Ruby

Ruby

Vorlesung Dynamische Programmiersprachen

Carl Friedrich Bolz, David Schneider

SoSe 2010 - Heinrich-Heine Universität Düsseldorf

About Ruby

Dynamic/Interpreted Language

Released in 1995

Developed by Yukihiro “matz” Matsumoto

Inspired by Smalltalk, Lisp, Perl among others

Many implementations on different Platforms

JRuby (JVM)

IronRuby (.Net/Mono)

MacRuby (Mac OS)

MagLev (Smalltalk)

Some general hints

Ruby is an interpreted language

There are no statements: everything is an expression

Every line is evaluated and has a return value

Object Orientation in Ruby

Everything is an Object

```
"str".capitalize()
=> "STR"
```

Message Based

```
"str".capitalize()
```

```
"str".capitalize()

# is equivalent to

"str".capitalize
```

```
"str".capitalize()

# is equivalent to

"str".capitalize

# is equivalent to

"str".send('capitalize')
```

Operators are Messages

```
# Infix operator notation
3 + 2
```

```
# Infix operator notation
3 + 2

# is equivalent to
3.+(2)
```

```
# Infix operator notation
3 + 2

# is equivalent to
3.+(2)

# and equivalent to
3.send('+', 2)
```

Object Orientation

Every object is an instance of a class

Single inheritance

(Types can be extended with modules)

```
"foo".class()
=> String
```

```
"foo".class()
=> String

String.class()
=> Class
```

```
"foo".class()
=> String

String.class()
=> Class

String.superclass()
=> Object
```

```
"foo".class()
=> String

String.class()
=> Class

String.superclass()
=> Object

String.ancestors()
=> [String, Object, Kernel, BasicObj
```

Example

```ruby
class A
  def initialize()
    @i = 123
  end
  def i=(value)
    @i = value
  end

  def i()
    @i
  end
end
```

```
class B < A
  def hi()
    @i + 10
  end
end
```

Open Classes

Extend loaded classes with additional behaviour

Redefine existing behaviour of classes

User and core classes can be opened

```
class Array
  def rand
    self[Kernel::rand(self.size)]
  end
end
```

Problems with Open Classes

Name clashes

Unintended behaviour

Breaks modularity

Example

Open Clases, Operators

```ruby
class Vector
  def initialize(values)
    @values = values
  end

  def *(n)
    b = Array.new(@values.length)
    for i in (0...@values.length)
      b[i] = @values[i] * other
    end
    Vector.new(b)
  end
```

```
class Fixnum

  alias mult *

  def *(other)
    if other.is_a?(Vector)
      other * self
    else
      self.mult(other)
    end
  end
end
```

Message sending and lookup

Messages are sent to a receiver

```
receiver.message(*arguments)

# is the same as

receiver.send('message', *arguments)
```

The receiver can be implicit

```
message(*arguments)

# is the same as

self.message(*arguments)
```

Method lookup

lookup the method in the class of the receiver

lookup the method in the ancestors of the class

If the method is found call it

Example

Method Missing

Python style attributes

```
class PyObject
  def initialize(properties)
    @dict = properties
  end
# snip
# cont.
  def method_missing(name, *args)
    if name.to_s[-1] == '='
      @dict[name.to_s[0...-1]] = arg
    elsif @dict.include?(name.to_s)
      @dict[name.to_s]
    else
      super
    end
  end
end
```

The Object Model

```
cat = "lol"
dog = "grr"
```

```
def cat.meme?()
  true
end
```

```
cat.meme?()

=> true
```

```
dog.meme?()

NoMethodError
```

How does this work?

This works through invisible classes

These objects are called singleton class

Mostly invisible in the language

```
klass = class << cat; self; end

klass.instance_methods.include?(:mem

=> true
```

```
klass = class << cat; self; end

klass.instance_methods.include?(:mem

=> true


cat.class()

=> String
```

```
klass = class << cat; self; end
klass.instance_methods.include?(:mem
=> true

cat.class()
=> String

cat.is_a?(klass)
```

Singleton Classes

Every object has exactly one

Every singleton class is associated to exactly one object

It sits between the object and its class

Singleton Classes (2)

If classes are Objects

And classes are instances of object

how are class methods defined?

Example

```
class Cat
  def self.miaow()
    "miaow"
  end
end
```

Example (2)

```
class Cat
  def Cat.miaow()
    "miaow"
  end
end
```

Singleton Classes (3)

classes also have singleton classes

classes are instances of the class Class

class methods are defined on the singleton class of the class

Object Model

Simle Objectmodel

Simle Objectmodel

Simle Objectmodel

Blocks

Blocks

```
[1, :two, "three"].each do |item|
  puts "#{item}: #{item.class}"
end
```

Blocks

In the beginning they were designed to abstract loop iterations

Blocks are basically nameless functions

Code passed as argument to methods

Blocks

Can be executed

Can take parameters

Can be invoked by the function it is an argument to

Example

```
def with_square(x)
  yield x**2
  print "After block #{x}"
end

with_square(2) { |v| puts "In block
with_square(23) { |v|
        puts "Square of #{x} is #{v}
=> NameError: undefined local variab
      method 'x' for #<Object:0x0…>

x = 23
with_square(23) { |v|
        puts "Square of #{x} is #{v}
=> Square of 23 is 529
=> After block 23
```

Blocks

Blocks can be transformed to closures which are called Procs

Procs are similar to lambdas anonymous, callable, support for currying

Procs can be converted to blocks

Blocks also

Are objects

Can be instantiated from their class

Examples

Example

Map

```
class Array
  def my_map()
    raise ArgumentError unless block
    a = Array.new(self.length)
    for i in (0...self.length)
      a[i] = yield(self[i])
    end
    a
  end
end
```

```
class Array
  def my_map()
    raise ArgumentError unless block
    Array.new(self.length) { |i|
      yield self[i]
    }
  end
end
```

Example

Blocks

```
"my_file.txt".as_file {|f|
  puts f.readlines
}
"my_file.txt".as_file('w') { |f|
  f << 'hi'
}
"my_file.txt".as_file => File object
```

```ruby
class String
  def as_file(m='r')
    f = File.new(self, m)
    if block_given?
      yield f
      f.close
    else
      f
    end
  end
end
```

```
class String
  def as_file(m='r', &block)
    File.open(self, m, &block)
  end
end
```

Example

getters and setters

```
class A
  attr_accessor :i
  def initialize
    @i = 10
  end
end
```

```
class Foo
  my_attr_accessor :a
end
```

```
class Class
  def my_attr_accessor(name)
    define_method("#{name}=") do |va
      instance_variable_set(
        "@#{name}", value
      )
    end

    define_method(name) do
      instance_variable_get("@#{name
    end
  end
end
```

Ruby is a Dynamic Language

but Why?

Dynamic Typing

```
a = 1
a.class()

a = 'asdf'
a.class()
```

Most Things Changeable at Runtime

Example: Vector, Operator Definitions

Reflection

```
class Foo
  def self.hi()
    puts "hi"
  end
  def self.bye()
    puts "bye"
  end
end

Foo.singleton_methods()
=> [:hi, :bye]
```

"Late Bound Everything"

```
def b(i)
  a(i)
end

def a(i)
  i * 2
end

>> b(10)
=> 20
```

Everything is an Object

```
10.times { puts "Foo Bar" }
```