CMSC 330: Organization of Programming Languages

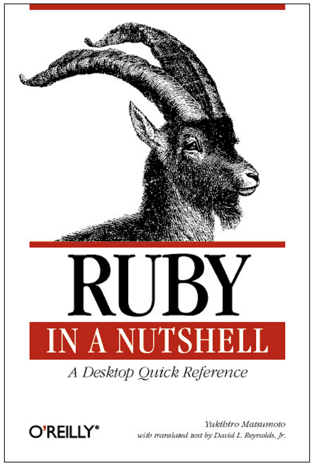
Ruby

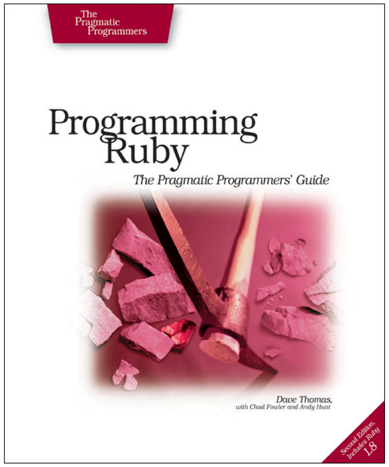
Introduction

- Ruby is an *object-oriented, imperative scripting language*
  - "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language."
  - "I believe people want to express themselves when they program. They don't want to fight with the language. Programming languages must feel natural to programmers. I tried to make people enjoy programming and concentrate on the fun and creative part of programming when they use Ruby."

– Yukihiro Matsumoto ("Matz")

Books on Ruby

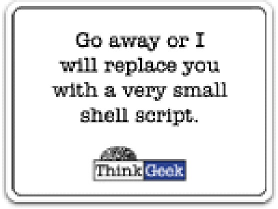

RUBY
IN A NUTSHELL
A Desktop Quick Reference
O'REILLY
Yukihiro Matsumoto
with translated text by David L. Reynolds, Jr.


The Pragmatic Programmers
Programming Ruby
The Pragmatic Programmers' Guide
Dave Thomas, with Chad Fowler and Andy Hunt
Second Edition Includes Ruby 1.8

- Earlier version of Thomas book available on web
  - See class webpage

Applications of Scripting Languages

- Scripting languages have many uses
  - automating system administration
  - automating user tasks
  - quick-and-dirty development



Go away or I will replace you with a very small shell script.

- Major application:

Text processing

Output from Command-Line Tool

```
% wc *
     271     674    5323 AST.c
     100     392    3219 AST.h
     117    1459  238788 AST.o
    1874    5428   47461 AST_defs.c
    1375    6307   53667 AST_defs.h
     371     884    9483 AST_parent.c
     810    2328   24589 AST_print.c
     640    3070   33530 AST_types.h
     285     846    7081 AST_utils.c
      59     274    2154 AST_utils.h
      50     400   28756 AST_utils.o
     866    2757   25873 Makefile
     270     725    5578 Makefile.am
     866    2743   27320 Makefile.in
      38     175    1154 alloca.c
    2035    4516   47721 aloctypes.c
      86     350    3286 aloctypes.h
     104    1051   66848 aloctypes.o

...
```

Climate Data for IAD in August, 2005

```
================================================================================
1   2   3   4   5  6A  6B    7    8   9   10  11  12  13   14  15   16   17  18
                                          AVG MX 2MIN
DY MAX MIN AVG DEP HDD CDD  WTR  SNW DPTH SPD SPD DIR MIN PSBL S-S WX    SPD DR
================================================================================


 1  87  66  77   1   0  12 0.00  0.0    0  2.5  9 200   M    M   7 18     12 210
 2  92  67  80   4   0  15 0.00  0.0    0  3.5 10  10   M    M   3 18     17 320
 3  93  69  81   5   0  16 0.00  0.0    0  4.1 13 360   M    M   2 18     17 360
 4  95  69  82   6   0  17 0.00  0.0    0  3.6  9 310   M    M   3 18     12 290
 5  94  73  84   8   0  19 0.00  0.0    0  5.9 18  10   M    M   3 18     25 360
 6  89  70  80   4   0  15 0.02  0.0    0  5.3 20 200   M    M   6 138    23 210
 7  89  69  79   3   0  14 0.00  0.0    0  3.6 14 200   M    M   7 1      16 210
 8  86  70  78   3   0  13 0.74  0.0    0  4.4 17 150   M    M  10 18     23 150
 9  76  70  73  -2   0   8 0.19  0.0    0  4.1  9  90   M    M   9 18     13  90
10  87  71  79   4   0  14 0.00  0.0    0  2.3  8 260   M    M   8 1      10 210
...
```

Raw Census 2000 Data for DC

```
u108_S,DC,000,01,0000001,572059,72264,572059,12.6,572059,572059,572059,0,0,
    0,0,572059,175306,343213,2006,14762,383,21728,14661,572059,527044,15861
    7,340061,1560,14605,291,1638,10272,45015,16689,3152,446,157,92,20090,43
    89,572059,268827,3362,3048,3170,3241,3504,3286,3270,3475,3939,3647,3525
    ,3044,2928,2913,2769,2752,2933,2703,4056,5501,5217,4969,13555,24995,242
    16,23726,20721,18802,16523,12318,4345,5810,3423,4690,7105,5739,3260,234
    7,303232,3329,3057,2935,3429,3326,3456,3257,3754,3192,3523,3336,3276,29
    89,2838,2824,2624,2807,2871,4941,6588,5625,5563,17177,27475,24377,22818
    ,21319,20851,19117,15260,5066,6708,4257,6117,10741,9427,6807,6175,57205
    9,536373,370675,115963,55603,60360,57949,129440,122518,3754,3168,22448,
    9967,4638,14110,16160,165698,61049,47694,13355,71578,60875,10703,33071,
    35686,7573,28113,248590,108569,47694,60875,140021,115963,58050,21654,36
    396,57913,10355,4065,6290,47558,25229,22329,24058,13355,10703,70088,657
    37,37112,21742,12267,9475,9723,2573,2314,760,28625,8207,7469,738,19185,
    18172,1013,1233,4351,3610,741,248590,199456,94221,46274,21443,24831,479
    47,8705,3979,4726,39242,25175,14067,105235,82928,22307,49134,21742,1177
    6,211,11565,9966,1650,86,1564,8316,54,8262,27392,25641,1751,248590,1159
    63,4999,22466,26165,24062,16529,12409,7594,1739,132627,11670,32445,2322
    5,21661,16234,12795,10563,4034,248590,115963,48738,28914,19259,10312,47
    48,3992,132627,108569,19284,2713,1209,509,218,125
...
```

A Simple Example

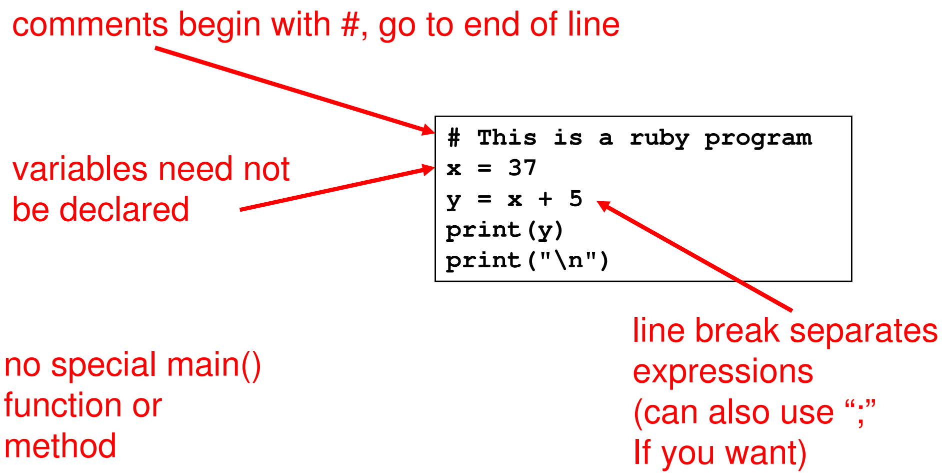
- Let’s start with a simple Ruby program

ruby1.rb:

```
# This is a ruby program
x = 37
y = x + 5
print(y)
print("\n")
```

```
% ruby -w ruby1.rb
42
%
```

Language Basics



comments begin with #, go to end of line

variables need not be declared

```
# This is a ruby program
x = 37
y = x + 5
print(y)
print("\n")
```

no special main() function or method

line break separates expressions (can also use ";" If you want)

Run Ruby, Run

- There are three ways to run a Ruby program
  - ruby -w *filename* – execute script in *filename*
    - tip: the -w will cause Ruby to print a bit more if something bad happens
  - irb – launch interactive Ruby shell
    - can type in Ruby programs one line at a time, and watch as each line is executed

```
irb(main):001:0> 3+4
=> 7
irb(main):002:0> print("hello\n")
hello
=> nil
```

Run Ruby, Run (cont'd)

- Suppose you want to run a Ruby script as if it were an executable

```
#!/usr/local/bin/ruby -w
print("Hello, world!\n")
```

    - ./filename   # run program
  - the first line ("shebang") tells the system where to find the program to interpret this text file
  - must chmod u+x *filename* first
    - or chmod a+x *filename* so everyone has exec permission
  - warning: not very portable
    - depends on location /usr/local/bin/ruby

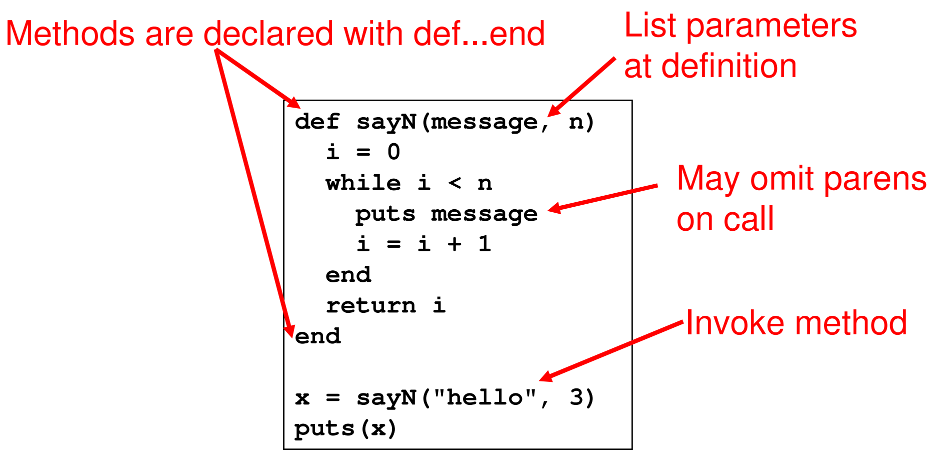
Explicit vs. Implicit Declarations

- Java and C/C++ use *explicit variable declarations*
  - variables are named and typed before they are used
    - int x, y; x = 37; y = x + 5;
- In Ruby, variables are *implicitly declared*
  - first use of a variable declares it and determines type
    - x = 37; y = x + 5;
      - x, y exist, will be integers

Tradeoffs?

Explicit declarations	Implicit declarations
higher overhead	lower overhead
helps prevent typos	easy to mistype variable name
forces programmer to document types	figures out types of variables automatically

Methods in Ruby



Methods are declared with def...end

List parameters at definition

May omit parens on call

Invoke method

```
def sayN(message, n)
  i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end

x = sayN("hello", 3)
puts(x)
```

(Methods must begin with a lowercase letter, and be defined before they can be called, if being called in the same scope which they're defined in.)

Method (and Function) Terminology

- *Formal parameters* – the parameters used in the body of the method
  - message, n in our example
- *Actual parameters* – the arguments passed in to the method at a call
  - "hello", 3 in our example

More Control Statements in Ruby

- A *control statement* is one that affects which statement is executed next
  - we've seen two so far in Ruby
    - while and function call
- Example using Ruby conditionals:

```
if grade >= 90 then
  puts "You got an A"
elsif grade >= 80 then
  puts "You got a B"
elsif grade >= 70 then
  puts "You got a C"
else
  puts "You’re not doing so well"
end
```

What is True?

- The *guard* of a conditional is the expression that determines which branch is taken

```
if grade >= 90 then
...
```

guard

- The *true* branch is taken if the guard evaluates to anything except
  - false
  - nil
- **Warning** to C programmers: 0 is *not* false!

Yet More Control Statements in Ruby

- unless *cond* then *stmt-f* else *stmt-t* end
  - same as "if not cond then *stmt-t* else *stmt-f* end"

```
unless grade < 90 then
  puts "You got an A"
else unless grade < 80 then
  puts "You got a B"
end
end
```

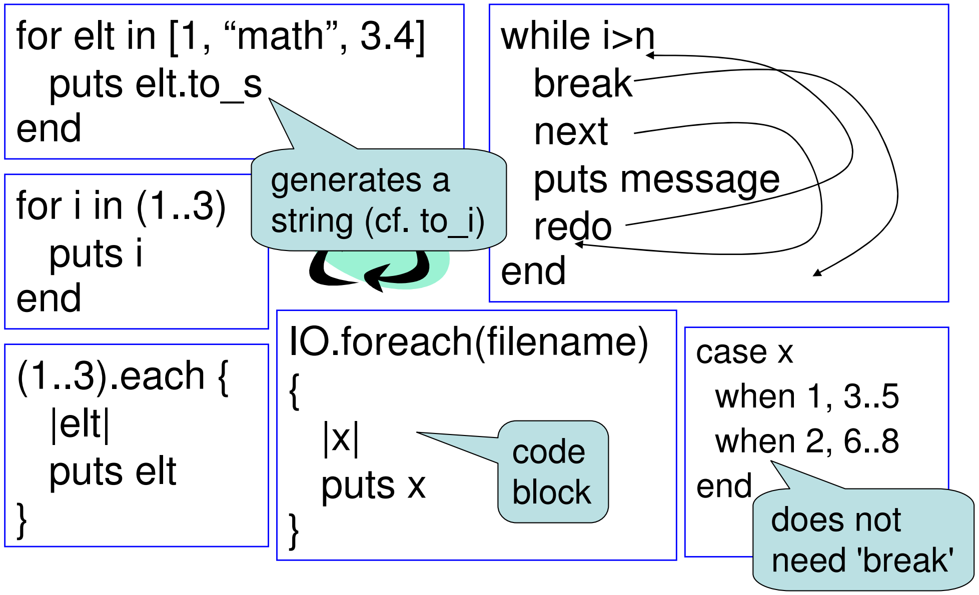
- until *cond body* end
  - same as "while not *cond body* end"

```
until i >= n
  puts message
  i = i + 1
end
```

Using If and Unless as Modifiers

- Can write if and unless *after* an expression
  - puts "You got an A" if grade >= 90
  - puts "You got an A" unless grade < 90
- Why so many control statements?
  - Is this a good idea?
  - Advantages? Disadvantages?

Other Useful Control Statements



```
for elt in [1, "math", 3.4]
  puts elt.to_s
end
```

generates a string (cf. to_i)

```
for i in (1..3)
  puts i
end
```

```
while i>n
  break
  next
  puts message
  redo
end
```

```
(1..3).each {
  |elt|
  puts elt
}
```

```
IO.foreach(filename)
{
  |x|
  puts x
}
```

code block

```
case x
 when 1, 3..5
 when 2, 6..8
end
```

does not need 'break'

Using Ruby Control Statements

Ruby function to print all even numbers from 1 to some given value x

```
def even(x)
         for i in (1..x)
            if i % 2 == 0
       puts i
    end
  end
end
```

```
def even(x)
  (1..x).each{
     |i|
     if i % 2 == 0
        puts i
     end
  }
end
```

Classes and Objects

- Class names begin with an uppercase letter
- The "new" method creates an object
  - s= String.new creates a new String and makes s refer to it
- Every class inherits from Object

Everything is an Object

- In Ruby, *everything* is in fact an object
  - (-4).abs
    - integers are instances of Fixnum
  - 3 + 4
    - infix notation for “invoke the + method of 3 on argument 4”
  - "programming".length
    - strings are instances of String
  - (4.13).class
    - use the class method to get the class for an object
    - floating point numbers are instances of Float
  - String.new
    - classes are objects with a new method

Objects and Classes

- Objects are items of data
- Classes are types (the kind of data that things are)
- But in Ruby, classes themselves are objects!

Object	Class
10	Fixnum
-3.30	Float
"CMSC 330"	String
String.new	String
Fixnum	Class
String	Class

- Fixnum, Float, String, etc., (including Class), are objects of type Class

Two Cool Things to Do with Classes

- Since classes are objects, you can manipulate them however you like

  -
    ```
    if p then x = String else x = Time end   # Time is
     …                                       # another class
    y = x.new           # creates a String or a Time,
                        # depending upon p
    ```
- You can get names of all the methods of a class
  - Object.methods
    - => ["send", "name", "class_eval", "object_id", "new", "autoload?", "singleton_methods", ...]

The nil Object

- Ruby uses a special object nil
  - all uninitialized fields set to nil (@ refers to a class field)

    ```
    irb(main):004:0> @x
    => nil
    ```
  - like NULL or 0 in C/C++ and null in Java
- nil is an object of class NilClass
  - it's a *singleton object* – there is only one instance of it
    - NilClass does *not* have a new method
  - nil has some methods (like to_s, which returns a string representation of an object) but not other methods that don't make sense

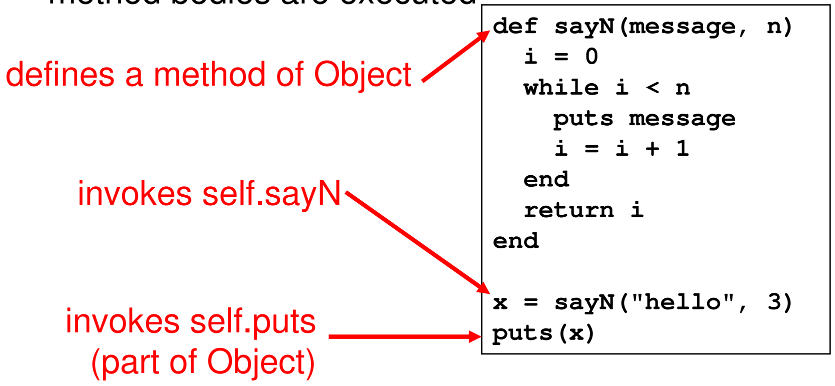
    ```
    irb(main):006:0> @x + 2
    NoMethodError: undefined method `+' for nil:NilClass
    ```

What is a Program?

- In C/C++, a program is...
  - a collection of declarations and definitions
  - with a distinguished function definition
    - int main(int argc, char *argv[]) { ... }
  - when you run a C/C++ program, it's like the OS calls main(...)
- In Java, a program is...
  - a collection of class definitions
  - with a class Cl that contains a method
    - public static void main(String[] args)
  - when you run java Cl, the main method of class Cl is invoked

A Ruby Program is...

- The class Object
  - when the class is loaded, any expressions not in method bodies are executed



defines a method of Object

invokes self.sayN

invokes self.puts (part of Object)

```
def sayN(message, n)
  i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end

x = sayN("hello", 3)
puts(x)
```

Ruby is Dynamically Typed

- Recall we don’t declare types of variables
  - but Ruby does keep track of types at runtime

    x = 3; x.f

    NoMethodError: undefined method 'f' for 3:Fixnum
- We say that Ruby is *dynamically typed*
  - types are determined and checked at run time
- Compare to C, which is *statically typed*

```
# Ruby
x = 3
x = "f"   # gives x a
          # new type
```

```
/* C */
int x;
x = 3;
x = "f"; /* not allowed */
```

Types in Java and C++

- Is Java statically or dynamically typed?
  - a little of both
  - many things are checked statically

    Object x = new Object();

    x.println(“hello”);   // No such method error at compile time
  - but other things are checked dynamically

    Object o = new Object();

    String s = (String) o; // No compiler warning, fails at run time

    // (Some Java compilers may be smart enough to warn about above cast)

Tradeoffs?

Static types	Dynamic types
more work to do when writing code	less work when writing code
helps prevent some subtle errors	can use objects incorrectly and not realize until execution
fewer programs type check	more programs type check