Contents

Reading Assignment........................................3
Ruby...............................................................6
  Some Formatting........................................7
  String Literals...........................................8
  Defining Functions.....................................9
  Naming Convention.................................10
  Arrays & Hashes..................................11
  Control Structures.................................12
  Blocks.................................................14

Copyright ©, All rights reserved. 2005 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
References

# Reading Assignment


[On-line version](#)

<table>
<thead>
<tr>
<th>Date</th>
<th>Chapters</th>
</tr>
</thead>
</table>
| 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

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


Some Ruby Aware IDEs

Ruby extensions to editors:

- [http://www.rubygarden.org/ruby?EditorExtensions](http://www.rubygarden.org/ruby?EditorExtensions)

Arachno

  - good editor & debugger,
  - Windows & Linux,
  - Commercial with 30 day free trial

Ruby Eclipse plugin - rdt

- [http://sourceforge.net/projects/rubyeclipse](http://sourceforge.net/projects/rubyeclipse)
Some Basic Boring Stuff
Some Formatting

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

Number literals

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Fixnum</td>
</tr>
<tr>
<td>0d1234</td>
<td>Fixnum</td>
</tr>
<tr>
<td>1_234</td>
<td>Fixnum</td>
</tr>
<tr>
<td>-1234</td>
<td>Fixnum</td>
</tr>
<tr>
<td>0xaa12</td>
<td>Fixnum hex</td>
</tr>
<tr>
<td>0377</td>
<td>Fixnum octal</td>
</tr>
<tr>
<td>0b10_110</td>
<td>Fixnum binary</td>
</tr>
<tr>
<td>123_456_678_912_345</td>
<td>Bignum</td>
</tr>
<tr>
<td>12.34</td>
<td>Float</td>
</tr>
<tr>
<td>0.123e3</td>
<td>Float</td>
</tr>
<tr>
<td>1234e-2</td>
<td>Float</td>
</tr>
</tbody>
</table>

Fixnum = native machine word minus 1 bit
Float = machines double data type
String Literals
Single Quoted

'stuff' or %q/stuff/

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hello'</td>
<td>hello</td>
</tr>
</tbody>
</table>
| '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}

<table>
<thead>
<tr>
<th>String</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;With &quot;double Quotes&quot;&quot;</td>
<td>With &quot;double Quotes&quot;</td>
</tr>
<tr>
<td>%Q/With &quot;double Quotes&quot;/</td>
<td>With &quot;double Quotes&quot;</td>
</tr>
<tr>
<td>%&lt;With &quot;double Quotes&quot;&gt;</td>
<td>With &quot;double Quotes&quot;</td>
</tr>
<tr>
<td>cat = 5</td>
<td></td>
</tr>
<tr>
<td>%[cat = #{cat}]</td>
<td>cat + 1 is 6</td>
</tr>
<tr>
<td>%(cat + 1 is #{cat + 1})</td>
<td>cat + 1 is 6</td>
</tr>
<tr>
<td>%{more complex #{dog = 3; cat + dog}}</td>
<td>more complex 8</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Local</th>
<th>Global</th>
<th>instance</th>
<th>Class</th>
<th>Constants &amp; Class names</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>$dog</td>
<td>@bird</td>
<td>@@x</td>
<td>PI</td>
</tr>
<tr>
<td>cat_tail</td>
<td>$DOG</td>
<td>@bird_toe</td>
<td>@@x_pos</td>
<td>String</td>
</tr>
<tr>
<td>_26</td>
<td>$dog_bone</td>
<td>@X</td>
<td>@@N</td>
<td>MyClass</td>
</tr>
</tbody>
</table>
Arrays & Hashes

```ruby
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

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

Hashes

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

```ruby
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

```ruby
{ puts "Hello World" }

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

Calling a Block

```ruby
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

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

Output

```
Start
End
3
```
**Blocks with Arguments**

```ruby
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}
```