CS 683 Emerging Technologies
Fall Semester, 2006
Doc 15 Ruby Blocks, RE, Case, Exceptions, Classes
Oct 12, 2006

References

Programming Ruby, The Pragmatic Programmers' Guide, Dave Thomas, ISBN 0-9745140-5-5

Version 1 of the above book in on-line at:
http://www.rubycentral.com/book/index.html

Blocks, Procs & lambda

Blocks as variables

```
x = {puts 'hello'}
#Error
```

lambda short-cut for Proc.new

```
hello = lambda {puts 'hello'}
hello.call

add = lambda {|x| x + 1}
add.call(5)

def do_it(bar)
  bar.call
end

bye = Proc.new {puts 'bye'}

do_it(bye)
do_it(hello)
```

Block as Parameter

```
def as_arg(&x)
    x.call
end

def normal
    yield
end

as_arg {puts 'hello'}
normal {puts 'hello'}
```

Proc as Block

```
hello = lambda {puts 'hello'}

def normal
    yield
end

normal {puts 'hello'}
normal(&hello)
```

Some Magic

```
def local
    x = 5
    return lambda {puts x}
end

x = 3

action = local()
action.call
```

Output

```
5
```

Variable Scope & Blocks, Oh no!

```
[1, 2, 3].each {|x| y = x + 1}
puts y , x
```

Result

Error, x & y not defined for puts

```
if false
  x = nil
  y = nil
end
[1,2, 3].each {|x| y = x + 1}
puts y , x
```

Result

4

3

```
x = nil
y = nil
[1, 2, 3].each {|x| y = x + 1}
puts y , x
```

Result

4

3

Regular Expressions

Regular Expressions

Creation

```
a = Regexp.new('^\s*[a-z]')
b = /^\s*[a-z]/
c = %r{^\s*[a-z]}
```

Match

```
name = 'Roger Whitney'
name =~ /g/
/g/ =~ name
```

Pattern Matching Variables

$& what was matched by pattern	$& 'g'
$` part of string preceding match	$` 'Ro'
$' part of string after match	$' 'er Whitney'
$~ MatchData object	

show_regexp for Later Slides

```
def show_regexp(string, pattern)
  if string =~ pattern
    "#{$`}<<#{$&}>>#{$'}"
  else
    'no match'
  end
end
```

```
show_regexp('cat', /a/)          c<<a>>t
show_regexp('yes | no', /\|/)    yes <<|>> no
```

Anchors

^	beginning of line
$	end of line
\A	beginning of string
\Z, \z	end of string
\b	word boundaries
\B	nonword boundaries

show_regexp('cat rat\nrat cat',/^rat/)	no match
show_regexp("cat rat\nrat cat",/^rat/)	cat rat <<rat>> cat
show_regexp("cat rat\nrat cat",/cat/)	<<cat>> rat rat cat
show_regexp("cat rat\nrat cat",/cat\Z/)	cat rat rat <<cat>>
show_regexp("cat rat\nrat cat",/\bat/)	no match
show_regexp("cat rat\nrat cat",/at/)	c<<at>> rat rat cat

Character Classes

[abc]	matches characters a, b or c
[^abc]	matches all characters except a, b, or c
.	matches any character except newline
[a-z]	matches all characters between a & z inclusive

\d	[0-9]
\D	[^0-9]
\s	[\s\t\r\n\f]
\S	[^\s\t\r\n\f]
\w	[A-Za-z0-9]
\W	[^ A-Za-z0-9]

show_regexp("bat cat rat", /[aeiou]/)	b<<a>>t cat rat
show_regexp("bat cat rat", /[\s]/)	bat<< >>cat rat
show_regexp("bat cat rat", /\s/)	bat<< >>cat rat
show_regexp("bat cat rat", /[\s][aeiou]/)	no match
show_regexp("bat cat rat", /[\s].[aeiou]/)	bat<< ca>>t rat
show_regexp("bat cat rat", /[a-z]/)	<<b>>at cat rat
show_regexp("bat cat rat", /[a-z]\s[a-z]/)	ba<<t c>>at rat
show_regexp("bat cat rat", /[a-z].[a-z]/)	<<bat>> cat rat

Repetition

r*	matches zero or more occurrences of r
r+	matches one or more occurrences of r
r?	matches zero or one occurrences of r
r{m.n}	matches at least m and at most n occurrences of r
r{m,}	matches at least m occurrences of r
r{m}	matches exactly m occurrences of r

show_regexp("bat cat rat sat", /[a-z]*/)	<<bat>> cat rat sat
show_regexp("bat cat rat sat", /\s.*\s*/)	bat<< cat rat sat>>
show_regexp("bat cat rat sat", /\s.*?\s/)	bat<< cat >>rat sat
show_regexp("bat cat rat sat", /(a|t){2,3}/)	b<<at>> cat rat sat

Substitution

a ="bat cat rat sat"	
a.sub(/[^aeiou]/, 'x')	xat cat rat sat
a.gsub(/[^aeiou]/, 'x')	xaxxxaxxxaxxxax
a.sub(/^./) {|x| x.upcase}	Bat cat rat sat
a.gsub(/[a]/) {|x| x.upcase}	bAt cAt rAt sAt
a.gsub(/\b\w/) {|x| x.upcase}	Bat Cat Rat Sat
a.gsub(/(\w+)\s(\w+)/, '\2 \1')	cat bat sat rat

Boolean Expressions and Control Structures

Boolean Expressions

Value	Boolean Value
nil, false	false
all other values	true

Operator	Explanation
or, ||	or, shortcircuit evaluation
and, &&	and, shortcircuit evaluation
not, !	negation

```
x = if 0
    5
  else
    10
  end
```

What is x?

Equality

Operator	Explanation
==	Equal value
===	and, shortcircuit evaluation
not, !	negation
<=>	Returns -1, 0, 1
=~	Regular expression match
eql?	True if receiver & argument have same type and equal values
equal?	compares object IDs

Case on Steroids

```
leap = case
    when year % 400 == 0: true
    when year % 100 == 0; false
    else year % 4 == 0
    end
puts leap
```

```
case input_line
    when "test"
        run_test_cases
        print_test_results
    when /p\s*(\w+)/
        print_source_code
    when "q", "quit"
        exit
end
```

```
grade = case score
    when 0..60: 'F'
    when 61..70: 'D'
    when 71..80: 'C'
    when 81..90: 'B'
    when 90..100: 'A'
    else        'Illegal score'
end
```

Break, Redo, Next & Retry

break	terminates immediate enclosing loop
redo	repeats loop from start without updating condition or fetching next element
next	start next iteration of loop
retry	restarts iterator loop

```
i = 0
loop do
  i +=1
  next if i < 3
  print i
  break if i > 4
end
```

Output

345

```
for k in 1..5
  puts "Now at #{k}. Restart?"
  retry if gets =~ /^y/
end
```

Input/Output

Now at 1. Restart? n
Now at 2. Restart? n
Now at 3. Restart? y
Now at 1. Restart? n
Now at 2. Restart? n
Now at 3. Restart? n
Now at 4. Restart? n
Now at 5. Restart? n

Ranges

a..b from a to b including b
a...b from a to b, excluding b

(2..4).to_a	[2, 3, 4]
(2...4).to_a	[2,3]
('a'..'d').to_a	['a', 'b', 'c', 'd']
('car'..'cat').to_a	['car', 'cas', 'cat']
(1..5) === 3	true
(1..5).include?(3)	true
(1..5).min	1
(1..5).reject {|k| k < 3}	[3, 4, 5]

Exceptions

Exceptions

```
begin
  file = File.new('foo', 'w')
rescue SyntaxError => typo
  puts 'there was a typo' + typo
  raise typo
rescue NameError, IOError
  if file.path = 'foo'
    file = File.new('bar', w)
    retry
  else
    raise
  end
rescue                    #defaults StandardError
  puts 'error' + $!
  raise
rescue Exception => all_errors    #catches all errors
  puts 'This is the top'
else
  puts 'OK'
ensure
  file.close if nil != file
end
```

Exception Information

```
begin
  raise 'Extra info'
rescue RuntimeError => error
  puts "Message: " + error
end
```

Output

```
Message: Extra info
```

```
begin
  raise IndexError, 'Extra info'
rescue IndexError => error
  puts "Message: " + error
  puts "Message: " + error.message
end
```

Output

```
Message: Extra info
Message: Extra info
```

```
def test
  raise IndexError
end

begin
  test
rescue IndexError => error
  puts error.backtrace
end
```

Output

```
RaisingException.rb:19:in `test'
RaisingException.rb:22
Support/tmruby.rb:126:in `load'
Support/tmruby.rb:126
Support/tmruby.rb:100:in `fork'
Support/tmruby.rb:100
```

Catch-Throw

```
def sample
  x = 1
  throw :foo if x == 1
end

catch :foo do
  puts 'start'
  sample
  puts 'end'
end
```

Output

```
start
```

Classes

Defining a Class

```
class NoState
    def hello
        return 'hello'
    end

    def increase(x)
        x + 1
    end
end
```

```
require 'test/unit'

class TestExample < Test::Unit::TestCase
    def test_hello
        object = NoState.new
        assert_equal('hello', object.hello )
        assert( 'hello' == object.hello() )
        assert( 2 == object.increase(1)      )
    end
end
```

Instance Variables

```
class Book
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end

    def to_s
        "Book: #@title by #{@author}}"
    end
end
```

```
require 'test/unit'

class TestExample < Test::Unit::TestCase
    def test_new
        cat = Book.new("Cat", "Me", "1990")
        assert_instance_of(Book, cat)
        assert( "Book: Cat by Me" == cat.to_s)
    end
end
```

Instance Variable Access

```
class Book
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end

    def author
        @author
    end

    def author=(new_author)
        @author = new_author
    end
end
```

```
require 'test/unit'

class TestExample < Test::Unit::TestCase
    def test_author
        cat = Book.new("Cat", "Me", "1990")
        assert_not_nil( cat)
        assert( 'Me' == cat.author)
        assert( 'Me' == cat.author() )
        cat.author = 'You'
        assert_equal( 'You',   cat.author )
        assert_raise(NoMethodError) {cat.title}
    end
end
```

Attributes

```
class Book
    attr_reader :title, :author, :date
    attr_writer :title

    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end
end
```

```
require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
    def test_author
        cat = Book.new("Cat", "Me", "1990")
        assert( 'Me' == cat.author)
        assert( 'Cat' == cat.title)
        cat.title = 'Dog'
        assert_equal( 'Dog', cat.title)
        assert_raise(NoMethodError) do
            cat.author = 'Me'
        end
    end
end
```

Access Control

```
class Foo
        def method1 #default is public
        end

    protected
        def bar
        end

    private
        def foo
        end
    public
        def ok
        end
end
```

Public methods

Accessible by anyone

Protected methods

Accessible by defining class and subclasses

Private Methods

Accessible by defining class

Alternative Access Declaration

```
class Foo
    def method1
    end

    def method2
    end
    ....
    public    :method1, method3
    protected     :method2
    private   :method4, method5
end
```

Class Methods and Variables

```
class Book
    @@count = 0

    def Book.count
        @@count
    end

    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
        @@count += 1
    end
end
```

```
require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
    def test_count

        assert( Book.count == 0, 'First count')
        cat = Book.new("Cat", "Me", "1990")
        assert( Book.count == 1, 'Second count')
        assert_raise(NoMethodError) {cat.count}
    end
end
```

Singleton

```
class One
    private_class_method :new

    @@instance = new

    def One.instance
        @@instance
    end
end
```

```
require 'test/unit'
require 'one'

class TestSingleton < Test::Unit::TestCase
    def test_unique
        assert_raise(NoMethodError) {One.new}
        assert_kind_of(One, One.instance)

        assert_same( One.instance, One.instance)
        assert( One.instance.equal?( One.instance))
        assert( One.instance.object_id ==
                One.instance.object_id)
    end
end
```

Singleton - Using builtin Singleton

```
require 'singleton'

class One
    attr_accessor :foo
    include Singleton
end
```

Operators

```
class Book
    attr_reader :title
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end

    def <(book)
        return false if book.kind_of(Book)
        title < book.title
    end
end
```

```
require 'test/unit'
require 'book'

class TestExample < Test::Unit::TestCase
    def test_operator
        cat = Book.new("Cat", "Me", "1990")
        dog = Book.new("Dog", "Me",
"1990")
        assert( cat < dog)
    end
end
```

Inheritance

```
class Book
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end

    def to_s
        "Book: #@title by #{@author}"
    end
end
```

```
class ElectronicBook < Book
    def initialize(title, author, date, format)
        super(title, author, date)
        @format = format
    end
    def to_s
        super + " in #{@format}"
    end
end
```

Adding Methods to Existing Classes

```
class Fixnum
    def foo
        self + 1
    end
end
```

```
require 'test/unit'

class TestAddingMethods < Test::Unit::TestCase
    def test_integer_foo

        assert( 1.foo == 2 )
        assert( -45.foo == -44)

    end
end
```

alias

```
class Fixnum
  alias old_plus +

  def +(other)
    old_plus(other).succ
  end
end
```

```
1 + 2   -> 4
```

Adding Methods to an Object

```
class Book
    attr_reader :title
    def initialize(title, author, date)
        @title = title
        @author = author
        @date = date
    end
end
```

```
require 'test/unit'
require 'book'
class TestExample < Test::Unit::TestCase
    def test_singelton_method
        cat = Book.new("Cat", "Me", "1990")
        dog = Book.new("Dog", "Me", "1990")

        def cat.foo
            5
        end

        assert( cat.foo ==5 )
        assert_raise(NoMethodError) {dog.foo}
    end
end
```