# Ruby
for programmers

elegant yet powerful

Fiodar Zboichyk - zboichyk@us.ibm.com
IBM jStart - www.ibm.com/jstart

State of the Ruby

‘93	Yukihiro “matz” Matsumoto wanted a language more powerful than Perl and more OO than Python
Xmas ‘96	1.0 released.
’99	Ruby 1.3 made it out of Japan
’03	Ruby 1.8 released.
’05	Ruby on Rails (RoR) gets out, everyone goes crazy about Ruby.
Xmas ‘07	Ruby 1.9 released. It is NOT [fully] compatible with 1.8. In addition to an interpreter, gets a VM which is much faster. Migration from 1.8 to 1.9 is v e r y s l o w .
Feb ’13	Ruby 2.0 is released. Intends to be backward compatible with 1.9
June ’13	1.8 End Of Life. Migrate to 1.9 NOW! Or even 2.0 if you can.

TOC

Basic syntax

- Less brackets
- Optional termination
- Dynamic (no types)
    - *nil* is “no value”
    - *true* / *false* / *TRUE* / *FALSE*
- CONSTANTS are uppercase
- EVERYTHING is an object

```ruby
# Comment
MSG = "Hello world!" ;  # you can terminate with ';'
MSG = "Hello there!"    # but don't have to
                        # warning: already initialized
a = 4 + 5
    -3
puts a                  # 9, not 6. use '\' to escape
NL

if ( MSG )
    puts( MSG )         # with brackets
end
if MSG then
    puts MSG            # and without
end

p MSG.class             # String
p "A string".class      # String
p 2.class               # Fixnum
p true.class            # TrueClass
p nil.class             # NilClass

emptyString = ""
if emptyString
  puts "Empty string is logical TRUE"
end

unless nil
  puts "nil is the only logical FALSE"
end
```

Data types

- No types
- String
- Numeric
  - Integer
    - Fixnum
    - Bignum
  - Float
- TrueClass
- FalseClass
- NilClass
- Symbol
- Array

```ruby
v = 2*3;
v.class             # Fixnum
v /= 12.0
v.class             # Float
abs(-5)             # !! undefined method
-5.abs              # 5
r = Rational(1,3)
r2 = Rational("2/3")
s = "v = #{v}";       # "v = 0.5"
s = "v = %3.2f" % v   # "v = 0.50"
s = "-" * 10;         # "----------"

:hello.class          # Symbol
"hello".intern.class  # Symbol
(a="hello ") << "world"   # "hello world"
:hello << :world       # "<<" not defined

a = ["one", "two", 3 ]
a[1]          # "two"
a[1..2]       # ["two",3]
a[-1]         # 3
a[2..-1] = %w{red blue}
# ["one","two","red","blue"]
a.join(" fish " ) + " fish"
# you know that one
```

Data types

- Hash
- Sequence
- Regex

```ruby
h = { :method => "GET",
      :path => "/index.html" }
h[ :method ]              # "GET"
h[ :method ] = "POST";
h.keys            # [ :method, :path ]
h.values          # [ "GET", "/index.html" ]

def dosmth( value, options = {} )
end
dosmth "with this",
    :verbose => true, :threads => 3

1..5.class      # bad value for range
(1..5).class    # Range
MARKS = 'A'..'F'

for c in MARKS
  puts c;         # A B C D E F
end
for i in 0...5
  puts i;         # 0 1 2 3 4
end

"ABC123" =~ /[A-Z]+[0-9]+/ # 0
"ABC" =~ /[a-z]+/i          # 0
```

Syntactic sugar

- Syntax trick that make the code look more natural

```
i_prev = 0; i = 1;
i_prev,i = i,i+1     # no i++
i_prev,i = [0,1]

def whatever( reqd, deft = 0, *opt )
    puts opt.class          # Array
end
params = [ "this one required", 1 ]
whatever *params     # whatever("...",1)
car,*cdr = [1,2,3,4]
*all,last = [1,2,3,4]

class Vector
    def *(v)
        # vector multiplication
    end
end
v1 = Vector.new(...)
v2 = Vector.new(...)
v1 * v2       # you can v1 *= v2
               # v1 becomes Fixnum
1.+(2)        # 1+2 = 3
```

Classes and objects

- Single inheritance
- “initialize” instead of constructors
- ‘=‘, ‘?’, ‘!’ are valid in method name
- Result of the last expression is the return value
- Object attributes prefixed ‘@’
- class methods - self.<name>

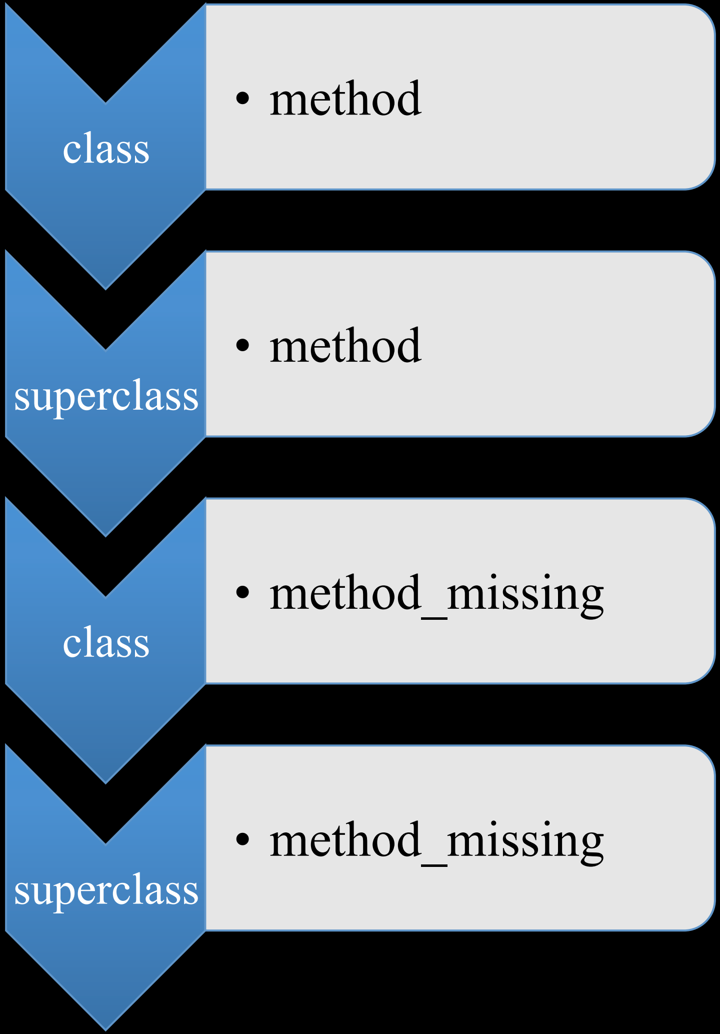
```
class SuperClass
public
    def initialize(name)
        @attr = "Hello, #{name}"
    end
    def self.report
        puts "I am a SuperClass"
        @@reported = true
    end
end

class SubClass < SuperClass
    def attr
        @attr     # the return value
    end

    def attr=( value )
        @attr = "Hello, #{value}"
    end
    public :attr,:"attr="
end

object = SubClass.new( "world" )
puts object.attr              # Hello, world
object.attr = "universe"
puts object.attr              # Hello, universe
object.report                 # !! no method defined
SuperClass.report             # I am a SuperClass
SubClass.report               # I am a SuperClass
```

Classes and objects

- Methods are truly dynamic.
- Resolved by name on the runtime, more like sending a message
- No types => no method overloading
- “super” invokes superclass method with the same parameters

```ruby
class Parent
    def run
        init_run          # not defined (yet)
        puts "Running!"
    end
end
# Parent.new.run
# undefined local variable
  or method `init_run'
class Child < Parent
    def init_run
        puts "Preparing!"
    end
end
Child.new.run
# Preparing!
# Running!
```

Classes and objects



```ruby
class SmartParent
    def run
        unless respond_to? :init_run
            puts "Can't init"
            return nil
        end
        init_run
        puts "Running!"
    end
end
```

Classes and object

```ruby
SmartParent.new.run     # prints Can't init
class SmartChild < SmartParent
     def method_missing method_name, *args
        case method_name
        when :init_run
            puts "Okay, okay, I am ready"
        else
            super
        end
     end
     def respond_to? name
        case name
        when :init_run
          true
        else
          super
        end
     end
end
SmartChild.new.run
```

Modules

- Provide a namespace
- Can have its own:
  - Constants
  - Static methods
  - Regular methods
- Mixin: “include <Name>”
- Can refer to members of class the module will be included in.

```ruby
module MyLib
    class String
        def whoami
            puts "Special String"
        end
    end

    def MyLib.whoami
        puts "MyLib module"
    end

    def whoami
        puts "MyLib user"
    end
end

String.new.whoami    # undefined method `whoami'
s2 = MyLib::String.new
s2.whoami           # Special String
MyLib.whoami        # MyLib module
MyLib::whoami       # MyLib module
class MyImpl
  include MyLib
end
MyImpl.new.whoami    # "MyLib user".
MyImpl.whoami        # undefined method

Class.superclass     # Module
```

Exceptions

```ruby
class MyError < StandardError
end

con = UnreliableConnection.new("address or smth")
con.setData ['h'.ord,'e'.ord,'l'.ord,'o'.ord]
retry_count = 0
begin
    unless con.send
      raise MyError.new, 'Failed to send'
    end
rescue MyError => er
    if (retry_count += 1) <= 3
        retry
    end
else
      puts "Unknown error"
ensure
  con.close
end
```

Catch/throw

```ruby
a1 = [ 1,10,2,9,3,8,4,7,5,6 ]

def indexof v,n
  i = 0
  while i < v.length
    if v[i] == n
      throw :gotcha, i
    end
    i += 1
  end
end

for needle in [4,33]
  n = catch( :gotcha ) {
    indexof a1,needle
  }
  puts "needle #{needle} is " +
         (n ? "found at #{n}" :
           "not found");
end
```

Statements == expressions

if / unless

while / until

case + when

loop

for

break / continue / redo

```ruby
raise Hell.new if invalid?( parameter )
array = [1,2,3]
puts "next: #{array.pop}" until array.empty?
a,b = 2,3
min = if a < b then a else b end
puts "Min: #{min}"

char = 'r'
chCase = case char
when 'A'..'Z'
  'U'
when 'a'..'z'
  'L'
else
  'O'
end
puts "Case of #{char} is #{chCase}"

puts "It is lowercase" if ('a'..'z') === char
puts "Match!" if /^[_a-zA-Z]+([0-9]+)$/ ===
'MP_123'
puts "The digits: #{$~[1]}"
```

Blocks

- Block of code passed to a method
- In the context of the caller
- May be used by the method when needed

```ruby
p1 = Proc.new do |x|
   x*2
end
p1.call(7)    # 14
p2 = lambda { |n,m|
   s = 1
   for i in (n..m)
     s *= i
   end
   s
}
p2.call(4,7)          # 840
[ 2,3,4 ].map( p1 )
# !! wrong num of arg
[ 2,3,4 ].map &p1      # [ 4,6,8 ]
[ 2,3,4 ].map { |x| x*2 }

[ 2,3,4 ].reduce(0) { |accu,x| accu + x }
# 9
p2 = lambda do |n,m|
  (n..m).reduce(1) { |ac,n| ac*n }
end
```

Closures

- Variables used by the block are added to Closure
- Closure keeps the context
- Will survive the original context
- new block/proc creates new closure (context)

```ruby
s = 1
(n..m).each { |n|
      s *= n
}

def multBy n
  lambda { |x| x*n }
end
m2 = multBy 2
[2,3,4].map &m2
[2,3,4].map &multBy(2)

def newCounter
  c = 0
  lambda { c+=1 }
end

c1 = newCounter
c1.call # 1
c1.call # 2
c2 = newCounter
c2.call # 1
```

Blocks in Ruby 1.9

Changes in Ruby 1.9

- Block parameters have their own context
- Ability to define block-local variables

```ruby
# 1.9
p1 = ->(x) { x*2 }

x = 77
3.times { |x| puts x; }
puts x;
# 2 (ruby 1.8) or 77 (1.9)
# local block variable, 1.9 only
3.times { |y;x| x=y*y; puts x }
```

Using blocks

- *yield* passes control to the block
- *block_given?* answers if method has a block
- can pass parameters to block as *yield* parameters

```
class ManyName
  def initialize( *b )
    @name = b
  end

  def each
    for n in @name
      yield n
    end
  end
end
```

Using blocks

- module defines methods that all rely on “each” method
- “each” expected to yield every element of the collection

```
module Enumrbl
    def select
        result = []
        each { |n|
          result << n if yield n
        }
        result
    end

    def reject
      select { |n| ! yield n }
    end

    def inject( acc = nil )
      each { |x|
        acc = if acc then
                yield acc,x
          else x
          end
      }
      acc
    end
end
```

Using blocks

- the class that mixes in Enumrbl defines "each"
- An example of using blocks - build a histogram of words in the input stream/files

```ruby
class ETest
  include Enumrbl
  def each(&b)
    [2,3,4,5].each( &b )
  end
end
et = ETest.new
et.inject { |ac,n| ac+n }          # 14
et.inject(-10) { |ac,n| ac+n }   # 4

#!/usr/bin/env ruby
# build freq dict
$; = /[ ,.?!;:()=\-<>'"\t]+/
dict = Hash.new {|hash,key| hash[key]=0}
$_.split.each { |w| dict[w.chomp] += 1 } while gets
dict.sort {|a,b| b[1] <=> a[1] }.each { |b|
  puts "#{b[0]}\t-\t#{b[1]}"
}
```

Open classes

- second definition is not an error
- re-open/close class to add methods
- can open even basic Ruby classes
- can define static methods without class... end

```ruby
class ETest
    def initialize( arr )
        @myarr = arr
    end
end

class ETest
    include Enumrbl
    def each(&b)
        @myarr.each( &b )
    end
end

class Array
  def all_even?
    ! find { |n| n%2 != 0 }
  end
end

[2,4,6].all_even?   # true

def Array.fib(size)
  n1,n2 = 1,0
  ar = Array.new(size) {
        (n1,n2 = n1+n2,n1)[1]
  }
end

Array.fib(7)  # [ 1,1,2,3,5,8,13 ]
```

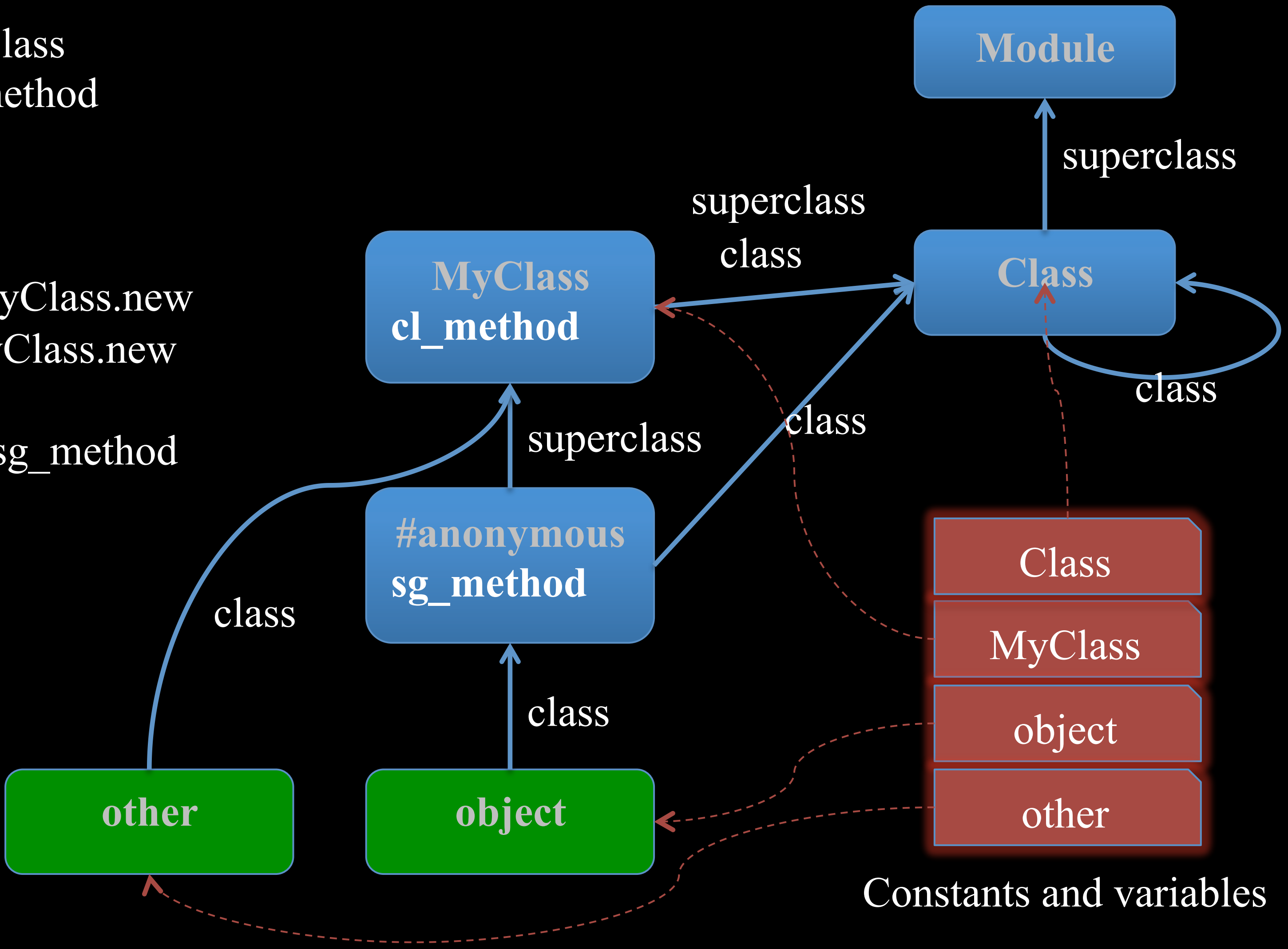
Singletons

- Not the GoF singleton pattern
- Dynamically created anonymous class
- aka metaclass
- contains methods define just for 1 object

```ruby
class Test
  def test
      puts "Test::test"
  end
end

t1 = Test.new
t2 = Test.new
def t1.uniq_test
  puts "unique to t1"
end
t1.uniq_test # "unique to t1"
t2.uniq_test # undefined method
def t2.test
    puts "t2's own test"
end
t1.test         # Test::test
t2.test         # t2's own test
t1.class        # Test
t1.singleton_class
# #<Class:#<Name:0x..>>

t1.extend SomeModule
# mixin module into singleton
```

Classes and Objects-2

```
Class MyClass
    def cl_method
    end
end

object = MyClass.new
other = MyClass.new

def object.sg_method
end
```



Constants and variables

Dynamic method definition

- attr_reader defines methods to access attributes
- attr_writer defines methods to set attributes
- attr_accessor does both
- code within class … end is actually executed

```
class Name
  attr_accessor :first,:last

  def initialize( first,last )
    @first,@last = first,last
  end
end
name = Name.new "John","Doe"
name.first # John
name.last = "Carpenter"

class Name
 puts self.public_instance_methods
end
```

Sample implementation

- open Class class
- myattr is instance method, not class
- ivar - symbol for attr
- getm - symbol for get method
- setm - symbol for set method
- define two methods
- myattr available to any class

```ruby
class Class
def myatr(*m)
  m.each { |mn|
    ivar = ("@" + mn.to_s).to_sym
    getm = mn.to_sym
    setm = (mn.to_s + "=").to_sym
    define_method(getm) do
      instance_variable_get(ivar)
    end
    define_method(setm) do |value|
      instance_variable_set(ivar,value)
    end
  }
end
end

class MyOwn
  myatr :atr1,:atr2
end
```

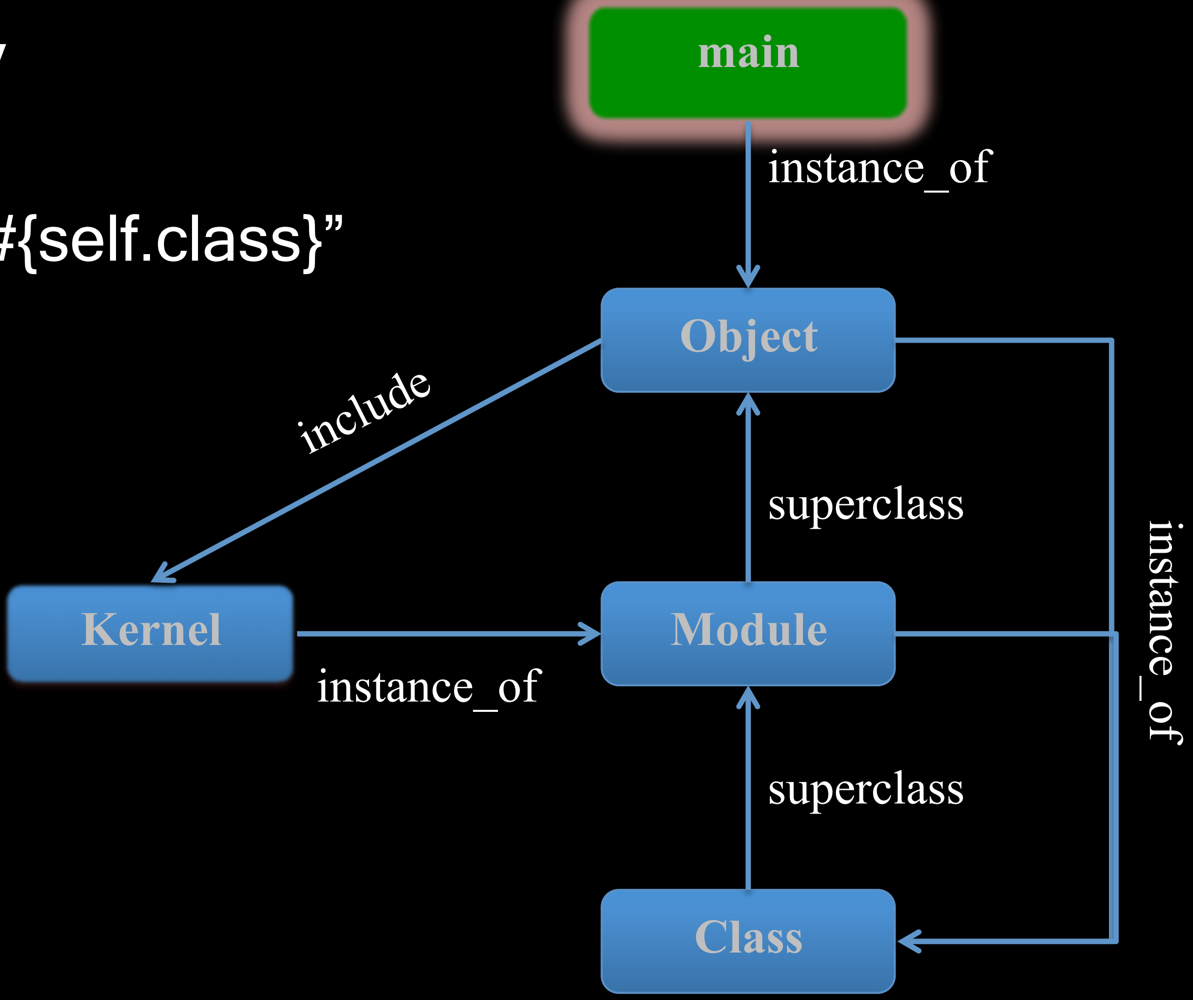
A better way

- Do not pollute standard classes unless really need
- Define a base class, define class methods
- Use in any sub-class

```ruby
class MyBase
  def self.myatr(*m)
    #...
  end
end

class MyOwn < MyBase
  myatr :attr1, :attr2
end
```

Whose line is it anyway

```
#/usr/bin/env ruby
puts “Hello”
puts “Running in #{self.class}”
# Object
puts self
# main
```



Gems

- libraries
- managed by “gem” command
- gem help
- included by “require” statement
- in 1.8 need “require ‘rubygem’” line

```ruby
#!/usr/bin/env ruby
require 'rubygems'    # 1.8 only
require 'xml/libxml'
require 'libxslt'

if ARGV.length < 2
  puts 'Usage: xslt.rb file.xml '+
    'file.xsl. May switch if '+
    'extensions are "xml" and "xsl[t]"'
  exit 1;
end

fxml,fxsl = ARGV[0],ARGV[1];
if fxml =~ /\.xslt?$/ && fxsl =~ /\.xml$/
  fxml,fxsl = fxsl,fxml;
end

puts "Processing xml file #{fxml} with stylesheet #{fxsl}"

ssheet = LibXSLT::XSLT::Stylesheet.new(
    LibXML::XML::Document.file( fxsl ) )
xmldoc = LibXML::XML::Document.file( fxml );

puts ssheet.apply( xmldoc );
```

Sinatra web app

- Microframework for web apps
- One file can have it all
- Framework defines Delegate module and then calls “extend Delegate” on “main” object
- that’s’ where *get* comes from

```
#!/usr/bin/env ruby
#require 'rubygems'  # 1.8 only
require 'sinatra'

fzisme = self.singleton_class.ancestors

get '/hello/:name' do
  @name = params[:name]
  @me = fzisme
    erb :hello
  end

__END__
@@ layout
<html>
  <body>
    <%= yield %>
  </body>
</html>

@@ hello
<h3><%= @name %> in the <%= @me %> </h3>
```

Extras: ruby on Mac

Install ruby on Mac using rbenv

1. Install homebrew (written in ruby)

   ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"

   Warning: Install the "Command Line Tools for XCode": http://connect.apple.com

2. brew install rbenv
3. brew install ruby-build
4. rbenv install 1.9.3-p392