to\_i -> 0 !

'4.56fg'.to\_f -> 4.56

is\_a? to compare data types (Return true / false)

b.is\_a? (String) / same as .kind\_of? String

Equality: == // !=

Boolean and logical operators

Similar to JS

Nil && true -> nil

5 || 2 -> 5 (stops at first true) ; 5 && 2 -> 2 (checks all conditions so runs the last one too)

### Number

2\*\*3 : 2 power of 3 ; % rest in Euclidian division

7/2 -> 2 ! Dividing a fixed number by a fixed number returns a fixed number

way around simply write one float

7.odd? => true

Number.round(n)

Round UP : .ceil / round dowm .floor

(self.to\_f).round

Also found very weird (self.to\_f +0.5 ).to\_i

=> ensure self is a float, and adding 0.5 will ensure the corresponding integer is right

### Strings

Print : display (like console.log)

**Puts** : same & adds a new line underneath

Get input from user - always returns a string!

gets (get string)

gets.chomp to avoid recording the “enter key” at the end, cause it adds a line

.downcase / .upcase / .capitalize (first letter only) / .swapcase (swap for every letter)

multiply strings: ‘gui’\*3 -> guiguigui

text.include? 'string’

.start\_with? / .end\_with? / String.index(‘letter’)

‘string’.index(‘t’) -> 1

‘string’.chars[1] -> ‘t’  *!! inside [ ]*

Slice strings (and arrays)

String[1,3] : from index, return array with 3 elements

**Change text**

.split(‘where’)

Concat: +, or << (append to existing word), or word1.concat(word2)

“blab li blu”.sub(‘bla’, ‘boum’) // or gsub for all occurrences of bla

delete all capital letters => string.delete "/[A-Z]/"

Partition to separate text

Searches sep or pattern (regexp) in the string and returns the part before it, the match, and the part after it. If it is not found, returns two empty strings and str.

"hello".partition("l") #=> ["he", "l", "lo"]

"hello".partition("x") #=> ["hello", "", ""]

"hello".partition(/.l/) #=> ["h", "el", "lo"]

**Regular expressions: match text by its patter**

“replace All text in Capitals”.gsub(/[A-Z]/, '0')

Regular expressions are put between /…/

[A-Z] is pattern for capital letters

“Find character next to a whitespace”.match(/ ./, index to start search) -> I

.match + regEx: is this word following the pattern?

array.keep\_if { |i| /[aeiou]/.match(i.chars[0]) }

**Sring interpolation – must have double-quotes**

**puts "the number you want is #{number}. How cool"**

Below are some of the more common escape sequences that can appear inside of double quotes.

* \" – double quote
* \\ – single backslash
* \a – [bell/alert](http://en.wikipedia.org/wiki/bell_character" \o "w:bell character" \t "_blank)
* \b – [backspace](http://en.wikipedia.org/wiki/backspace" \o "w:backspace" \t "_blank)
* \r – [carriage return](http://en.wikipedia.org/wiki/carriage_return" \o "w:carriage return" \t "_blank)
* \n – [newline](http://en.wikipedia.org/wiki/newline" \o "w:newline" \t "_blank)
* \s – [space](http://en.wikipedia.org/wiki/space_character" \o "w:space character" \t "_blank)
* \t – [tab](http://en.wikipedia.org/wiki/tab_key" \o "w:tab key" \t "_blank)

**puts** "Hello\t\tworld"

**puts** "Hello\b\bGoodbye world"

**puts** "Hello\rStart over world"

**puts** "1. Hello\n2. World"

The result:

$ **double-quotes.rb**

Hello world

HeGoodbye world

Start over world

1. Hello 2. World

Comment: #

**Dates**

Convert to 31/10/2013 format

date.strftime "%d/%m/%Y"

**Range**

(1..20).max = (1..20).last = 20

(1…20).max = 19 but last = 20

**Access a file: IO (input / ouput)**

# count the number of words in a file

IO.read(file\_path).split.length

### LOOPS

Ctrl+c to exit

**If then else**

print 'Enter a number'

number = gets.to\_i

if number > 0

puts 'you entered a positive number'

elsif number < 0

puts 'you entered a negative number'

else

puts 'you entered zero'

end

# don’t forget the end statement!

**Unless**, just like if !xx

user\_registered = false

puts 'Please sign up to continue' unless user\_registered

Or 1 line:

if true then puts 'Hello!' else puts 'Bye' end

**Case statement**

puts 'Exit the program? (yes or no): '

answer = gets.chomp.downcase

case answer

when 'yes'

puts 'Goodbye!'

when 'no'

puts 'Ok. Continue!'

else

puts 'Not sure what you mean by that'

end

**While / until**

number =1

while number < 11

puts number

number = number + 1

end

//

until number >10

puts number

number = number + 1

end

**For loops are NOT recommended**

for i in 1..10 !!! .. up to and including last one /// … up to and not including last one

break if i > 5

puts i

end

**100.times do / up to / down to**

**end**

10.times do |k| puts "Number #{k+1}"   
# times will start at 0, so on the 10th iteration, k is equal to 9 end

1.upto(10){ |k| puts "Number #{k}"}

(1..100).each do |k|

puts "#{k}. This is Ruby preferred way of doing loops, when possible"

end

(1..100).each{ |k| puts "#{k}. Curly braces make it even shorter"}

## Functions / Methods

String.methods => all methods linked to strings

.sort.inspect to list them in a condensed look

def say\_hello

puts 'Hello World'

end

*# no need for () to call it*

def add(number1, number2)

“ the result is #{number1 + number2}”

end

***## by defaut RB will “return” the last line of the function***

Unlike JS, methods MUST be declared before being called

***!! Scoping in Ruby !!***

***All variables must be declared INSIDE the method. It won’t understand variables declared outside***

🡪 Pass all variables you’ll need as arguments of the function

x=0

def doStuff(x)

foo = x

puts foo

end

doStuff(x)

Use default value if no argument are passed when calling methods

def hello\_students(students=’james’)

puts students

end

🡪 hello\_student(‘lauren’) returns ‘lauren’

🡪 hello\_student() returns ‘james’

### Blocks

10.times { puts ‘heyyy’ }

10.times do

|n|

*#more than one line*

end

Shorthand for a block

array.reject(&:!)

same as array.reject { |e| e==alse } since e==false same as !e same as !

Other example: array.map {|i| i.reverse } = array.map(&:reverse)

array.map(&:reverse)

**You can always add a block to a method**

* **use YIELD to declare the elements on which to pass the block on**
* call the function using the block

def **doge**(word1, word2)

phrases = [**'**wow**'**, **"**much #{word1}**"**, **"**so #{word2}**"** ]

for phrase in phrases

yield phrase

end

end

doge(**'**ruby**'**, **'**syntax**'**) {|str| puts str.upcase }

* WOW
* MUCH RUBY
* SO SYNTAX

You get an error if you don’t’ pass a block

block\_given? to check if a block was passed on

way around:

for phrase in phrases

yield phrase if block\_given?

end

### Enumerables

Each is a method on the enumerable module

**%w(**jeremy lauren mathilda lexie**)**.each { |name| print\_name(name) }

* jeremy lauren mathilda lexie

On a hash by default each uses key-value pair

hash.each {|key, value| puts **"**#{key} score was #{value}**"** }

On hash can use each\_with\_index to play with the index:

**%w(**jeremy mathilda lauren**)**.each\_with\_index do |name, index|

[28] pry(main)\* **#code**

[28] pry(main)\* end

See more in arrays/hashes

.detect => finds 1st item to match condition

.select / .find\_all

.reject /.drop

.keep\_if / .delete\_if

**.reduce / .inject takes |sum, n| as arguments**

[**1**, **2**, **3**, **4**, **5**, **6**, **8**, **9**, **10**].reduce { |sum,n| sum + n }

* sum=1, n=2 ; next sum = 3, n=3; etc

Short-form: (**1**..**10**).inject(:+)

Can also provide an initial value, put it under bracket: .inject(set init value)

EG: counting the occurrence of each word in a sentence

* it will be stored in a new hash, which has default value zero
* first round: {}
* second round grabs first word so hash = {‘word’ => 1 }

long\_sentence.split(**'** **'**).inject(**Hash**.new(**0**)) do |hash, word|

hash[word.downcase] += **1**

hash

end

### Splat Operator – methods with variable list of parameters

def introduction(age, gender, \*names)

"Meet #{names.join(' ')}, who's #{age} and #{gender}"

end

**def** add(\*numbers)

numbers.inject(0) { |sum, number| sum + number }

**end**

**def** add\_with\_message(message, \*numbers)

"#{message} : #{add(\*numbers)}"

**end**

puts add\_with\_message("The Sum is", 1, 2, 3)

### Passing Blocks {} as parameters – options

def release\_if (&block)

output = @bikes.select &block // gonna pass our block in the .select method

output.each { |bike| @bikes.delete(bike) }

end

docking\_station.release\_if ( { :broken? } )

* broken? is a method on class Bike -> bike.broken? returns true or false
* if I run release\_if { :broken? }
  + bike.select &block becomes bike.select(&:broken?)
* if I run release\_if{ : \*\*\*\*\* need help for syntax to do releas\_if bike.broken? == false

***& says ‘hey I’m gonna pass an argument to you’***

def add(a\_number, another\_number, options = {})

sum = a\_number + another\_number

sum = sum.abs if options[:absolute]

sum = sum.round(options[:precision]) if options[:round]

sum

end

puts add(1.0134, -5.568) -> -4.5546

puts add(1.0134, -5.568, absolute: true) -> 4.5546

puts add(1.0134, -5.568, absolute: true, round: true, precision: 2) -> 4.55

You can't use both splatted arguments and last-parameter-is-a-hash at the same time through Ruby, so you'll have to work on the arguments inside of calculate. There's no neat way to do this - you have to check if the last argument to calculate is a Hash, then remove it from the list before calling add or subtract.

def add(\*numbers)

numbers.inject(0) { |sum, number| sum + number }

end

def subtract(\*numbers)

current\_result = numbers.shift

numbers.inject(current\_result) { |current\_result, number| current\_result - number }

end

def calculate(\*arguments)

# if the last argument is a Hash, extract it

# otherwise create an empty Hash

options = arguments[-1].is\_a?(Hash) ? arguments.pop : {}

options[:add] = true if options.empty?

return add(\*arguments) if options[:add]

return subtract(\*arguments) if options[:subtract]

end

## Arrays & Hashes

Array to a Hash

Hash[\*array] (same as hash = Hash.new

# some say it can crash on loads of data so use => Hash[array.each\_slice(2).to\_a]

If array = [**'**a**'**, **'**b**'**, **'**c**'**, **'**d**'**]

array.each\_slice(**2**).to\_a

* [[**"**a**"**, **"**b**"**], [**"**c**"**, **"**d**"**]]

### Arrays

<http://ruby-doc.org/core-2.1.5/Array.html>

arr.methods

**Create array**

my\_array = [1,2,3,4,5,6]

.first / .last /

go [-1] to get the last one

(1..10).to\_a => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

%w(monday tuesday wednesday) => ["monday", "tuesday", "wednesday"]

Array << item // array.push(item) #at the end // Array.unshift(item) #at the beginning

arr.unshift(arr.pop) -> take the last one and put in first position

arr << 9 << 7 << 6 : insert multiple

arr.insert(3, 'orange', 'pear', 'grapefruit') : at index 3, add all the following

**Delete**

.pop : removes last and returns it (does not alter the original array)

.shift : removes first and returns it (does not alter the original array)

arr.delete(2) : delete element 2

arr.delete\_at(2) : delete on index 2

[1,2,3,4,5,6,7,8,9].delete\_if{ |i| i%2 == 0 } : delete IF following condition is met

**Look for elements**

Array.count(item) : how many times in item present

array.include? 'item'

array.emptty? => true or false

Index

a = [ "a", "b", "c" ]

a.index("b") #=> 1

Intersection with &

Set Intersection — Returns a new array containing elements common to the two arrays, excluding any duplicates. The order is preserved from the original array

**Finding elements, according to criteria (length, being odd, etc**

.max, .min

item in array with max number of words: array.max\_by(&:length) // or .min

.sort /.sort!

sort by last element array.sort\_by {|e| e[-1]}

**Operations on arrays**

.sample : returns random element

.suffle : shuffles everything

.reverse : does not alter the original array, so need to store it into another variable

.reverse! : WILL alter the original array

.uniq : remove duplicates ( and .uniq!)

.flatten array if there is an array inside / .flatten!

.compact to remove the nils / .compact!

**Iterating thourgh arrays**

Array.each { |n| do something on n }

.each\_with\_index { |item, index| }

.reverse\_each : iterate in reverse order

.map will return a new array

.map! will modify the array

.map.with\_index

Non-destructive selection : select if matches criteria; reject removes those who don’t match

arr = [1, 2, 3, 4, 5, 6]

arr.select { |a| a > 3 } #=> [4, 5, 6] // same as .find\_all

arr.reject { |a| a < 3 } #=> [3, 4, 5, 6]

arr.drop\_while { |a| a < 4 } #=> [4, 5, 6]

arr #=> [1, 2, 3, 4, 5, 6]

.drop(n) : Drops first n elements from ary and returns the resulting array

.take Returns first n elements from the array.

Take\_while

a = [1, 2, 3, 4, 5, 0]

a.take\_while { |i| i < 3 } #=> [1, 2]

.delete\_if and .keep\_if

[1,2,3,4,5,6,7,8,9].delete\_if{ |i| i.odd? } : delete IF following condition is met

**.reduce / .inject takes |sum, n| as arguments**

[**1**, **2**, **3**, **4**, **5**, **6**, **8**, **9**, **10**].reduce { |sum,n| sum + n }

* sum=1, n=2 ; next sum = 3, n=3; etc

Short-form: (**1**..**10**).inject(:+)

Can also provide an initial value, put it under bracket: .inject(set init value)

EG: counting the occurrence of each word in a sentence

word.split(**'** **'**).inject(**Hash**.new(**0**)) do |hash, word|

[32] pry(main)\* hash[word.downcase] += **1**

[32] pry(main)\* hash

[32] pry(main)\* end

Combinations: yields all combinations of length n of elements from the array and then returns the array itself.

a = [1, 2, 3, 4]

a.combination(1).to\_a #=> [[1],[2],[3],[4]]

a.combination(2).to\_a #=> [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]

### Hashes

<http://ruby-doc.org/core-2.1.5/Hash.html>

**Create**

***Syntax! england: ‘London’ for key as symbol***

***‘england’ => ‘London’ for key as string***

Capital = {england: **'**London**'**, france: **'**Paris**'**}

Chuck = { “punch” => 99, “kick”=> 98, etc}

chuck = Hash[:punch, 99, :kick, 98, :stops\_bullets\_with\_hands, true]

Access: capital[:england] => ‘London’

my\_friends[:sophie] = 'likes cinema'

Add/modify: capital[:italy] = **"**Rome**"**

Or capital = **Hash**.new(**'**This country is not listed**'**)

* Default value if you access a key that doesn’t exist – will return msg instead of neg
* Eg capital[:usa] => ‘this country is not listed’

**Frequent confusion to access HASH and CLASSES**

**Hash with hash[thing] // class with class.thing**

*Add a sub-hash to all keys in the hash*

students.keys.each {|i| students[i][:test\_subhash] = [**'**test**'**, **'**array**'**]}

*:Symbol vs “String”*

Symbol cannot be changed. Use it when you want to prevent the element to be changed (it would return an error)

Symbol is stored in memory while string created each time -> symbol more memory efficient

List all symbols currently stored in memory: Symbol.all\_symbols

#### Playing with key-value pairs

options.fetch(:capacity, DEFAULT\_CAPACITY)

Hash.fetch method retrieves the value for the given key ( options[:capacity] in this case) and if the key is not found, it returns the second argument ( DEFAULT\_CAPACITY ). So if the capacity is passed, it's used, otherwise the default one will be assigned.

* works same as options[:capacity] || DEFAULT\_CAPACITY
* use CAP LETTERS to signify constant variable

h = { "a" => 100, "b" => 200, "c" => 300, "d" => 300 }

h.key(200) => "b"

or .assoc

h.assoc("letters") #=> ["letters", ["a", "b", "c"]]

h.assoc("foo") #=> nil

.invert => Returns a new hash created by using *hsh*’s values as keys, and the keys as values

Flatten hash to put all in array – needs (n) to tell how recursive you want it (flatten array is recursive, not hash)

a = {1=> "one", 2 => [2,"two"], 3 => "three"}

a.flatten # => [1, "one", 2, [2, "two"], 3, "three"]

a.flatten(2) # => [1, "one", 2, 2, "two", 3, "three"]

**Delete by accessing the key**

{:england=>"Liverpool", :france=>"Paris", :australia=>"Canberra"}

capital.delete(:england)

{}.each do |key, value|

xxx

end

{}.keys / {}.values

## Object Oriented Programming

file:/Users/Guiton/Desktop/WDI\_13/classwork/week\_4\_ruby/

* obj\_orient\_prog/person.rb has lesson and explanations
* /oop\_rental\_appINCLASS has exo and homework correction

**Don’t forget to list the attribute accessors or nothing will work!!**

Use “options” to simplify input of long arguments

class Building

def initialize(options = {})

# options is a hash which will hold the attributes - no need to worry about the order

@floor = options[:floor]

@address = options[:address]

@apartments = options[:apartments]

@age = options[:age]

@concierge = options[:concierge]

end

end

If using DEFAULT\_CAPACITY, put it right under the class name

class DockingStation

DEFAULT\_CAPACITY = 10

def initialize(options={})

@capacity = options.fetch(:capacity, DEFAULT\_CAPACITY)

@bikes = []

end

To override the msg coming out when typing “puts” on an object

Add method to the Class, then run it on an instance

def to\_s

"My name is #{@name} and I study #{@course}"

end

=> dave = Student.new(‘Dave’, ‘wdi’)

=> puts dave => my name is Dave and I study wdi

**List ALL objects of a class**

def self.all

ObjectSpace.each\_object(self.class).to\_a

end

# Debugging and Test Driven Development

## Debugging

### Pry for Ruby

Require ‘pry’ at beginning of code

Potentially require ‘pry-byebug’

If using a main.rb to put require ‘fry’

* require\_relative ‘other files.rb’

Put binding.pry on a line to act like a debugger

Need code after binding.pry, just add nil if end of page

Ls to list all variables I have access to

Can look at what they are, play with them, etc

‘step’ to the next point

## Test Driven Development

# RSPEC

Better: :/Users/Guiton/Desktop/WDI\_13/homework/week\_4\_ruby/boris\_bike

Test files must be named xxx\_spec.rb

Good to write code in lib folder, and test in spec folder

Ex: write code on **greeter**.rb

Create **bike**\_rspec.rb

require\_relative ‘**bike’** // path relative to where the file is

I will focus on a CLASS called **Bike**:

describe ‘**Bike’** do

let(:bike) {Bike.new} => whenever I call ‘bike’ it will refer to this bike.new

it 'should not be broken when created' do

expect(bike.broken?).to be false

end

end

if using xit (Xit), the test will not run and will show as pending

Useful to create a spec\_helper.rb file

* put require\_relative ‘spec\_helper’ at top of each test file
* then in spec\_helper write all the files required for every test file
  + require ‘pry’, require\_relative ‘bike’, require\_relative ‘van’, etc

If all tests are on the same directory (spec) => run rspec

run full test on one file => spec greeter\_spec.rb

or run test on line 12 => spec greeter\_spec.rb:12

\*\*\*\*\* Doc from the Website \*\*\*\*\*

## Install

If you want to use rspec-expectations with rspec, just install the rspec gem and RubyGems will also install rspec-expectations for you (along with rspec-core and rspec-mocks):

gem install rspec

Want to run against the master branch? You'll need to include the dependent RSpec repos as well. Add the following to your Gemfile:

%w[rspec-core rspec-expectations rspec-mocks rspec-support].each do |lib| gem lib, :git => "git://github.com/rspec/#{lib}.git", :branch => 'master' end

If you want to use rspec-expectations with another tool, like Test::Unit, Minitest, or Cucumber, you can install it directly:

gem install rspec-expectations

## Basic usage

Here's an example using rspec-core:

RSpec.describe Order do it "sums the prices of the items in its line items" do order = Order.new order.add\_entry(LineItem.new(:item => Item.new( :price => Money.new(1.11, :USD) ))) order.add\_entry(LineItem.new(:item => Item.new( :price => Money.new(2.22, :USD), :quantity => 2 ))) expect(order.total).to eq(Money.new(5.55, :USD)) end end

The describe and it methods come from rspec-core. The Order, LineItem, Item and Money classes would be from *your* code. The last line of the example expresses an expected outcome. If order.total == Money.new(5.55, :USD), then the example passes. If not, it fails with a message like:

expected: #<Money @value=5.55 @currency=:USD> got: #<Money @value=1.11 @currency=:USD>

## Built-in 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))

Note: The new expect syntax no longer supports the == matcher.

### 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/)

Note: The new expect syntax no longer supports the =~ matcher.

### Types/classes

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

### Predicate matchers

expect(actual).to be\_xxx # passes if actual.xxx? expect(actual).to have\_xxx(:arg) # passes if actual.has\_xxx?(:arg)

### Ranges (Ruby >= 1.9 only)

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) # ...which is the same as: expect(actual).to match\_array(expected\_array)

## Examples

expect([1, 2, 3]).to include(1) expect([1, 2, 3]).to include(1, 2) expect([1, 2, 3]).to start\_with(1) expect([1, 2, 3]).to start\_with(1, 2) expect([1, 2, 3]).to end\_with(3) expect([1, 2, 3]).to end\_with(2, 3) expect({:a => 'b'}).to include(:a => 'b') expect("this string").to include("is str") expect("this string").to start\_with("this") expect("this string").to end\_with("ring") expect([1, 2, 3]).to contain\_exactly(2, 3, 1) expect([1, 2, 3]).to match\_array([3, 2, 1])

## should syntax

In addition to the expect syntax, rspec-expectations continues to support the should syntax:

actual.should eq expected actual.should be > 3 [1, 2, 3].should\_not include 4

See [detailed information on the should syntax and its usage.](https://github.com/rspec/rspec-expectations/blob/master/Should.md)

## Compound Matcher Expressions

You can also create compound matcher expressions using and or or:

expect(alphabet).to start\_with("a").and end\_with("z") expect(stoplight.color).to eq("red").or eq("green").or eq("yellow")

### Composing Matchers

Many of the built-in matchers are designed to take matchers as arguments, to allow you to flexibly specify only the essential aspects of an object or data structure. In addition, all of the built-in matchers have one or more aliases that provide better phrasing for when they are used as arguments to another matcher.

Examples

expect { k += 1.05 }.to change { k }.by( a\_value\_within(0.1).of(1.0) ) expect { s = "barn" }.to change { s } .from( a\_string\_matching(/foo/) ) .to( a\_string\_matching(/bar/) ) expect(["barn", 2.45]).to contain\_exactly( a\_value\_within(0.1).of(2.5), a\_string\_starting\_with("bar") ) expect(["barn", "food", 2.45]).to end\_with( a\_string\_matching("foo"), a\_value > 2 ) expect(["barn", 2.45]).to include( a\_string\_starting\_with("bar") ) expect(:a => "food", :b => "good").to include(:a => a\_string\_matching(/foo/)) hash = { :a => { :b => ["foo", 5], :c => { :d => 2.05 } } } expect(hash).to match( :a => { :b => a\_collection\_containing\_exactly( a\_string\_starting\_with("f"), an\_instance\_of(Fixnum) ), :c => { :d => (a\_value < 3) } } ) expect { |probe| [1, 2, 3].each(&probe) }.to yield\_successive\_args( a\_value < 2, 2, a\_value > 2 )

Usage outside rspec-core

You always need to load rspec/expectations even if you only want to use one part of the library:

require 'rspec/expectations'

Then simply include RSpec::Matchers in any class:

class MyClass include RSpec::Matchers def do\_something(arg) expect(arg).to be > 0 # do other stuff end end

Also see

* <http://github.com/rspec/rspec>
* <http://github.com/rspec/rspec-core>
* <http://github.com/rspec/rspec-mocks>
* <http://github.com/rspec/rspec-collection_matchers>

# WEB DEVELOPMENT – WEB DEVELOPMENT

Everything we did in Javascript was on a public folder. The .js file is accessible, and the data passed to the file can be hacked

What we do here is all on the main server; it is secure

## Gems

‘bundle init in’ a folder, wll create a gem file where you specify the gems you need to use

Running bundle will go through the gemfile and install any gem missing

## HTTP Language

***GREAT EXO w5 day 3: todo\_app : link sql, Sinatra, http methods***

**Homework w5 d3 videos\_tube**

Address

[www.domainname.com/?gui.perr](http://www.domainname.com/?gui.perr)

the ? query string is a great way to send key-value pairs

### HTTP verbs

# GET ALL items on the main page (main = /items): GET/items-index: we do get on items, and render an index

# GET new item: GET/items/new

# POST a new item: POST/items

# SHOW an item: GET/items/:id

# EDIT the item (send info about the edition): GET/items/:id/edit

# UPDATE the item: POST/items/:id (because patch/put aren't fully supported)

# DELETE an item: DELETE/items/:id

| **URL** | **HTTP Verb** | **Action** |
| --- | --- | --- |
| /photos/ | GET | index |
| /photos/new | GET | new |
| /photos | POST | create |
| /photos/:id | GET | show |
| /photos/:id/edit | GET | edit |
| /photos/:id | PATCH/PUT | update |
| /photos/:id | DELETE | destroy |

### HTTP ERRORS

200 – 299 : all good

300 - 399 : all good, but redirected somewhere

400 – 499 : user made mistake

404 : wrong url, 402 : unauthorized (Eg wrong password), 420:

500 – 599 : website fault

<http://en.wikipedia.org/wiki/List_of_HTTP_status_codes#3xx_Redirection>

### Params

**Params is the info passed around in the URL, when doing GET request**

**Within a response-request cycle, Params is the info I give, so that the response can work**

* **it returns a HASH of PARAMS: key is 'name', value is 'what we submit'**
* **params[:first] = whatever text I submitted**

### Template RESTful routes to create website

***Taken from exo week5 day 3: todo\_app : link sql, Sinatra, http methods***

***Also see* Homework w5 d3 videos\_tube**

*# My homepage will be localhost/items*

get '/' do

redirect to ('/items')

end

*# LIST ALL items we created*

get '/items' do

sql = "SELECT \* FROM items"

@items = run\_sql(sql)

erb :index

end

*# Send me to the page where a form enables to post a new item*

get '/items/new' do

erb :new

end

*# Post the new item on the list of all items*

post '/items' do

sql = "insert into items (item, details) values ('#{params['item']}', '#{params['details']}')"

run\_sql(sql)

redirect to ('/items')

end

*# SHOW a specfic item*

get '/items/:id' do

sql = "select \* from items where id = #{params['id']}"

# result is a PG object (~array) with 1 object inside: {'id'=1, 'item' => 'milk', 'details' => nil}

@item = run\_sql(sql).first

erb :show

end

**# Edit a specific item, => Go to page where Form enables to post/put/patch new details**

get '/items/:id/edit' do

sql = "select \* from items where id = #{params['id']}"

# result is a PG object (~array) with 1 object inside: {'id'=1, 'item' => 'milk', 'details' => nil}

@item = run\_sql(sql).first

erb :edit

end

*# Post the value to the existing item*

post '/items/:id' do

sql = "UPDATE items SET item = '#{params['item']}', details = '#{params['details']}' WHERE id = #{params['id']}"

run\_sql(sql)

redirect to ("/items/#{params['id']}")

end

*# Delete an item*

delete '/items/:id/delete' do

sql = "DELETE from items where id = #{params['id']}"

run\_sql(sql)

redirect to ('/items')

end

*See forms below for specific forms to create/edit info, and weird one to delete*

## Embedded Ruby for Web Dev – using Sinatra

See week\_5 day\_2,

***GREAT EXO week5 day 3: todo\_app : link sql, Sinatra, http methods***

**Homework w5 d3 videos\_tube**

### Basic concept

On the Ruby file

Require ‘pry’

require ‘sinatra’

requir 'sinatra/reloader' if development?

gem Sinatra-contrib provides us with a reloader

**Syntax is :**

**# *DO\_HTTP\_VERB*  *‘ON\_THIS\_PATH’* do**

**# # your code here**

**# end**

get '/' do # when URL go to our home/root page : localhost:4567

puts "you are on the homepage"

"Hellofgfdghg gth"

end

get '/about' do # URL localhost:4567/about

"This is all Meeeee"

end

get '/friends/:name' do # dynamic part of URL: name can be Gui, John, etc

binding.pry

"Your friend is called: #{params[:name]}"

end

whereami => when in pry, will tell you where you are

params is a hash:

{**"**splat**"**=>[], **"**captures**"**=>[**"**gui**"**], **"**name**"**=>**"**gui**"**}

Remember…

**Params is the info passed around in the URL, when doing GET request**

**Within a response-request cycle, Params is the info I give, so that the response can work**

* **it returns a HASH of PARAMS: key is 'name', value is 'what we submit'**
* **params[:first] = whatever text I submitted**

### ERB: embed ruby into a page

*GREAT EXO week5 day 3: todo\_app : link sql, Sinatra, http methods*

**Views/layout.erb -> will be used as layout for ALL pages**

* this is where you link the CSS, common html stuff, JS, etc

**Html pages on a views folder && extension .erb => here views/home.erb**

* all html files only show the specific content for this page
* embedd ruby in the various pages, in .ebd format

**All CSS, JS, etc in public folder**

Layout page

<!DOCTYPE html>

<html>

<head> with <title>

<title>Hello Sinatra</title>

<link rel="stylesheet" type="text/css" href="/style.css"> *# can add CSS link to all pages in one go, it knows it is inside the public folder*

</head>

<body>

*Yield means execute the block passed on -> get it from app.rb file for the specific html page!*

***Without yield nothing will be passed on! Withouth the = sign nothing will be displayed***

<h1>Welcome to my site</h1>

**<%= yield %>**

</body>

</html>

About page – EMBED RUBY

**<%= this to embed ruby, and = sign to say I want to display on page %>**

**@instance\_variable => if need to embed it in a page/view**

**local\_variable to just do some coding**

<h2>This is the <%= @page %> page </h2>

<p>I know how to code using:</p>

<ul>

<% @languages.each do |language| %>

**! every ruby line is between <% %> !**

<li> <%= language %> </li>

<% end %>

</ul>

On main ruby: app.rb

Require ‘pry’

require ‘sinatra’

requir 'sinatra/reloader' if development?

get '/' do when going on home page

@page = 'home' *# the @page is an instance variable, this is how we pass code to the page*

erb :home  *load the home.erb file – always at the end!!*

end

#### Using forms to get/edit/post info

**Get info with a form**

<form actions= "/" method='get'>

<label for='first'>FIRST NUMBER</label>

**<input type='text' name='first' placeholder='Enter the first number'>**

</form>

Remember… **Params is the info passed around in the URL, when doing GET request**

* **it returns a HASH of PARAMS: key is 'name', value is 'what we submit'**
* **params[:first] = whatever text I submitted**

**Post some new element with a form**

In main.rb I say: when going on url/new render what is on page new.erb

get ‘/items/new do

erb :new

end

So on new.erb I create the form to post the new content

<form action='/items' method='POST'>

<input type='text' name='item' id='item' autofocus>

<input type='text' name='details' id='details' autofocus>

<button>Add task</button>

</form>

=> type buy milk – 2 pints => params is {**"**item**"**=>**"**buy milk**"**, **"**details**"**=>**"**2 pints skimmed**"**}

Need to tell main what to do with the post method on /items

post '/items' do

sql = "insert into items (item, details) values ('#{params['item']}', '#{params['details']}')"

end

***!!! sql values have to be put between ‘quotes’ unless we want to write an integer !!!***

Note on forms: value=’name’ will pre-type a value, placeholder will disappear s soon as we type

**Delete element with a form -- funny syntax --**

<form action="/items/<%= @item['id'] %>/delete" method="post">

<button name="\_method" value="delete"> Delete </button>

</form>

# Nokogiri – Web Scraping

## In the command line, pry

require 'nokogiri'

require 'open-uri'

If I select the whole page

page = Nokogiri::HTML(open('http://google.com'));

I can then call specific element with css type selectors

page.css('#lga')

we can even get text inside an element

page.css(**'**#hplogo div**'**).text

GET everyone’s Codewards score

**%w(**gperrin01 tvc25 nickstoneman**)**.each do |user|

page = **Nokogiri**::HTML(open(**"**http://www.codewars.com/users/#{user}**"**))

score = page.css(**'**.honor**'**).children[**1**].text

puts **"**#{user} score is #{score}**"**

end

gperrin01 score is 62

tvc25 score is 16

nickstoneman score is 8

* [**"**gperrin01**"**, **"**tvc25**"**, **"**nickstoneman**"**]

## Nokogiri in rails

Put in the gemfile gem ‘nokogiri’

rails g task scrape get\_user\_scores

it creates lib/tasks/scrape.rake , and I can update the description

add require 'open-uri' at top of file

require 'open-uri'

namespace :scrape do

desc "Scrape Codewars score for user"

task get\_user\_scores: :environment do

puts 'Getting scores...'

User.all.each do |user|

begin

page = Nokogiri::HTML(open("http://www.codewars.com/users/#{user.name}"))

score = page.css('.honor').text.scan(/\d+/).shift.to\_i

user.update\_attributes score: score

rescue Exception => e

puts "error, the user #{user.name} is not found"

end

end

puts 'done'

end

end

and run rake scrape:get\_user\_scores

NOTE THE BEGIN RESCUE END syntax

Enables to continue although there was an error