CS 683 Emerging Technologies
Fall Semester, 2006
Doc 14 Ruby Intro
Oct 10, 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

Reading Assignment

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

Date	Chapters
Oct 9	Ruby.new More About Methods Containers, Blocks, and Iterators
Oct 11	Standard Types Classes, Objects, and Variables Expressions
Oct 16	Exceptions, Catch, and Throw Modules Basic Input and Output Threads and Processes

Chapter 22 The Ruby Language is very useful

Some Links

Main web site: http://www.ruby-lang.org/en/

Some Ruby Aware IDEs

Ruby extensions to editors:
http://www.rubygarden.org/ruby?EditorExtensions

Arachno
http://www.ruby-ide.com/ruby/ruby_ide_and_ruby_editor.php
good editor & debugger,
Windows & Linux,
Commercial with 30 day free trial

Ruby Eclipse plugin - rdt
http://sourceforge.net/projects/rubyeclipse

Formating

```
a = 1
b = 2; c = 'cat'
d = 1 + 2 +
    5       #no '\' needed
e = 1 + 2
    + 3     #'\' needed
```

Number Literals

1234	1234	Fixnum
0d1234	1234	Fixnum
1_234	1234	Fixnum
-1234	-1234	Fixnum
0xaa12	43538	Fixnum hex
0377	255	Fixnum octal
0b10_110	22	Fixnum binary
123_456_678_912_345	12345667891234 5	Bignum
12.34	12.34	Float
0.123e3	123.0	Float
1234e-2	12.34	Float

Fixnum = native machine word minus 1 bit

Float = machines double data type

String Literals - Single Quoted

'stuff' or %q/stuff/

'hello'	hello
'backslash \'\\\'	backslash '\'
%q!backslash '\'!	backslash '\'
%q/this is a single quoted string/	this is a single quoted string
%q[this is a 'single quoted' string]	this is a 'single quoted' string
%q(look (nesting) works)	look (nesting) works

String Literals - Double Quoted

"stuff", %Q/stuff/ or %/stuff/

Expands \n, \t etc and #{ruby_code }

"With \"double Quotes\""	With "double Quotes"
%Q/With "double Quotes"/	With "double Quotes"
%<With "double Quotes">	With "double Quotes"
cat = 5	
%[cat = #{cat}]	cat = 5
%(cat + 1 is #{cat + 1})	cat + 1 is 6
%{more complex #{dog = 3; cat + dog}}	more complex 8

Literal Arrays

```
a = [1, 'dog', 12]
a[0]
a[1] = 3
a[5] = 'what now'
puts a
```

```
a = ['this', 'is', 'painful', 'to', 'type']
b = %w{ this is a shortcut for an array of strings}
```

Output

```
1
3
12
nil
nil
what now
```

Short-cut Notation

a = 2, 3, 4

b = [2, 3, 4]

Array Methods

Array is a class

```
stack = Array.new
stack.push(5)
stack.push(4)
stack.push(3)
sorted = stack.sort
```

Listing methods

```
puts Array.public_instance_methods
```

(prints 118 methods)

Array docs at
http://www.ruby-doc.org/core/

Hash

```
aHash = {
  'cat' => 'mammal',
  'ant' => 'insect',
  'dog' => 'mammal'
}
aHash['lizard'] = 'reptile'
puts aHash['dog']
```

Assignments

```
a = b = 1 + 2
a = (b = 1 + 2) + 3
a, b = b, a

x = 0
a, b, c = x, (x += 1), (x += 1)
```

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

a, *b = [1, 2, 3, 4]
puts b
```

Defining Functions

```
def test_me
  return 1
end

puts test_me
puts test_me()
```

```
def two_args(a, b)
  return a + b
end

puts two_args 1, 2          # warning
puts two_args(1, 2)
```

```
def one_arg(x)
  return x + 1
end

puts one_arg(1)
puts one_arg 2              #Generates warning
puts one_arg 2 + 3          #Generates warning
```

Default Parameters

```
def default_values(a, b='cat', c='dog')
  "#{a}, #{b}, #{c}"
end

puts default_values(1)
```

Expanding Array Arguments

```
def four(a, b, c, d)
  "#{a}, #{b}, #{c}, #{d}"
end

x = [1, 2, 3]
puts four(0, *x)
```

Naming Convetions

Local	Global	instance	Class	Constants & Class names
cat	$dog	@bird	@@x	PI
cat_tail	$DOG	@bird_toe	@@x_pos	String
_26	$dog_bone	@X	@@N	MyClass

Control Structures

```
if x > 5
  puts "Greater than 5"
elsif x < 3
  puts "Less than 3"
else
  puts "looks like 4"
end
```

```
while x < 10
  puts x
  x += 1
end
```

Assigning an If

```
x = gets.to_i

if x > 12
  y = 6
else
  y = 1
end
puts y
```

```
y = if x > 12
      6
    else
      1
    end
puts y
```

More ifs

```
if x > 12 then
  y = 6
else
  y = 1
end
```

```
if x > 12:
  y = 6
else
  y = 1
end
```

```
if x > 12 then y = 6
else y = 1
end
```

```
if x > 12: y = 6
else y = 1
end
```

unless

negation of if

```
unless x <= 12
  y = 1
else
  y = 6
end
```

Statement Modifiers

```
grade = 102
if grade > 100
  puts "Invalid Grade"
end

puts "Invalid Grade" if grade > 100
```

```
powers = 2
while powers < 100
  powers = powers*powers
end
puts powers

powers = 2
powers = powers*powers while  powers < 100
```

Blocks

```
{ puts "Hello World" }

do
  x = x + 1
  y = y -1
end
```

Calling Blocks

```
def block_example
  puts "Start"
  yield
  yield
  puts "End"
end
```

```
block_example {puts "Hello"}
```

Output

```
Start
Hello
Hello
End
```

```
x = 1
block_example do
  x +=1
end
puts x
```

Output

```
Start
End
3
```

Blocks with Arguments

```
def call_block
  yield(4)
end

call_block { |x| puts x}

call_block do |x|
  puts x
end
```

```
def call_block
  yield(4, 5)
end

call_block {|x , y| puts x + y}

call_block do |x , y|
  puts x + y
end
```

Parameters and Block

```
def call_block(a, b)
  yield(a, b + 1)
end

call_block(1,2) {|x , y| puts x + y}
```

Blocks are Closures

```
x = 3
def test
  x = 1
  puts x
end
test
puts x
```

Output

1
3

```
x = 3
def test
  x = 1
  yield
end
test {puts x}
puts x
```

Output

3
3

Iterators

Code	Output
[1, 2, 3].each {|x| puts x}	1 2 3
x = 4 4.times do puts x x *= 2 end	4 8 16 32
2.upto(5) {|x| puts x}	2 3 4 5
3.step(10, 3) {|x| puts x}	3 6 9

Iterators

Code	Output
a = [1, 2, 3].collect {|x| x + 3} puts a	4 5 6
a = [1, 2.345, 3].select {|x| x.integer?} puts a	1 3
a = [1, 2.345, 3].find_all {|x| x.integer?} puts a	1 3
('g'..'j').each {|char| puts char}	h i j

Variable-Length Argument Lists

```
def variable_number(a, *b)
  (0..b.length - 1).each {|k| puts b[k]}
end

variable_number(1,2, 3, 4, 5)
```

Output

```
2
3
4
5
```