Introduction to Ruby

SWEN-250
Personal Software Engineering
A Bit of History

• Yukihiro "Matz" Matsumoto
  – Created a language he liked to work in.
  – Been around since mid-90s.
  – Caught on in early to mid 00s.

• Lineage
  – Smalltalk – dynamic, OO-centric
  – CLU – yield to blocks
  – Pascal – basic concrete syntax
  – AWK / Python / Perl – scripting & regular expressions
  – Matz's own predilections
Ruby Characteristics

• Everything is an object – *everything*.
  – 3.times { puts "hello" }
  – "Mike is smart".sub(/Mike/, "Pete")
  – str = str[0..9] unless str.length < 10

• Every statement is an expression:
  – Generally the last value computed.
  – No need for return – but it's there anyway.

• Rich built in data types:

<table>
<thead>
<tr>
<th>String</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Unbounded numbers (factorial)</td>
</tr>
<tr>
<td>Hash</td>
<td>Blocks &amp; procs</td>
</tr>
<tr>
<td>RegExp</td>
<td>Anonymous functions</td>
</tr>
</tbody>
</table>
Exploring Ruby

- ri – Ruby information
- irb – Interactive Ruby
- Script files: filename.rb
Ruby Control Structures: Selection

```ruby
if condition
  statements
elsif condition
  statements
else
  statements
end

unless condition
  statements
end
```

**Conditions in Ruby**

Comparisons, etc., return a boolean:
- `true` (the only member of TrueClass)
- `false` (the only member of FalseClass)

**Evaluating conditions**

- `false` evaluates to false.
- `nil` evaluates to false.
- Everything else is `true` (including 0).

**Statement Modifiers (a la Perl)**

```ruby
statement if condition
statement unless condition
```
Ruby Control Structures: Loops

while condition
    statements
end

until condition
    statements
end

begin
    statements
end while condition

begin
    statements
end until condition

Early Termination
next
break
redo

We don't need no stinkin' loops!
Iterators

• Explict loops are rare in Ruby
• Instead, we usually use iterators
  – Iterators are defined on collection classes
  – "Push" elements into a block one at a time.
  – The basic iterator is `each`.
  – Show with arrays (the simplest collection)

```ruby
fibo = [ 1, 2, 3, 5, 8 ]
fibo.each { | value | puts "The next value is #{value}" }
fibo.each_index { | i | puts "fibo[#{i}] = #{fibo[i]}" }
fibo.select { | value | value % 2 == 1 }
fibo.inject(0) { | sum, value | sum += value }
puts "Total = #{fibo.inject(0) { | s, v | s += v }}"
```
But, For Completeness

- **loop**
  
  ```ruby
  loop { puts "forever" }
  loop do
    line = gets
    break if ! line
    puts line
  end
  ```

- **for statement**

  ```ruby
  for v in collection
    statements
  end
  ```

  ```ruby
  collection.each do |v|
    statements
  end
  ```
Strings

• Literals
  "abcdef" vs. 'abcdef'
  "abc #{3 % 2 == 1} def"
  %q{xyz#{1}} → non-interpolate
  %Q{xyz#{1}} → interpolate

• Operators
  + and +=
  s1 = "a" + "b" ; s1 += "c"
  *
  "oops! " * 3
  []
  should be obvious, but "abcd"[1..2]
  == < <=> comparisons
  =~ and !~
  r.e. match (and not match)

• Some of the methods (many have ! variants)
  capitalize
  sub(r.e, str)
  downcase
  include?(str)
  upcase
  index(str or r.e.)
Arrays

- **Literals**
  
a = [ 1, "foo", [ 6, 7, 8 ], 9.87 ]
  
b = %w{ now is the time for all good men } → Interpolated array of words

- **Operators**
  
  & (intersection)    + (catenation)    - (difference)
  
  * _int_ (repetition)    * _str_ (join w/_str_ as separator)
  
  []   []=  as expected for simple indices
  
  << obj (push on end)

- **Some of the methods**
  
  [1, "hello", 3].collect { |v| v * 2 }  # alias map
  
  [1, 2, 5].include?(2)
  
  [1, 2, 5].first    [1, 2, 5].last
  
  [1, 2, 5].length    [1, 2, 5].empty?
Hashes

- **Literals**
  
  ```
  \{ "door" => "puerta", "pencil" => \texttt{lapiz} \}
  ```

  ```
  \texttt{new} Hash( \texttt{default} )
  ```

- **Operators**

  ```
  h[\texttt{key}] \quad h[\texttt{key}] = \texttt{value}
  ```

- **Some methods**

  ```
  \texttt{each} \quad \texttt{each\_key} \quad \texttt{each\_value}
  ```

  ```
  \texttt{empty?} \quad \texttt{has\_key?} \quad \texttt{has\_value?} \quad \texttt{size}
  ```

  ```
  \texttt{keys} (\texttt{returns array}) \quad \texttt{values} (\texttt{returns array})
  ```

  ```
  \texttt{sort} (\texttt{returns an array of 2-element arrays})
  ```

  ```
  \texttt{sort} \{ |p1, p2| \texttt{expression returning} -1, 0, +1 \}
  ```
I/O

- **Class File**
  
  ```ruby
  f = File.new(name, mode)
  ```
  
  - *name* is a string giving the file name (host dependent).
  - *mode* is an access string: "r", "rw", "w", "w+"

  ```ruby
  f.close
  ```
  
  ```ruby
  f.puts, f.printf, f.gets, etc.
  ```
  
  - puts, printf are implicitly prefixed by $stdout.
  - gets is implicitly prefixed by $stdin

  ```ruby
  File.open(name, mode) block – open the file name, call block with the open file, close file when block exits.
  ```

- **Class Dir**
  
  ```ruby
  d = Dir.new(name) – open named directory.
  ```
  
  ```ruby
  d.close
  ```
  
  ```ruby
  Dir.foreach(name) block – pass each file name to block.
  ```
RegExps

• Literals

/regular expression/
%r{regular expression}

• Some examples

"xxAAyyBBzz".gsub(/A+[^B]*B+/,'\&<->\&')

"xxAAyyBBzz".gsub(/(A+)([^B]*)(B+)/,'\3\2\1')

"xx(AA)Azz".gsub(/\(A+\)/,'###')
Functions

- call: puts "abc" or puts("abc")
- define:
  def putNtimes(string, count)
  puts string * count
  end

Requiring modules

require string

- Looks for string.rb and imports whatever is in there.
- Typically service functions, classes, etc.
- Looks in "standard" locations as well as current directory.

Example: require 'pp'

- Makes a function pp available.
- Similar to puts, but presents structures in a nested, easier to read format.
Miscellaneous

• Symbols
  – :foobar, :myname
  – like a string but unique, immutable, and fast
  – Often used as hash keys, identifiers, etc.

• Duck typing: "If it looks like a duck . . ."
  def putlengths anArray
    anArray.each { |x| puts x.length }
  end
  putlengths [ [1, 2, 3], "abcde", {"a" => "b", "c" => "d"} ]
ON TO THE ACTIVITY