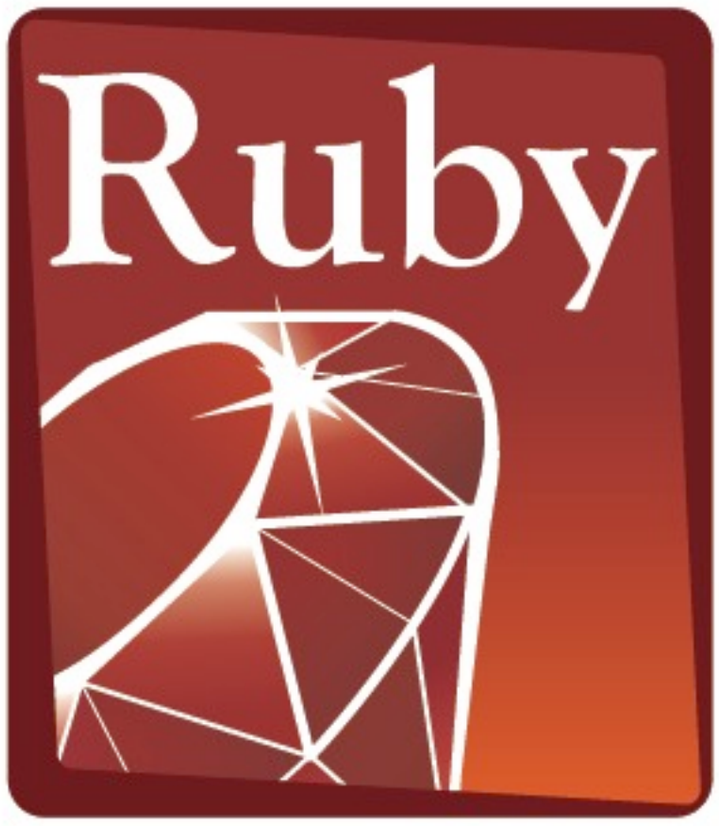
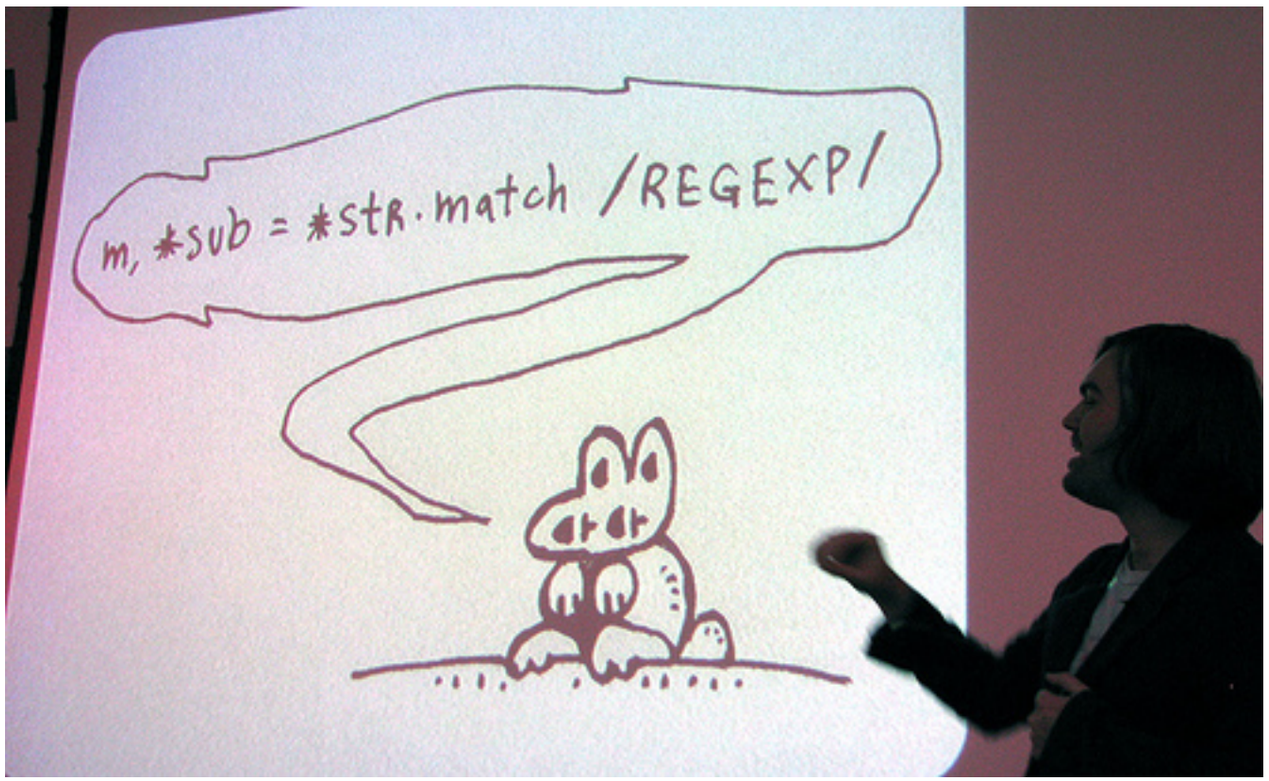


Ruby: The Core Language

Ruby Programming Certificate
UW PCE CP110
Winter 2013
Week 3



I admire programmers who take risks. They aren't afraid to write dangerous or "crappy" code. If you worry too much about being clean and tidy, you can't push the boundaries (I don't think!).

_why

Stuff

- Survey email
- Homework Trouble
- Communicating with me
- No Office Hours
- HipChat should be live only
- Canvas Inbox or Email

Agenda

- Community - Local Meetups
- Project Requirements
- Group Exercise
- Lightning Talks!
- Homework - This Week
- Rake
- Expressions, Operators, Methods & Blocks
- Homework - Last Week / Lab

Community:
Local Meetups

- Ruby at Racer
  - Mondays 7-9
- Seattle Ruby Brigade
  - Tuesdays 7-9
- Beer && Code
  - Every other Wednesday 7-10

Final Project

- Review Requirements
- Ideas
- Q & A

Guest Speaker:
Nell Shamrell

Why Test?

Group Exercise

- Create a Gem
- 20 minutes - see where you get...
- Find your group. GO!

Lightning Talks

- 5 - 7 minutes
- Have fun
- Project your voice

Wk3 Homework

- Write some methods
- Write some specs
- Topics
  - Expressions
  - Operators
  - Methods
  - Blocks

Break!

Rake

- Ruby version of make; A DSL to define tasks
- Great at managing dependencies
- Easy way to automate your project

```
# Rakefile
desc "Clean up and Update everything"
task :clean_update do
  system( clean_up_script.sh )
  MyClass.update_everything('./path')
end

$ rake clean_update
```

- Default task should run you tests

Expressions

- evaluate to a value
- literals are primary expressions

```
# [2], 'Hello', File
# true, false, nil, self
```

- compound expressions leverage operators
- all expressions, including control flow and class definition, return a value

Variables

- holds a value. well, actually hold a reference to an object
- initialize first to prevent NameError exception

```
a = 1
# a is 1

a = b
# => NameError: undefined local
variable or method 'b' for main:Object
```

Not a copy

```
a = "Wow"
b = a
# b is also "Wow"
# b is a REFERENCE to a
# a.object_id is same as b.object_id

c = a.dup
# c is "Wow", a COPY of a

b[0] = "P"
# b is now "Pow"

# a is also changed to "Pow"

# c is a copy, so it is still "Wow"
```

Variable types

- Local: `my_var`
- Global: `$password`
- Class instance: `@name`
- Class: `@@counter`
- Constant: `MY_CONST`
- Scoped: `MyClass::SCOPED_CONST`

Assignment

- assignment (lvalue) vs. reference (rvalue)

```
a = b
# lvalue = rvalue
```

- abbreviated assignment

```
a += 1

# same as
a = a + 1
```

Parallel Assignment

```
a, b = 1, 2
# a is 1, b is 2

a, b = [1, 2]
# a is 1, b is 2

a, b, c = [1, 2]
# same, but c is nil

# swap values
a, b = b, a
# b is 1, a is 2
```

*splat operator

```
array = [1,2]

a, b, c = array, 3, 4
# a is [1,2]

a, b, c = *array, 3, 4
# a is 1, b is 2, c is 3

# not really an operator
# *array by itself is not an expression
```

Operators

- Unary: `- ! * ~`

```
!false, -3
```

- Binary: `+ - * ** << =~ ||= +=`

```
1 + 1, array << object
```

- Ternary: `?:`

```
my_variable ? ‘True’ : ‘False’
```

Methods

messages get passed to objects

- the object
- the message
- the parameters
- the block
- the body

Calling Methods

`object.message(parameter) {block}`

```
my_method
my_method parameter
my_method(parameter)
my_method parameter, another_parameter
my_object.some_method

# no block passed
register user, "CP110", 2013

# with block
register user, "CP110", 2013 do |name, course|
  tweet! "#{name} just registered for #{course}"
end
```

Defining Methods

`def message(parameters) {block}`

- where you define the method determines what object(s) method is attached to
- prepend last parameter with & to name the block being passed in. Only required for using `block.call`

```
def obfuscate string, &my_block
```

- unnamed blocks can be called, or yielded to, with `yield`

the object

- object that receives the message
- remember, every object has methods
- can be a class
- defaults to self (whatever the current scope is)
- methods defined outside of a class/module definition get added to Object

```
# in irb
def my_bare_method; end
Object.respond_to? :my_bare_method
# => true
```

the message

- only part that is required
- begin message (aka method) name with lowercase letter or _ and use snake_case

- predicate methods end in ?

```
# [].empty? is true
```

- dangerous method end in !

```
# my_array.uniq! modifies my_array
# my_array.uniq does not
```

the message (cont.)

- attribute setter method names end in =

```
def name= str
  @name = str
end
obj.name = 'Brandon'
```

- Index lookup (-ish) methods can use [ ]

```
class MyObj
  def [] num
    @collection[num]
  end
end
# obj[index] now returns @collection[num]
```

the parameter(s)

- 0 or more
- comma separated
- defaults can be defined

```
def say str1, str2=nil
  "#{str1} --- #{str2}"
end

say 'hi', 'bye'
# => 'hi --- bye'
say 'hi'
# => 'hi --- '
```

the block

- arbitrary code (a nameless function) passed to a method with `{ code }` or `do code end`
- for example, in the each method, the block passed in defines what to do with each thing
- any method can receive a block
  - the method definition does not have to name it as a method parameter
  - ignored if not explicitly used by the method

more on blocks

- run when the method calls `yield` or `block.call`
- the method RECEIVES the block
  - then YIELDS control to the block
- keyword `block_given?` is true when current method DID receive a block; common usage:

```
# run block if it exists
yield if block_given?
```

- block parameters are defined inside the method by any parameters passed to yield

block parameters

- completely independent of method parameters
- block parameters live inside vertical bar (aka pipe) characters and are passed to the block by the method

```
def my_method method_param
  value_for_block = method_param.do_something
  yield value_for_block
end
my_method param { |block_param|
  # use block_param here
}
my_method param do |block_param| # same as above
  # use block_param here
end
```

Simple Block Example

```
def run
  print "I'm running your code!"
  yield
end

run {print 'POW!'}
prints: "I'm running your code! POW!"

# again, but call it without a block
run
# => LocalJumpError: no block given
```

Another Block Example

```
def repeat num
  print "I'm running your code!"
  num.times do
    yield if block_given?
  end
end
```

```
repeat(2) {print 'POW!'}
prints: "I'm running your code! POW!POW!"

repeat(2) # no block provided
prints: "I'm running your code!"
```

```
module Enumerable

  def map
   result = []
   self.each do |object|
     result << yield object
   end
    result
  end

end

[1, 2, 3].map { |n| n * 2 }
# => [2, 4, 6]
```

Lab

Homework

(will be posted soon)

- Reading assignment
- Homework
- Wk3 Quiz