CS 5142
Scripting Languages

10/25/2012

Ruby

Outline

- Ruby

About Ruby

- Invented 1995 by Yokihiro “Matz” Matsumoto
- Influenced by SmallTalk
  - Everything is an object (even e.g., integers)
  - Blocks and user-defined control constructs
- Influenced by Perl
  - RegExp match `=~ /…/`, default variable `$_`
- Common use: RoR (Ruby on Rails) framework for web programming

How to Write + Run Code

- One-liner: **`ruby -e '`***command*`'`**
- Run script from file: **`ruby`** *file*`.rb`**
- Debugger: **`ruby -rdebug`** *file*`.rb`**
- Check syntax only: **`ruby -c`** *file*`.rb`**
- Read-eval-print loop (interactive Ruby): **`irb`**
- Run stand-alone: **`#!/usr/bin/env ruby`**

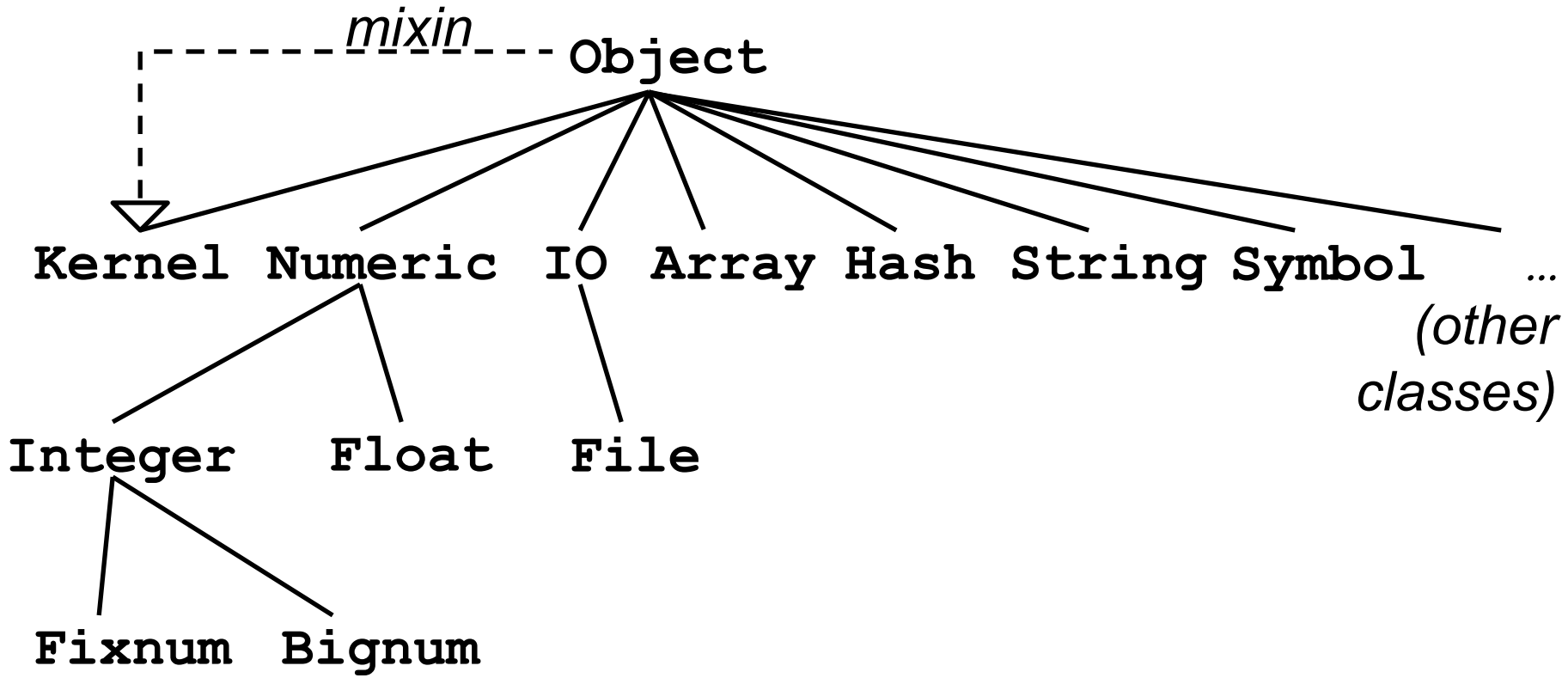
Example

```
#!/usr/bin/env ruby
$cup2g = { 'flour'=>110, 'sugar'=>225, 'butter'=>225 }
$volume = { 'cup'=>1, 'tbsp'=>16, 'tsp'=>48, 'ml'=>236 }
$weight = { 'lb'=>1, 'oz'=>16, 'g'=>453 }
while gets
  $_ =~ /([0-9.]+) (\w+) (\w+)/
  $qty, $unit, $ing = [$1, $2, $3]
  if $cup2g.key?($ing) and $volume.key?($unit)
    $qty = $qty.to_f * $cup2g[$ing] / $volume[$unit]
    $unit = 'g'
  elsif $volume.key? $unit
    $qty = $qty.to_f * $volume['ml'] / $volume[$unit]
    $unit = 'ml'
  elsif $weight.key? $unit
    $qty = $qty.to_f * $weight['g'] / $weight[$unit]
    $unit = 'g'
  end
  puts "qty #{$qty.to_i}, unit #{$unit}, ing #{$ing}\n"
end
```

Lexical Peculiarities

- Case sensitive
- Single-line comment: `#`…
- Multi-line comment: `=begin` … `=end`
- Line break ends statement, optional semicolon (`;`)
- Sigils indicate scope, not type
- Heredocs
- Expression interpolation: `"`… `#{` … `}` …`"`
- Literals: `"s"`, `'s'`, `true`, `nil`, RegExp `/`…`/`, Array `[1,2,3]`, Hash `{'x'=>1,'y'=>2}`, Symbol `:foo`

Types


mixin
Object
Kernel
Numeric
IO
Array
Hash
String
Symbol
…
(other classes)
Integer
Float
File
Fixnum
Bignum

Variable Declarations

- There are no explicit variable declarations
- Reading an undefined variable is an error
- Scope

Scope	First letter
Local variable or method	Lowercase or _
Object variable (private)	@
Class variable (static)	@@
Global variable	$
Constant; convention: • MixedCase = class name • ALL_CAPS = value	Uppercase

Ruby

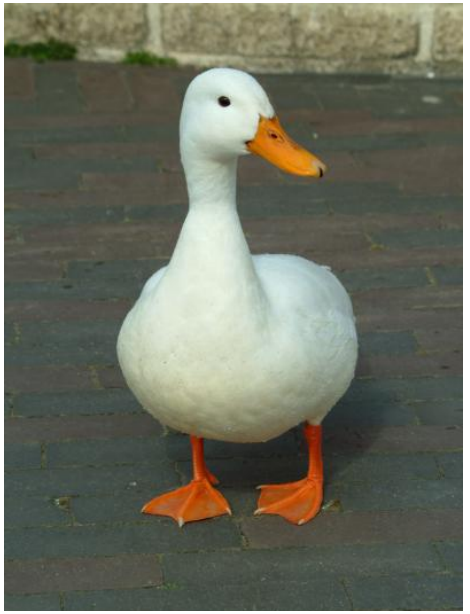
Type Conversions

Class	Value	Boolean	Number	String
`FalseClass`	`false`	Identity	Error	Error
`TrueClass`	`true`			
`Fixnum`	`0`	`true`	Identity	Error
	Other			
`String`	`""`	`true`	Error	Identity
	`"0"`			
	Other			
`NilClass`	`nil`	`false`	Error	Error
`Array`	Empty	`true`	Error	Error
	Other			
`Object`		`true`	Error	Error

Explicit conversions: `x.to_i()`, `x.to_f()`, `x.to_s()`, …

Duck Typing

- If it walks like a duck and quacks like a duck, it's a duck.
- If a value supports the operations for a certain type, Ruby treats it as that type.
- This is normal for dynamic typing, but unusual for object-oriented languages.



Input and Output

- Output: **`p()`**, **`print()`**, **`printf()`**, **`puts()`**
- Input: **`gets()`**, **`readline()`**, **`readlines()`**
- Like Perl, Ruby has implicit variables
  - E.g., **`$1`**, **`$2`**, … are groups from pattern match, and **`$_`** is result of **`gets`** or **`readline`**
- Many operators are actually methods
  - E.g., **`+`** and **`-`** are methods on number objects, and **`=~`** is method on string objects for RegExp match

Ruby

Operators

`::`	2	Scope resolution
`[]`, `[]=`	2	Read from array, write to array
`**`	2	Exponentiation
`+`, `-`, `!`, `~`	1	Positive, negative, negation, complement
`*`, `/`, `%`	2	Multiplicative
`+`, `-`	2	Additive
`<<`, `>>`	2	Shifting
`&`, `|`, `^`	2	Bitwise (not all same precedence)
`>`, `>=`, `<`, `<=`	2	Comparison
`<=>`, `==`, `===`, `!=`, `=~`, `!~`	2	Identity, pattern matching
`&&`, `||`	2	Logical (not all same precedence)
`..`, `...`	2	Range inclusive, exclusive
`?:`	3	Conditional
`=`, `+=`, `-=`, ...	2	Assignment
`not`, `and`, `or`		Logical (not all same precedence)
`defined?`	1	(no precedence)

Control Statements

Conditional		**`if`** *expr* **`then`** … **`elsif`** … **`else`** … **`end`** **`unless`** *expr* **`then`** … **`elsif`** … **`else`** … **`end`** **`case`** *expr* **`when`** *expr*`:`… … **`else`** … **`end`**
Loops	Fixed	**`for`** *var* **`in`** *expr* **`do`** … **`end`**
	Indefinite	**`while`** *expr* **`do`** … **`end`** **`until`** *expr* **`do`** … **`end`**
Unstructured control		**`break`**, **`redo`**, **`next`**, **`retry`** **`return`** [*expr*] **`yield`**
Exception handling		**`begin`** … **`rescue`** *Cls* … **`ensure`** … **`end`** **`raise`** [*Cls*]
Modifiers		*stmt* [**`if`** | **`unless`** | **`while`** | **`until`**] *expr*

Alternative Control Syntax

```
if x==5: puts 'five' end
if x==5 then puts 'five' end
if x==5
  puts 'five'
end
```

Less variation for other control statements, e.g.,

- **`while`** with **`do`** or *newline*
- **`for`** with **`do`** or *newline*

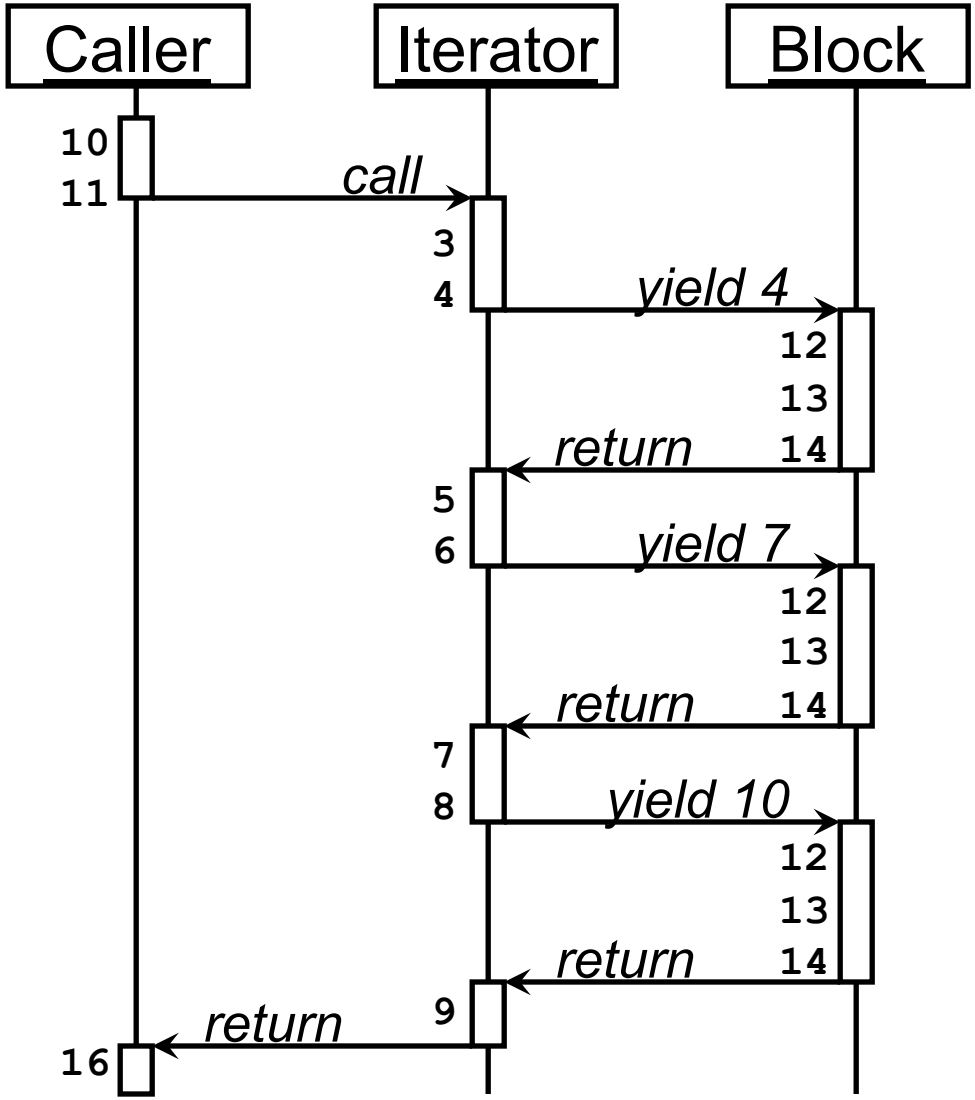
Writing Subroutines

- Declaration: **`def`** *id* [**`(`***arg*`*`**`)`**] ... **`end`**
  - Explicit **`return`**, or value of last expression in body
  - Convention: when *id* ends with **`?`**, return boolean; **`=`**, set field; **`!`**, make destructive update
  - **`yield`** executes block parameter; when block terminates, resume current subroutine after **`yield`**
- Arguments: *arg* ::= [**`*`**]*id*
  - Splat (**`*`**) gives variable number of arguments
- Deleting a subroutine: **`undef`** *id*
- Duplicating a subroutine: **`alias`** id_1 id_2
- Block (parameter closure): *block* ::= **`{ |`***arg*`*`**`|`** ...**`}`**
- Procs (first-class closure): (**`proc`** | **`lambda`**) *block*

Ruby

Iterators

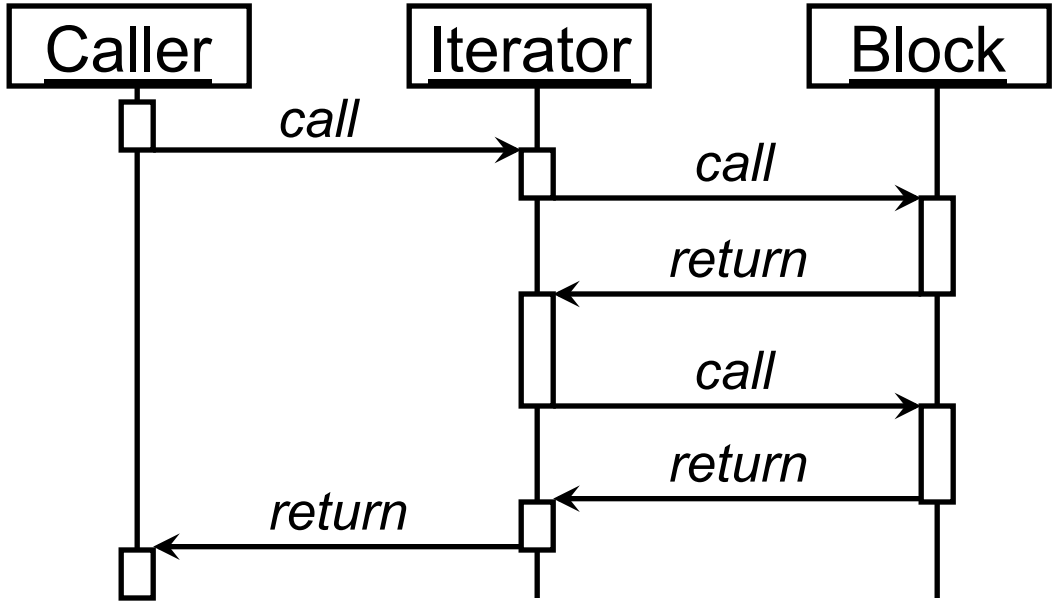
```
#!/usr/bin/env ruby   # 1
def myIterator(x)     # 2
  x += 3              # 3
  yield x             # 4
  x += 3              # 5
  yield x             # 6
  x += 3              # 7
  yield x             # 8
end                   # 9
$i = 0                #10
myIterator(1){|y|     #11
  $i += 1             #12
  puts "call #{$i}:" #13
  puts y              #14
}                     #15
puts 'done'           #16
```



Many Ruby classes have iterator methods that take a block parameter.

Code Blocks as Parameters

- Code block = { | *arg** | *stmt** }
  syntax for anonymous function
  - Similar to a lambda
  - Can be passed as parameter to iterator method
  - The code block is a closure
    (has access to caller's environment)



Using Objects

`a1 = Apple.new(150, "green")` `a2 = Apple.new(150, "green")`	Constructor calls
`a2.color= "red"`	Setter call
`puts a1.prepare("slice") + "\n"` `puts a2.prepare("squeeze") + "\n"`	Method calls

Ruby

Defining Classes

```
class Fruit
  def initialize(weight_)
    @weight = weight_ end
  def weight
    @weight end
  def weight= (value)
    @weight = value end
  def pluck
    "fruit(" + @weight + "g)" end
  def prepare(how)
    how + "d " + pluck end
end
```

Fruit
@weight
initialize() pluck() prepare()

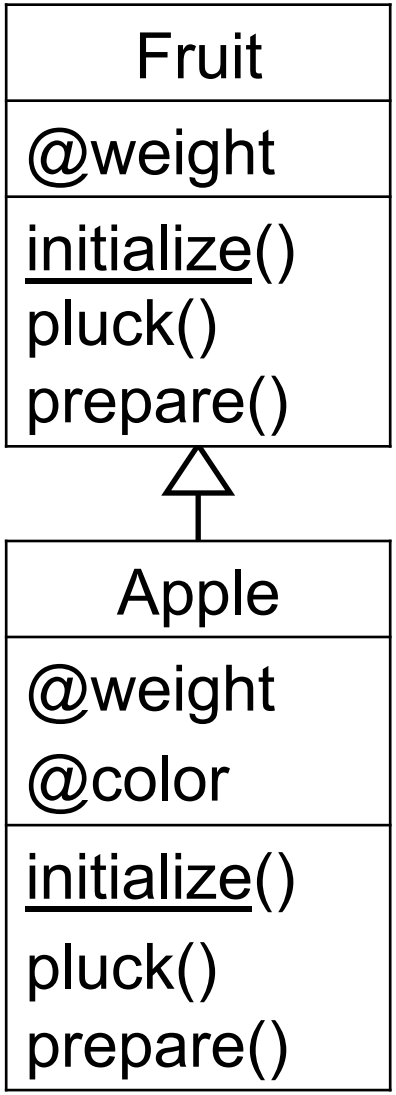
- All fields are private, external use requires accessors (e.g., **`@weight`**, **`weight`**, **`weight=`**)
- Classes are open, can add additional fields+methods

Ruby

Inheritance in Ruby

```
class Fruit
  def initialize(weight_)
    @weight = weight_ end
  def weight
    @weight end
  def weight= (value)
    @weight = value end
  def pluck
    "fruit(" + @weight + "g)" end
  def prepare(how)
    how + "d " + pluck end
end
```

```
class Apple < Fruit
  def initialize(weight_, color_)
    @weight = weight_
    @color = color_
  end
  def color
    @color end
  def color= (value)
    @color = value end
  def pluck
    self.color + " apple" end
end
```



Scopes and Visibility

- Visibility of class members
  - All instance variables are private
  - Methods can be private, protected, or public
- Accessor generation

```
class Fruit
  attr_accessor :weight
  def initialize(weight_)
    @weight = weight_
  end
  def pluck
    "fruit(" + @weight + "g)"
  end
  def prepare(how)
    how + "d " + pluck
  end
end
```

Generates `@weight` field and `weight`/`weight=` methods

Fruit
@weight
initialize() pluck() prepare()

Structure of a Ruby Application

- **`require`** *file*
- Module = like class, but can't be instantiated
  - Class can **`include`** ("mix in") one or more modules
  - Members of mix-in module are copied into class
  - Later definition with same name overrides earlier
  - Module can inherit from other module, but not class
  - Module can contain methods, classes, modules
- Module **`Kernel`** is mixed into class **`Object`**
- Top-level subroutines are private instance methods of the Kernel module
  - Visible everywhere, can't call with explicit receiver

Arrays

- Initialization: **`$a=[1,2,3]`**
  - With block: **`$a=Array.new(10){|e|2*e}`**
- Indexing: **`$a[…]`**
  - Zero-based, contiguous, integers only
  - Negative index counts from end
- Deleting: **`$a.clear()`**, **`$a.compact()`**, **`$a.delete_at(i)`**
- Lots of other methods

Hashes

- Initialization:

```
$h = {'lb'=>1,'oz'=>16,'g'=>453}
```

- Indexing: **`$h['lb']`**
  – Can use any object as key, not just strings
- Deleting: **`$h.clear()`**, **`$h.delete(k)`**
- Lots of other methods
- Can have a “default closure”:
  return value for keys not explicitly stored

Reference

Ruby Documentation

- http://www.ruby-lang.org
- http://www.rubyonrails.org
- Book: The Ruby Programming Language. David Flanagan, Yokihiro Matsumoto. O'Reilly, 2008

Soap-Box

Evaluating Ruby

Strengths

- Rails
- Purely object oriented
- Perl-like =~ and default variables

Weaknesses

- Less popular than Java and PHP
- Unusual syntax

Last Slide

- No announcements.
- Today's lecture
  - Ruby