TDP007

— Konstruktion av datorspråk —

Linköpings universitet

Ola Leifler & Peter Dalenius
{olale,petda}@ida.liu.se

Acknowledgments

Much material from these slides comes from Brian Amberg at the university of Freiburg (`http://ruby.brian-amberg.de/course/`).

Licence

Part I

Introduction

Setup at IDA

Get access to Ruby

```
$ module add prog/ruby
$ module initadd prog/ruby
```

Add ruby support to Emacs by editing or creating a file in your home directory called `.emacs`:

```
(nconc load-path '("/home/TDDB27/www-pub/materials/support/ruby-emacs"))
(require 'ruby-site)
```

The text above can be copied directly from `/home/TDDB27/www-pub/materials/support/lectures/examples/.emacs` to your own `.emacs` file.

Interpreter: Each ruby script you write should be prefixed by `#!/usr/env/bin/ruby -w`, to tell the system to search for ruby in the current environment (usually `/sw/ruby-1.8.5/bin/ruby`) when running from a terminal.

Ruby Shell: The interactive ruby shell `irb` can be used to try out parts of the code from within Emacs. Notice the Ruby menu when you open a Ruby file.

Ruby Documentation: Information about every class in ruby can be found using `ri`, the ruby interactive documentation system. It can also be accessed online from `ruby-lang.org`.

ri is ruby's fast helper

```
$ ri String#tr
------------------------------------------------------------ String#tr
     str.tr(from_str, to_str) => new_str
------------------------------------------------------------------------

     Returns a copy of str with the characters in from_str replaced by
     the corresponding characters in to_str.  If to_str is shorter than
     from_str, it is padded with its last character.  Both strings may
     use the c1–c2 notation to denote ranges of characters, and
     from_str may start with a ^, which denotes all characters except
     those listed.

        "hello".tr('aeiou', '*')    #=> "h*ll*"
        "hello".tr('^aeiou', '*')   #=> "*e**o"
        "hello".tr('el', 'ip')      #=> "hippo"
        "hello".tr('a-y', 'b-z')    #=> "ifmmp"
```

irb can be used to try out ideas

```
$ irb --simple-prompt
>> 'hal'.tr('za-y', 'ab-z')
=> "ibm"
>> class String
>>   def rot13
>>     self.tr('a-z', 'n-za-m')
>>   end
>> end
=> nil
>> a = 'geheimer text'
=> "geheimer text"
>> b = a.rot13
=> "trurvzre grkg"
>> b.rot13
=> "geheimer text"
```

Simple hello world example

This is a must

```
1 #!/usr/bin/ruby
2
3 puts 'Hello World'
```

```
1 Hello World
```

Function definition

Functions are defined using the def keyword

```
1 #!/usr/bin/ruby
2
3 def hello(programmer)
4   puts "Hello #{programmer}"
5 end
6
7 hello('Brian')
```

```
1 Hello Brian
```

In ruby everything is an object

See Chapter 3 in the course book.

Everything is an object, so get used to the "object.method" notation.

```
1  (5.6).round                                          » 6
2  (5.6).class                                          » Float
3  (5.6).round.class                                    » Fixnum
4
5  'a string'.length                                    » 8
6  'a string'.class                                     » String
7  'tim tells'.gsub('t', 'j')                           » "jim jells"
8
9  'abc'.gsub('b', 'xxx').length                        » 5
10
11 ['some', 'things', 'in', 'an', 'array'].length       » 5
12 ['some', 'things', 'in', 'an', 'array'].reverse      » ["array", "an", "in", "things", "some"]
13
14 # You can even write
15 1.+(2)                                               » 3
16
17 # but there is some sugar for cases like this
18 1 + 2                                                » 3
```

Class definition and inheritance

Base Class

```
1  class Person
2    def initialize(name)
3      @name = name
4    end
5
6    def greet
7      "Hello, my name is #{@name}."
8    end
9  end
10
11 brian = Person.new('Brian')
12 puts brian.greet
```

```
1 Hello, my name is Brian.
```

Sub Class

```
13 class Matz < Person
14   def initialize
15     super('Yukihiro Matsumoto')
16   end
17 end
18
19 puts Matz.new.greet
```

```
1 Hello, my name is Yukihiro Matsumoto.
```

All normal control structures are available

Ruby is simple to read

But if you already know some programming languages, there are sure some surprises here:

```ruby
def greet(*names)
  case names.length
  when 0
    "How sad, nobody wants to hear my talk."
  when 1
    "Hello #{names}. At least one wants to hear about ruby."
  when 2..5
    "Hello #{names.join(', ')}. Good that all of you are interested."
  when 6..12
    "#{names.length} students. Thats perfect. Welcome to ruby!"
  else
    "Wow #{names.length} students. We'll have to find a bigger room."
  end
end

puts greet('Alexander', 'Holger', 'Zyki', 'Sebastian', 'Johann', 'chenkefei',
  'JetHoeTang', 'Matthias', 'oanapop', 'Andrei', 'Phillip')
```

```
11 students. Thats perfect. Welcome to ruby!
```

Basic syntax rules

Ruby syntax tries to omit "noise"

```
1  # Functions are defined by the def keyword (define function)
2  # Function arguments can have default values.
3  def multi_foo(count = 3)
4    'foo ' * count
5  end                                          » nil
6
7  # Brackets can be omitted, if the situation is not ambiguous
8  multi_foo(3)                                 » "foo foo foo "
9  puts 'hello world'                           » nil
10
11 # Strings are written as
12 'Simple #{multi_foo(2)}'                     » "Simple \#{multi_foo(2)}"
13 "Interpolated #{multi_foo}"                  » "Interpolated foo foo foo "
14
15 # Numbers
16 10                                           » 10
17 0.5                                          » 0.5
18 2e-4                                         » 0.0002
19 0xFFFF                                       » 65535
20 010                                          » 8
```

Syntax: Variables, constants, methods, ...

Variables / methods: student, i, epsilon, last_time
Variables and methods look alike. This is reasonable because a variable can be substituted by a method.

Constants: OldPerson, PDF_KEY, R2D2
Constants can only be defined once.

Instance Variables: @name, @last_time, @maximum
Instance variables can only be accessed by the owning object.

Class Variables: @@lookup_table, @@instance

Class variables belong not to the instances but to the class They exist only once for the class, and are shared by all instances.

Global Variables: $global, $1, $count
Usage of global variables has been declared a capital crime by the school of good design.

Symbols: :name, :age, :Class
Symbols are unique identifiers, that we will encounter in various places.

Stylistic Rules

- Variables and methods should be written in snake_case
- Class Names should be written in CamelCase
- Constants should be written ALL_UPPERCASE

Exercises: Tools

irb and numbers:

Open up `irb` from Emacs with `C-c C-s` and set the variables $a = 1$, $b = 2$.

- Calculate a/b. Calculate $1.0/2.0$. Calculate 10^{200}.
- Write `require 'complex'` into irb to load the "Complex" library
  Create a constant I set to Complex.new(0, 1) and calculate $(1 + 2i) \cdot (2 + 1i)$

Array

```
1  # Literal Array
2  ['An', 'array', 'with', 5, 'entries'].join(' ')      » "An array with 5 entries"
3
4  # New Array
5  a = Array.new                                        » []
6  a << 'some' << 'things' << 'appended'                » ["some", "things", "appended"]
7  a[2]                                                 » "appended"
8  a[0] = 3                                             » 3
9  a                                                    » [3, "things", "appended"]
10
11 # Default Values can be used ...
12 Array.new(10, 0)                                     » [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
13
14 # ... but beware of the reference
15 a = Array.new(2, 'Silke')                            » ["Silke", "Silke"]
16 a[0] << ' Amberg'                                    » "Silke Amberg"
17 a                                                    » ["Silke Amberg", "Silke Amberg"]
```

Basic Containers

Arrays can be used as queues, stacks, deques or simply as arrays.

```
1  print 'Array as stack: '
2  stack = Array.new()
3  stack.push('a')
4  stack.push('b')
5  stack.push('c')
6  print stack.pop until stack.empty?
7
8  print "\n"
9  print 'Array as queue: '
10 queue = Array.new()
11 queue.push('a').push('b').push('c')
12 print queue.shift until queue.empty?
```

```
1  Array as stack: cba
2  Array as queue: abc
```

Hashes are fast associative containers

```
1  # Literal Hash
2  h0 = { 'one' => 1, 'two' => 2, 'three' => 3}   » {"three"=>3, "two"=>2, "one"=>1}
3  h0['one']                                      » 1
4
5  # Populating a hash
6  h1 = Hash.new                                  » {}
7  h1['gemstone'] = 'ruby'                        » "ruby"
8  h1['fruit'] = 'banana'                         » "banana"
9  h1                                             » {"gemstone"=>"ruby", "fruit"=>"banana"}
10
11 # Often symbols are used as keys
12 h2 = {:june => 'perl', :july => 'ruby'}        » {:june=>"perl", :july=>"ruby"}
13 h2[:july]                                      » "ruby"
14
15 # But arbitrary keys are possible
16 a = ['Array', 1]                               » ["Array", 1]
17 b = ['Array', 2]                               » ["Array", 2]
18 h3 = { a => :a1, b => :a2 }                    » {["Array", 1]=>:a1, ["Array", 2]=>:a2}
19 h3[a]                                         » :a1
```

Blocks and iterators

A function can take a block as an argument.
A block is a piece of code, which inherits the containing scope.

Using iterators

```
1 # A simple iterator, calling the block once for each entry in the array
2 ['i', 'am', 'a', 'banana'].each do | entry | print entry, ' ' end
```

```
1 i am a banana
```

```
1 # Another commonly used iterator. The block is called in the scope where it was
2 # created.
3 fac = 1                                                    » 1
4 1.upto(5) do | i | fac *= i end                            » 1
5 fac                                                        » 120
6
7 # The result of the block can be used by the caller
8 [1,2,3,4,5].map { | entry | entry * entry }                » [1, 4, 9, 16, 25]
9
10 # and more than one argument is allowed
11 (0..100).inject(0) { | result, entry | result + entry }    » 5050
```

Block Syntax

Blocks can be enclosed by do | | ... end.

```
1 [1,2,3,4,5].each do | e | puts e end
```

or by braces { | | ... }

```
1 [1,2,3,4,5].map { | e | e * e }          » [1, 4, 9, 16, 25]
```

A convention is to

- use do | | ... end wherever the side-effect is important, or there are several lines of expressions
- and braces where the return value is important, or there is only one expression.

Writing iterators

```
1 def f(count, &block)
2   value = 1
3   1.upto(count) do | i |
4     value = value * i
5     block.call(i, value)
6   end
7 end
8
9 f(5) do | i, f_i | puts "f(#{i}) = #{f_i}" end
```

Writing iterators

```ruby
def f(count, &block)
  value = 1
  1.upto(count) do | i |
    value = value * i
    block.call(i, value)
  end
end

f(5) do | i, f_i | puts "f(#{i}) = #{f_i}" end
```

```
f(1) = 1
f(2) = 2
f(3) = 6
f(4) = 24
f(5) = 120
```

Saving the block

```
1  class Repeater
2    def initialize(&block)
3      @block = block
4      @count = 0
5    end
6
7    def repeat
8      @count += 1
9      @block.call(@count)
10   end
11 end
12
13 repeater = Repeater.new do | count | puts "You called me #{count} times" end
14 3.times do repeater.repeat end
```

Saving the block

```ruby
class Repeater
  def initialize(&block)
    @block = block
    @count = 0
  end

  def repeat
    @count += 1
    @block.call(@count)
  end
end

repeater = Repeater.new do | count | puts "You called me #{count} times" end
3.times do repeater.repeat end
```

```
You called me 1 times
You called me 2 times
You called me 3 times
```

Exercises: Iterators

Refer to the exercise files for exact specification of the problems.

n_times

Write an iterator function `n_times(n)` that calls the given block n times. Write an iterator class `Repeat` that is instantiated with a number and has a method `each` that takes a block and calls it as often as declared when creating the object.

Faculty

Write a one-liner in irb using `Range#inject` to calculate 20!. Generalize this into a function.

Maximum

Write a function to find the longest string in an array of strings.

find_it

Write a function `find_it` that takes an array of strings and a block. The block should take two parameters and return a boolean value.
The function should allow to implement `longest_string`, `shortest_string`, and other functions by changing the block.

Ruby assignments.

```
1  # Every assignment returns the assigned value
2  a = 4                                          » 4
3
4  # So assignments can be chained
5  a = b = 4                                      » 4
6  a + b                                          » 8
7
8  # and used in a test
9  file = File.open('../lect1_2.tex')             » #<File:../lect1_2.tex>
10 linecount = 0                                  » 0
11 linecount += 1 while (line = file.gets)        » nil
12
13 # Shortcuts
14 a += 2                                         » 6
15 a = a + 2                                      » 8
16 #...
17
18 # Parallel assignment
19 a, b = b, a                                    » [4, 8]
20
21 # Array splitting
22 array = [1, 2]                                 » [1, 2]
23 a, b = *array                                  » [1, 2]
```

Control Structures - Conditionals

Ruby has all standard control structures.
And you may even write them to the right of an expression.

```
1  if (1 + 1 == 2)
2    "Like in school."
3  else
4    "What a surprise!"
5  end                                          » "Like in school."
6
7  "Like in school." if (1 + 1 == 2)            » "Like in school."
8  "Surprising!" unless (1 + 1 == 2)            » nil
9
10 (1 + 1 == 2) ? 'Working' : 'Defect'          » "Working"
11
12 spam_probability = rand(100)                 » 0
13 case spam_probability
14 when 0...10 then "Lowest probability"
15 when 10...50 then "Low probability"
16 when 50...90 then "High probability"
17 when 90...100 then "Highest probability"
18 end                                          » "Lowest probability"
```

Control Structures - True and False

Only *nil* and *false* are false, everything else is true.

```
1  def is_true(value)
2    value ? true : false
3  end                                  » nil
4
5  is_true(false)                       » false
6  is_true(nil)                         » false
7  is_true(true)                        » true
8  is_true(1)                           » true
9  is_true(0)                           » true
10 is_true([0,1,2])                     » true
11 is_true('a'..'z')                    » true
12 is_true('')                          » true
13 is_true(:a_symbol)                   » true
```

Join the equal rights for zero movement!

Ruby has a variety of loop constructs, but don't forget the blocks!

```
1  i = 1                                  » 1
2
3  while (i < 10)
4    i *= 2
5  end                                    » nil
6  i                                      » 16
7
8  i *= 2 while (i < 100)                 » nil
9  i                                      » 128
10
11 begin
12   i *= 2
13 end while (i < 100)                    » nil
14 i                                      » 256
15
16 i *= 2 until (i >= 1000)               » nil
17 i                                      » 1024
18
```

```
19 loop do
20   break i if (i >= 4000)
21   i *= 2
22 end                                    » 4096
23 i                                      » 4096
24
25 4.times do i *= 2 end                  » 4
26 i                                      » 65536
27
28 r = []                                 » []
29 for i in 0..7
30   next if i % 2 == 0
31   r << i
32 end                                    » 0..7
33 r                                      » [1, 3, 5, 7]
34
35 # Many things are easier with blocks:
36 (0..7).select { |i| i % 2 != 0 }       » [1, 3, 5, 7]
```

Exercises: Control Structures

Fibonacci

Write functions that calculate the fibonacci numbers using different looping constructs

$$fib(i) = \begin{cases} 0 & i = 0 \\ 1 & i = 1 \\ fib(i-1) + fib(i-2) & otherwise \end{cases}$$

while: Implement the function using a while loop.

for: Implement the function using a for loop.

times: Implement the function using the times construct.

loop: Implement the function using the loop construct.

Iterator

Write a fibonacci iterator function.
That is a function that takes a number n and a block and calls the block with $fib(0), fib(1), \ldots fib(n)$

Generator

Write a fibonacci generator class.
That is: A class that has a next function which on each call returns the next fibonacci number.

Part II

The dynamicity of ruby

Accessor Functions: Getting object properties

```
1  class Cell
2    def initialize
3      @state = :empty
4    end
5  end
```

```
7  class Board
8    def initialize(width, height)
9      @width = width; @height = height
10     @cells = Array.new(height) { Array.new(width) { Cell.new } }
11   end
12 end
```

Access a property

```
14 class Cell
15   def state
16     @state
17   end
18 end
```

```
20 cell = Cell.new     » #<Cell:... @state=:e...>
21 cell.state          » :empty
```

Calculated property

```
50 class Board
51   def size
52     self.width * self.height
53   end
54 end
```

Shortcut

```
34 class Cell
35   attr_reader :state
36 end
```

Accessor Functions: Setting object properties

```
1  class Cell
2    def initialize
3      @state = :empty
4    end
5  end
```

```
7  class Board
8    def initialize(width, height)
9      @width = width; @height = height
10     @cells = Array.new(height) { Array.new(width) { Cell.new } }
11   end
12 end
```

Set a property

```
23 class Cell
24   def state=(state)
25     @state = state
26   end
27 end
```

```
29 cell = Cell.new          » #<Cell:... @state=:e...>
30 cell.state               » :empty
31 cell.state = :king       » :king
32 cell.state               » :king
```

Shortcut

```
38 class Cell
39   attr_writer :state
40 end
```

Shortcut for getter and setter

```
42 class Cell
43   attr_accessor :state
44 end
```

Accessor Functions - Array-like accessors

```
1  class Cell
2    def initialize
3      @state = :empty
4    end
5  end
```

```
7  class Board
8    def initialize(width, height)
9      @width = width; @height = height
10     @cells = Array.new(height) { Array.new(width) { Cell.new } }
11   end
12 end
```

The method "[]" can be used to implement an array-like accessor.

```
56 class Board
57   def [](col, row)
58     @cells[col][row]
59   end
60 end
```

```
68 board = Board.new(8, 8)    » #<Board:... @height=8...>
69 board[0, 0]                » #<Cell:... @state=:e...>
70 board[0, 0] = Cell.new()   » #<Cell:... @state=:e...>
```

The method "[]=" can be used as an array-like setter.

```
62 class Board
63   def []=(col, row, cell)
64     @cells[col][row] = cell
65   end
66 end
```

```
68 board = Board.new(8, 8)    » #<Board:... @height=8...>
69 board[0, 0]                » #<Cell:... @state=:e...>
70 board[0, 0] = Cell.new()   » #<Cell:... @state=:e...>
71 board[0, 0].state = :tower » :tower
72 board[0, 0].state          » :tower
```

Exercise: Accessor Functions

PersonName

Create a class PersonName, that has the following attributes

Name	The name of the person.
Surname	The given name of the person.
Fullname	"#{surname} #{name}". Add also a fullname setter function, that splits (String::split) the fullname into surname and name.

Person

Create a class Person, that has the following attributes

Age	The person's age (in years).
Birthdate	The person's birthdate.
Name	A PersonName object.

- The person's constructor should allow to pass in name, surname and age. All optionally.
- The person's age and birth date should always be consistent. That means if I set the person's birth date, his age should change. And if I set a person's age, his birth date should change.

Ruby is Dynamic

Classes, functions, modules can be modified at runtime.

```
25 class PersonShort < BasePerson
26   attr_accessor :name, :surname
27 end
```

attr_accessor is not a special language construct, but a function, that creates getter and setter functions for each argument.

Ruby is Dynamic

You can extend existing classes

```ruby
class Integer
  def fac
    raise "Faculty undefined for #{self}" if self < 0
    return (1..self).inject(1) { |result, i| result * i }
  end
end

puts (0..13).map { |i| i.fac }.join(', ')
```

```
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800
```

Exercises: Extension of existing classes

Fibonacci II

Extend Integer with a function fib that calculates the corresponding fibonacci number.

Shuffle

Extend Array with a method shuffle that creates a random permutation of the elements in the array.

```
1 [0,1,2,3,4,5].shuffle      » [0, 3, 4, 5, 1, 2]
2 [0,1,2,3,4,5].shuffle      » [4, 1, 0, 5, 3, 2]
3 [0,1,2,3,4,5].shuffle      » [3, 4, 2, 1, 0, 5]
```

Set

Extend Array with the set methods union and intersect. E.g.:

```
1  a1 = [1, 2, 3]
2  a2 = [2, 3, 4]
3  a3 = [{:c => 'a', :v => 1}, {:c => 'b', :v => 2}]
4  a4 = [{:c => 'b', :v => 2}, {:c => 'c', :v => 3}]
5  a1.intersect(a2)                » [2, 3]
6  a2.intersect(a1)                » [2, 3]
7  a1.intersect(a3)                » []
8  a3.intersect(a4)                » [{:v=>2, :c=>"b"}]
9  a1.union(a2).union(a3)          » [1, 2, 3, 2, 3, 4, {:v=>1, :c=>"a"}, {:v=>2, :c=>"b"}]
10 a1.intersect(a1.union(a2))      » [1, 2, 3]
```

Modules provide namespaces

```ruby
module AntGame
  class Ant
    attr_accessor :x, :y, :direction, :next_action

    def initialize(x, y)
      @x = x; @y = y
      @direction = :north
      @next_action = Actions::WAIT
    end
  end

  module Actions
    WAIT = :wait
    TURN_LEFT = :turn_left
    TURN_RIGHT = :turn_right
    GO = :go
  end
end

AntGame::Ant.new(4, 5)
include AntGame
Ant.new(1, 2)
```

Modules provide controlled multiple inheritance

```ruby
module Observable
  def register(event=nil, &callback)
    @observers ||= Hash.new
    @observers[event] ||= []
    @observers[event] << callback
    self
  end

  protected
  def signal_event(event = nil, *args)
    @observers ||= Hash.new
    @observers[event] ||= []
    @observers[event].each do | callback |
      callback.call(self, *args)
    end
  end
end
```

```ruby
class Observed
  include Observable

  def foo=(a_foo)
    signal_event(:changed, @foo, a_foo)
    @foo = a_foo
  end
end

observed = Observed.new
observed.register(:changed) do | o, old, new |
  puts "#{old} -> #{new}"
end

observed.foo = 'Yukihiro'
observed.foo = 'Yukihiro Matsumoto'
observed.foo = 'Matz'
```

```
 -> Yukihiro
Yukihiro -> Yukihiro Matsumoto
Yukihiro Matsumoto -> Matz
```

Modules provide controlled multiple inheritance

```
1  module Enumerable
2    def call_on_each(method)
3      each do | item | item.send(method) end
4      self
5    end
6  end                                                    » nil
7
8  class Array
9    include Enumerable
10 end                                                    » Array
11
12 ['i', 'am', 'a', 'banana'].call_on_each(:reverse!)     » ["i", "ma", "a", "ananab"]
```

Modules provide controlled multiple inheritance

```
1     self
2   end
3 end                                                    » nil
4
5 class Sentence
6   include Enumerable
7
8   attr_accessor :subject, :predicate, :object
9
10  def initialize(subject, predicate, object)
11    self.subject, self.predicate, self.object = subject, predicate, object
12  end
13
14  def each
15    yield self.subject
16    yield self.predicate
17    yield self.object
18  end
19
20  def to_s() "#{self.subject} #{self.predicate} #{self.object}" end
21 end                                                    » nil
22
23 sentence = Sentence.new('i', 'am', 'banana')          » #<Sentence:0x1096414 @subject="i", @obj
24 sentence.to_s                                         » "i am banana"
```

Tree

Create a class TreeItem that has the following attributes:

- item That contains the list item used.
- left The left child of this item.
- right The right child of this item.
- each A function that takes a block and calls the block for each item in the subtree.

Include the module `Enumerable` into the tree item. E.g.

```
1  root = TreeItem.new("root")                  » #<TreeItem:0x10b6124 @item="root">
2  root.to_a.join(' | ')                        » "root"
3  root.left = TreeItem.new("left")             » #<TreeItem:0x10b2380 @item="left">
4  root.to_a.join(' | ')                        » "root | left"
5  root.right = TreeItem.new("right")           » #<TreeItem:0x10ae4ec @item="right">
6  root.to_a.join(' | ')                        » "root | left | right"
7  root.left.left = TreeItem.new("left-left")   » #<TreeItem:0x10a9f00 @item="left-left">
8  root.to_a.join(' | ')                        » "root | left | left-left | right"
9  root.left.right = TreeItem.new("left-right") » #<TreeItem:0x10a5810 @item="left-right">
10 root.to_a.join(' | ')                        » "root | left | left-left | left-right | right"
11 root.inject(0) { | r, e | r + 1 }            » 5
```

Example Implementation

```
1  class TreeItem
2    attr_accessor :left, :right, :item
3    include Enumerable
4
5    def initialize(item)
6      self.item = item
7    end
8
9    def each(&block)
10     block.call(self.item)
11     left.each(&block) if left
12     right.each(&block) if right
13   end
14 end                                  » nil
15
16 root = TreeItem.new("root")          » #<TreeItem:0x10b6124 @item="root">
17 root.to_a.join(' | ')                » "root"
18 root.left = TreeItem.new("left")     » #<TreeItem:0x10b2380 @item="left">
19 root.to_a.join(' | ')                » "root | left"
20 root.right = TreeItem.new("right")   » #<TreeItem:0x10ae4ec @item="right">
21 root.to_a.join(' | ')                » "root | left | right"
22 root.left.left = TreeItem.new("left-left") » #<TreeItem:0x10a9f00 @item="left-left">
23 root.to_a.join(' | ')                » "root | left | left-left | right"
```

List

Create a class `ListItem` that has the following attributes/methods:

- **item** That contains the list item used.
- **previous** The predecessor in the list. When this property is set the old and new predecessor's `next` property should be updated.
- **next** The successor in the list. When this property is set the old and new successor's `previous` should be updated.
- **each** Takes a block and calls the block for each item in the list. This should be done by following `previous` to the beginning of the list and then returning each item in list order.
- **insert** Inserts an item after this item into the list.

Include the module `Enumerable` into the list item, such that the following constructs work. E.g.

```
1 one = ListItem.new("one")                          » #<ListItem:... @item="on...>
2 one.next = ListItem.new("two")                     » #<ListItem:... @previous...>
3 one.next.next = ListItem.new("three")              » #<ListItem:... @previous...>
4 one.previous = ListItem.new("zero")                » #<ListItem:... @next=#<L...>
5 one.inject('List:') { | r, v | r + ' ' + v }       » "List: zero one two three"
6
7 one.insert ListItem.new("one point five")          » #<ListItem:... @previous...>
8 one.inject('List:') { | r, v | r + ' ' + v }       » "List: zero one one point five two three"
```

Example Implementation

```
1  require 'shortest_inspect'                    » true
2
3  class ListItem
4    attr_reader :next, :previous
5    attr_accessor :item
6    include Enumerable
7
8    def initialize(item)
9      self.item = item
10   end
11
12   def next=(other)
13     return if self.next == other
14     self.next.previous = nil if self.next and (self.next.previous != self)
15     @next = other
16     self.next.previous = self if other
17     self
18   end
19
20   def previous=(other)
21     return if self.previous == other
22     self.previous.next = nil if self.previous and (self.previous.next != self)
23     @previous = other
24     self.previous.next = self if other
```

List

Create a mixin ListItem that extends any class with the following attributes:

previous_item	The predecessor in the list. When this property is set the old and new predecessor's next_item property should be updated.
next_item	The successor in the list. When this property is set the old and new successor previous_item property should be updated.
each_item	Returns each item that is in the list. This should be done by following previous_item to the beginning of the list and then returning each item in list order.
list_to_array	Returns the list as an array.

E.g.

```
1  end                                          » nil
2
3  one = ListItem.new("one")                    » #<ListItem:... @item="on...>
4  one.next = ListItem.new("two")               » #<ListItem:... @previous...>
5  one.next.next = ListItem.new("three")        » #<ListItem:... @previous...>
6  one.previous = ListItem.new("zero")          » #<ListItem:... @next=#<L...>
7  one.inject('List:') { | r, v | r + ' ' + v } » "List: zero one two three"
8
9  one.insert ListItem.new("one point five") » #<ListItem:... @previous...>
10 one.inject('List:') { | r, v | r + ' ' + v } » "List: zero one one point five two three"
```

Part III

Regular Expressions

Regular Expressions

- Any character except `\/^$|.+*?()[]\{\}`, matches itself.
- `^` matches the start of a line, `$` matches the end of a line.
- `.` matches any character.
- If `a`, `b` are regular expressions, then:
  - `ab` is also a regular expression, that matches the concatenated strings.
  - `a*` is a regular expression matching the hull of `a`.
  - `a+` is equivalent to `aa*`.
  - `a|b` matches either `a` or `b`.
  - Expressions can be grouped by brackets. E.g: `(a|b)c` matches $\{`ac', `bc'\}$, `a|bc` matches $\{`a', `bc'\}$.
- `[characters]` Matches a range of characters. Example: `[a-zA-Z0-9]` matches the alphanumeric characters.
- `[^characters]` Matches the negation of a range of characters. Example: `[^a-zA-Z0-9]` matches all non-alphanumeric characters.
- `+`, and `*` are greedy, `+?`, `*?` are the non-greedy versions .
- `(?=regexp)` and `(?!regexp)` is positive and negative lookahead.
- There exist a couple of shortcuts for character classes. E.g. `\w` = `[0-9A-Za-z_]`, `\W` = `[^0-9A-Za-z_]`, `\s` = `[ \t\n\r\f]`, `\S` = `[^ \t\n\r\f]`,

More information can be found at: http://www.regular-expressions.info/tutorial.html

Examples

```
1  # Simple regexps
2  /ruby/ =~ 'perls and rubys'                  » 10
3  /ruby/ =~ 'complicated'                      » nil
4  /b(an)*a/ =~ 'ba'                            » 0
5  /b(an)*a/ =~ 'some bananas'                  » 5
6  /^b(an)*a/ =~ 'some bananas'                 » nil
7  /[tj]im/ =~ 'tim'                            » 0
8  /[tj]im/ =~ 'jim'                            » 0
9  /[tj]im/ =~ 'vim'                            » nil
10
11 # Extracting matches
12 /(.*) (.*)/ =~ 'thats ruby'                  » 0
13 [$1, $2]                                     » ["thats", "ruby"]
14
15 # The OO way
16 re = /name: "(.*)"/                         » /name: "(.*)"/
17 mr = re.match('name: "brian"')              » #<MatchData:0x10ba094>
18 mr[1]                                       » "brian"
```

Some functions

```
20 def showRE(string, regexp)
21   if regexp =~ string then "#{$`}<#{$&}>#{$'}" else "no match" end
22 end                                                    » nil
23
24 a = "The moon is made of cheese"                      » "The moon is made of cheese"
25 showRE(a, /\w+/)                                      » "<The> moon is made of cheese"
26 showRE(a, /\s.*\s/)                                   » "The< moon is made of >cheese"
27 showRE(a, /\s.*?\s/)                                  » "The< moon >is made of cheese"
28 showRE(a, /[aeiou]{2,99}/)                            » "The m<oo>n is made of cheese"
29 showRE(a, /mo?o/)                                     » "The <moo>n is made of cheese"
30
31 a = "rubys     are  brilliant  \t gemstones"          » "rubys     are  brilliant  \t gemstones"
32 a.gsub(/[aeiou]/, '*')                                » "r*bys     *r*  br*ll**nt  \t g*mst*n*s"
33 a.gsub!(/\s+/, ' ')                                   » "rubys are brilliant gemstones"
34 a.gsub(/(^|\s)\w/) { | match | match.upcase }         » "Rubys Are Brilliant Gemstones"
35 a.split(/ /)                                          » ["rubys", "are", "brilliant", "gemstones"]
36 a.scan(/[aeiou][^aeiou]/)                             » ["ub", "ar", "e ", "il", "an", "em", "on", "es"]
37 a.scan(/[aeiou](?=[^aeiou ])|
38         [^aeiou ](?=[aeiou])/x).length                » 14
39
40 File.open('/usr/share/dict/words') { | words |
41   words.select { | word | /a.*e.*i.*o.*u/ =~ word }
42 }[0..2].map{ | word | word.strip }                     » ["abietineous", "abstemious", "abstemiously"]
```

Exercises: Regular Expressions

Simple Match

Write a regular expression that matches lines, that begin with the string “USERNAME:”.

Character Classes

Write a function that extracts the tag names from a html document. E.g.

```
1 require 'open-uri.rb'                                    » true
2 html = open("http://www.google.de/") { | f | f.read } » "<html><head><meta http-equiv=\"
3 tag_names(html)                                          » ["html", "head", "meta", "title", "style
```

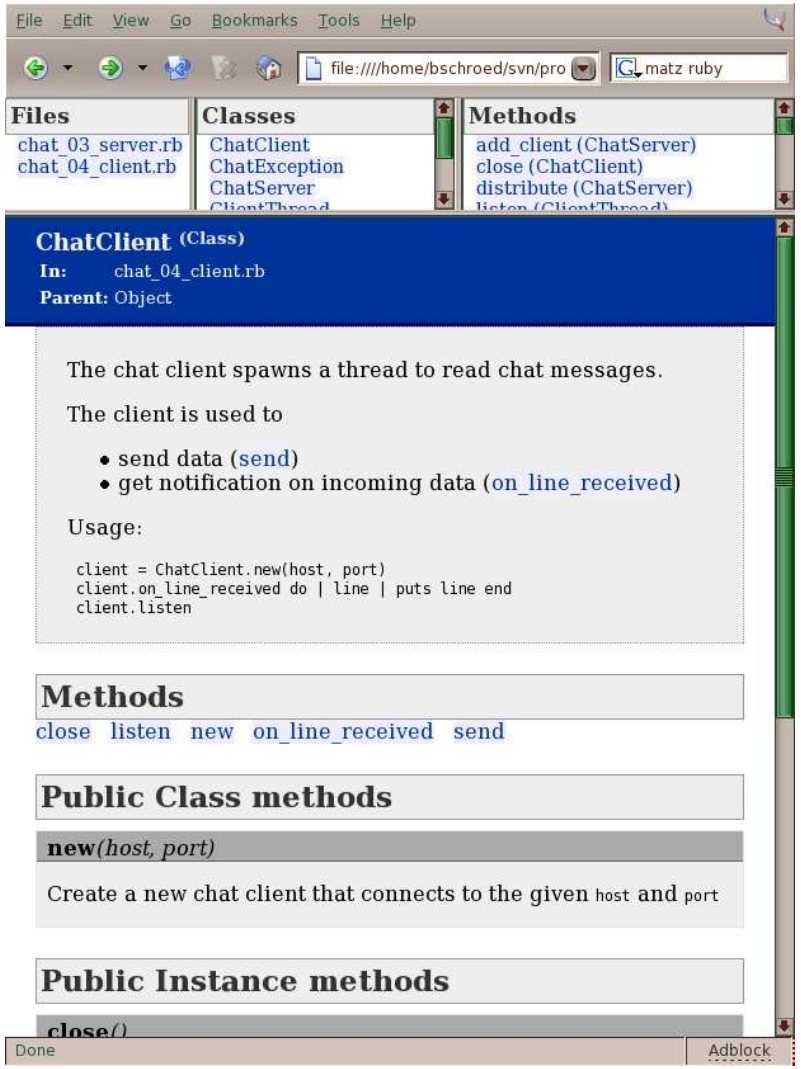
Extract Username

Write a regular expression that extracts the username from a string of the form “USERNAME: Brian”.

Documentation

The standard for documenting ruby programs is `rdoc`. From rdoc documentation the `ri` documentation and the standard library documentation is created. `rdoc` uses a wiki-like unobtrusive markup. E.g.

```
14  # The chat client spawns a thread that
15  # receives incoming chat messages.
16  #
17  # The client is used to
18  # * send data (#send)
19  # * get notification on incoming data
20  #   (#on_line_received)
21  #
22  # Usage:
23  #  client = ChatClient.new(host, port)
24  #  client.on_line_received do | line | puts line end
25  #  client.listen
26  class ChatClient
27
28    # Create a new chat client that connects to the
29    # given +host+ and +port+
30    def initialize(host, port)
31      @socket = TCPSocket.new(host, port)
32      @on_receive = nil
```



Unit Testing

- Unit tests are small programs, that compare the behaviour of your program against specified behaviour.
- Unit tests are collected while developing an application/library.
- Unit tests save you from breaking something with one change which you did not take into account when applying the change.

Example

```ruby
#!/usr/bin/ruby -w

require 'faculty_1'
require 'test/unit'

class TC_Faculty < Test::Unit::TestCase

  @@faculties = [[0, 1], [1, 1], [2, 2], [3, 6], [4, 24], [6, 720], [13, 6227020800]]

  def test_faculty
    @@faculties.each do | i, i_fac |
      assert_equal(i_fac, i.fac,"#{i}.fac returned wrong value.")
    end
  end
end
```

Unit Testing - Examples

Library

```
1  class Integer
2    def fac
3      (1..self).inject(1) { | r, v | r * v }
4    end
5  end
```

Test

```
10   def test_faculty
11     @@faculties.each do | i, i_fac |
12       assert_equal(i_fac, i.fac,"#{i}.fac returned wrong value.")
13     end
14   end
```

Result of Testsuite

```
1  Loaded suite faculty_1_test_1
2  Started
3  .
4  Finished in 0.028897 seconds.
5
6  1 tests, 7 assertions, 0 failures, 0 errors
```

Unit Testing - Examples

Test

```ruby
#!/usr/bin/ruby -w

require 'faculty_1'
require 'test/unit'

class TC_Faculty < Test::Unit::TestCase

  @@faculties = [[0, 1], [1, 1], [2, 2], [3, 6], [4, 24], [6, 720], [13, 6227020800]]

  def test_faculty
    @@faculties.each do | i, i_fac |
      assert_equal(i_fac, i.fac, "#{i}.fac returned wrong value.")
    end
  end

  def test_negative
    assert_raise(ENegativeNumber, '-1! should raise exception') do -1.fac end
    assert_raise(ENegativeNumber, '-10! should raise exception') do -10.fac end
    assert_raise(ENegativeNumber, '-111! should raise exception') do -111.fac end
  end
end
```

Unit Testing - Examples

Test

```
16  def test_negative
17    assert_raise(ENegativeNumber, '-1! should raise exception') do -1.fac end
18    assert_raise(ENegativeNumber, '-10! should raise exception') do -10.fac end
19    assert_raise(ENegativeNumber, '-111! should raise exception') do -111.fac end
20  end
```

Result of Testsuite

```
1  Loaded suite faculty_2_test_1
2  Started
3  .E
4  Finished in 0.039849 seconds.
5
6    1) Error:
7  test_negative(TC_Faculty):
8  NameError: uninitialized constant TC_Faculty::ENegativeNumber
9        faculty_2_test_1.rb:17:in `test_negative'
10
11 2 tests, 7 assertions, 0 failures, 1 errors
```

Unit Testing - Examples

Library

```
1 class ENegativeNumber < Exception; end
2
3 class Integer
4   def fac
5     raise ENegativeNumber if self < 0
6     (1..self).inject(1) { | r, v | r * v }
7   end
8 end
```

Unit Testing - Examples

Test

```
16  def test_negative
17    assert_raise(ENegativeNumber, '-1! should raise exception') do -1.fac end
18    assert_raise(ENegativeNumber, '-10! should raise exception') do -10.fac end
19    assert_raise(ENegativeNumber, '-111! should raise exception') do -111.fac end
20  end
```

Result of Testsuite

```
1 Loaded suite faculty_2_test_2
2 Started
3 ..
4 Finished in 0.011961 seconds.
5
6 2 tests, 10 assertions, 0 failures, 0 errors
```

Literature

James Britt:
The ruby-doc.org ruby documentation project.
http://www.ruby-doc.org/

Chad Fowler:
Ruby Garden Wiki.
http://www.rubygarden.org/ruby/

ruby-lang.org editors <www-admin@ruby-lang.org>:
The Ruby Language.
http://www.ruby-lang.org/

Dave Thomas:
RDOC - Ruby Documentation System.
http://www.ruby-doc.org/stdlib/libdoc/rdoc/rdoc/index.html

Dave Thomas, Chad Fowler, and Andy Hunt:
Programming Ruby - The Pragmatic Programmer's Guide.
Addison Wesley Longman, Inc, 1st edition, 2001.
http://www.ruby-doc.org/docs/ProgrammingRuby/

Dave Thomas, Chad Fowler, and Andy Hunt:
Programming Ruby - The Pragmatic Programmer's Guide.
Addison Wesley Longman, Inc, 2nd edition, 2004.

GNU Free Documentation License

The GNU Free Documentation License as applicable to this document can be found at:
`http://www.gnu.org/copyleft/fdl.html`