CS 683 Emerging Technologies
Fall Semester, 2005
Doc 17 Ruby Intro
Contents

References

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

Reading Assignment

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

On-line version

Date	Chapters
Nov 3	Ruby.new Classes, Objects, and Variables Containers, Blocks, and Iterators
Nov 8	Standard Types More About Methods Expressions
Nov 10	Exceptions, Catch, and Throw Modules Basic Input and Output Threads and Processes

Chapter 22 The Ruby Language is very useful

Future Rails Reading

On Nov 15 I plan to start going through

Agile Web Development with Rails, Thomas & Hansson, 2005, Pragmatic Bookshelf, ISBN 0-9766940-0-X

Quotes

Dynamic types are stronger than static types, as they don't flee the field at runtime.

Brian Foote

Static types give me the same feeling of safety as the announcement that my seat cushion can be used as a floatation device.

Don Roberts

Ruby

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

Some Basic Boring Stuff
Some 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	123456678912345	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

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 + 1 is 6
%(cat + 1 is #{cat + 1})	cat + 1 is 6
%{more complex #{dog = 3; cat + dog}}	more complex 8

Defining Functions

```
def test_me
  return 1
end

puts test_me
puts test_me()

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

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

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

Naming Convention

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

Arrays & Hashes

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

Output

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

String Array Shortcut

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

Hashes

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

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
```

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

Two Block delimiters

```
{ puts "Hello World" }

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

Calling a Block

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

block_example {puts "Hello"}
```

Output

```
Start
Hello
Hello
End
```

yield calls the block passed to the method

Second Block Example

```
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
```

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

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