A First Look At Ruby

```
#!/usr/bin/ruby
puts "Hello Class"
```
History

• 1993, February 24: Yukihiro Matsumoto ("Matz") started to work on Ruby.
• 1993, Summer: First "Hello, world!" program works.
• 1995, December: First release 0.95.
• 1996, December: 1.0 is released.
• 1999: Supposedly overtakes Python in Japan.
• 2000: The first official newsgroup.
• 2000-2001: Several books and magazine articles published.
• 2003, August 4: 1.8.0 is released.

The language was created by Yukihiro “Matz” Matsumoto, who started working on Ruby on February 24, 1993, and released it to the public in 1995.

"Ruby" was named after a colleague's birthstone. As of March 2006, the latest stable version is 1.8.4. Ruby 1.9 (with some major changes) is also in development.
Ruby versus …

Perl

Ruby is very Perl inspired.

With some fantasy, Ruby may be seen as an alternative evolution of Perl.

O'Reilly Interview: What bits of Perl did you incorporate in Ruby?

Matz: A lot. Ruby's class library is an object-oriented reorganization of Perl functionality--plus some Smalltalk and Lisp stuff. I used too much, I guess. I shouldn't have inherited $_, $&, and the other, ugly style variables.

Python

Both Ruby and Python were created as alternatives to Perl.

However, Python is more TIOOWTDI and Ruby is more TIMTOWTDI.

O'Reilly Interview: How about Python? What aspects of that language did you try to reuse in Ruby?

Matz: Far less than Perl. But I stole a few things, like exception names. Plus I learned a lot from its code.
Numbers

• Literals
  _ used for readability
  e# used for magnitude shift
  . must be followed by #

• Integer
  Two types - automatic conversion
  • Fixnum - native machine word minus 1 bit
  • Bignum - arbitrarily large

• Floating Point
  Two types
  • Decimals
  • Exponents
Strings

• Literals
  ‘ ‘ = %q/ / Only \ and \’ escaped
  “ “ = %Q/ / , %/ More escapes

• Support For
  – Concatenation of adjacent literals

  – Support for literals across linebreaks
## Ranges

<table>
<thead>
<tr>
<th>Ruby</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>x..y</td>
<td>xrange(x, y+1)</td>
</tr>
<tr>
<td>x…y</td>
<td>xrange(x,y)</td>
</tr>
</tbody>
</table>
Variables

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.
#Literal Hash
h0 = { 'one' => 1, 'two' => 2, 'three' => 3 } => { "three" => 3, "two" => 2, "one" => 1 }
  h0['one']

#Populating a hash
h1 = Hash.new
h1['gemstone'] = 'ruby'
h1['fruit'] = 'banana'
h1

# Often symbols are used as keys
h2 = { :june => 'perl', :july => 'ruby' } => { :july => 'ruby', :june => 'perl' }
  h2[:july]

# But arbitrary keys are possible
a = ['Array', 1]
b = ['Array', 2]
h3 = { a => :a1, b => :a2 } => { ["Array", 1] => :a1, ["Array", 2] => :a2 }
  h3[a]
## Blocks

**Block Syntax**

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

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

or by braces `{ | ... }`.

```ruby
[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
- and braces where the return value is important.
Assignments

```ruby
# Every assignment returns the assigned value
a = 4           » 4

# So assignments can be chained
a = b = 4        » 4
a + b            » 8

# and used in a test
file = File.open('..../slides.tex')  » #<File:../slides.tex>
linecount = 0    » 0
linecount += 1 while (line = file.gets) » nil

# Shortcuts
a += 2           » 6
a = a + 2        » 8
#

# Parallel assignment
a, b = b, a      » [4, 8]

# Array splitting
array = [1, 2]   » [1, 2]
a, b = *array    » [1, 2]
```
True and False

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

```ruby
def is_true(value)
  value ? true : false
end

is_true(false)   » false
is_true(nil)     » false
is_true(true)    » true
is_true(1)       » true
is_true(0)       » true
is_true([0,1,2]) » true
is_true('a'..'z')» true
is_true('')      » true
is_true(:a_symbol) » true
```

Join the equal rights for zero movement!
if (1 + 1 == 2)
  "Like in school."
else
  "What a surprise!"
end

"Like in school." if (1 + 1 == 2)
"Surprising!" unless (1 + 1 == 2)

(1 + 1 == 2) ? 'Working' : 'Defect'

spam_probability = rand(100)
case spam_probability
  when 0...10 then "Lowest probability"
  when 10...50 then "Low probability"
  when 50...90 then "High probability"
  when 90...100 then "Highest probability"
end

» "Like in school."
» "Like in school."
» nil
» "Working"
» 64
» "High probability"
Assignments

# Every assignment returns the assigned value
a = 4   → 4

# So assignments can be chained
a = b = 4 → 4
a + b    → 8

# and used in a test
file = File.open('../slides.tex') → #<File:'../slides.tex'>
linecount = 0 → 0
linecount += 1 while (line = file.gets) → nil

# Shortcuts
a += 2 → 6
a = a + 2 → 8
###

# Parallel assignment
a, b = b, a → [4, 8]

# Array splitting
array = [1, 2] → [1, 2]
a, b = *array → [1, 2]
Loops

```plaintext
i = 1          » 1
while (i < 10)
i *= 2
end          » nil
i           » 16

i *= 2 while (i < 100) » nil
i           » 128

begin
   i *= 2
end while (i < 100) » nil
i           » 256

i *= 2 until (i >= 1000) » nil
i           » 1024
```

```plaintext
loop do
   break i if (i >= 4000)
i *= 2
end                       » 4096
i                           » 4096

4.times do i *= 2 end       » 4
i                           » 65536

r = []                      » []
for i in 0..7
   next if i % 2 == 0
   r << i
end                          » 0..7
r                               » [1, 3, 5, 7]

# Many things are easier with blocks:
(0..7).select { |i| i % 2 != 0 } » [1, 3, 5, 7]
```
In Ruby, everything is an object.

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

```
(5.6).round           » 6
(5.6).class           » Float
(5.6).round.class     » Fixnum

'a string'.length    » 8
'a string'.class     » String
'tim tells'.gsub('t', 'j') » "jim jells"

'abc'.gsub('b', 'xxx').length » 5

['some', 'things', 'in', 'an', 'array'].length » 5
['some', 'things', 'in', 'an', 'array'].reverse » ['array", "an", "in", "things", "some"]

Float.class           » Class
Class.class            » Class
Object.class           » Class
```
Class definition and inheritance

Base Class

```ruby
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
```

Sub Class

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

Hello, my name is Yukihiro Matsumoto.
class Song
  def initialize(name, artist, duration)
    @name      = name
    @artist    = artist
    @duration  = duration
  end
end

aSong = Song.new("Bicylops", "Fleck", 260)
aSong.inspect  # "<Song:0x401b4924 @duration=260, @artist="Fleck", @name="Bicylops">"
Ruby Object Model (vs Python)

- Philosophy:
  - Ruby is OO made to look like a scripting language
  - Python is a scripting language with great OO support

- Ruby OO is “in your face”. It is ubiquitous and the central focus of the language
- Python OO is hidden so you only have to use it as much as you want to
Everything is an Object

• This is true of both Python and Ruby, but it is done differently
  – In Python there are functions and methods are a special case of functions
  – In Ruby there are only methods, or better yet, messages
    • The OO enthusiast thinks of method calls as messages sent to an object, rather than a choice of a certain object functionality.