CMSC330

Summer 2009
Kinga Dobolyi

Introduction to Ruby

Previous Lecture

- Many types of programming languages
  - Imperative, functional, logical, OO, scripting
- Many programming language attributes
  - Clear, orthogonal, natural…
- Programming language implementation
  - Compiled, interpreted

This lecture

- Scripting languages
- Ruby language
  - Implicit variable declarations
  - Many control statements
  - Classes & objects
  - Strings

But first, from last time

- Higher-order functions:
  - take one or more functions as an input, or
  - output a function
  - ```
    let rec map f = function [] -> []
      | x::l -> (f x)::(map f l)
    ```
    - Takes function f as an argument, and returns a function
    - Same as
      ```
      let rec map f somelist = match somelist with
      | [] -> []
      | x::xs -> (f x)::(map f xs)
      ```


What is Ruby?

- Ruby is an object-oriented, imperative scripting language
  - "Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines."

    –Yukihiro Matsumoto ("Matz")

Applications of Scripting Languages

- Scripting languages have many uses
  - Automating system administration
  - Automating user tasks
  - Quick-and-dirty development
- Major application :
  Text processing


      ```

Output from Command-Line Tool

```
% wc *
271      674     5323     AGT.c
100      302     2219     AGT.h
117     1459   238788     AGT.o
1874    5428    47461     AGT_defs.c
1375    4307    53667     AGT_defs.h
371      884     9883     AGT_parent.c
810     2328    24589     AGT_print.c
640     2070    22520     AGT_types.h
285      946     7081     AGT_utils.c
59       274     2158     AGT_utils.h
50       400    28756     AGT_utils.o
966     2757    25872     Makefile
270      725     5579     Makefile.am
966     2743    27220     Makefile.in
38       175     1158     alloca.c
2035    6516    67721     alloctypes.c
96       350     3286     alloctypes.h
104     1051    66888     alloctypes.o
...
```

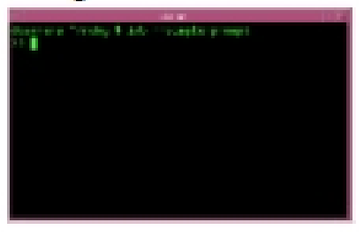

Climate Data for IAD in August, 2005

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

Raw Census 2000 Data for DC

```
u100_3,DC,000,01,0000001,572059,72268,572059,12.6,572059,572059,572059,0,0,
0,0,572059,175306,343213,2006,14762,383,21728,14661,572059,527048,15961
7,240061,1560,14605,291,1628,10272,45015,16699,2152,646,157,92,20090,42
89,572059,268827,3262,3048,3170,3241,2504,2286,2270,2475,2939,2647,2525
,3044,2828,2813,2749,2752,2933,2703,4056,5501,5217,4969,13555,24995,242
16,23726,20721,18803,16523,12318,4345,5810,3423,4690,7105,5738,3260,234
7,303232,3329,3057,2935,3429,3326,3456,3257,3754,3192,3523,3336,3276,29
89,2838,2824,2624,2807,2871,4941,6588,5625,5563,17177,27475,24277,22818
,21319,20851,19117,15260,5066,6708,4257,6117,10741,9427,6807,6175,57205
9,536373,370675,115963,55603,40360,57949,128440,122518,3754,3168,22448,
9367,6638,14110,14160,165698,61048,47684,13355,71578,60875,10703,33071,
35686,7573,28113,348590,108569,47634,60875,140021,115963,58050,21654,34
396,57913,10355,6065,6230,47558,25229,22329,24058,13355,10703,70088,657
27,27112,21782,12267,9475,9722,2572,2214,760,28625,8207,7469,728,19185,
18172,1013,1233,4351,3610,741,348590,189456,94221,46274,21463,24831,479
87,8705,3979,4726,39242,25175,14067,105225,82928,22307,48194,21782,1177
6,211,11565,8966,1650,86,1564,8316,56,8262,27392,25661,1751,348590,1159
63,4999,22466,26165,24062,16529,12809,7594,1739,132627,11670,32465,2322
5,21661,16238,12785,10563,8034,248590,115963,68738,28914,19259,10212,67
68,3992,132627,108569,19284,2713,1209,509,218,125
...
```


Running Ruby



- Execute script in file
  - `ruby -w filename.rb`
  - w option same as verbose
- Interactive Ruby shell
  - `irb` (image at right)
  - 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
```

- Executable

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")
```

line break separates expressions (can also use ";" to be safe)

no special main( ) function or method

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;
```

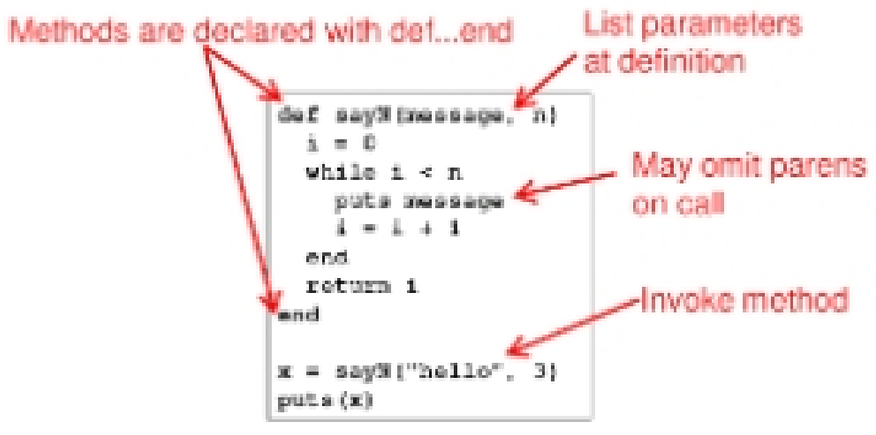
- In Ruby, variables are implicitly declared
  - First use of a variable declares it and determines type

```
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 names
Forces programmer to document types	Figures out types of variables automatically

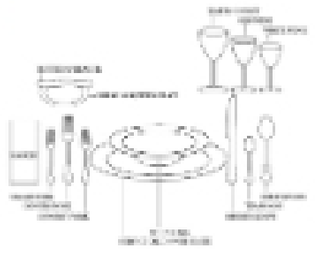
Methods in Ruby



(Methods should begin with lowercase letter and be defined before they are called)

Method terminology

- Formal parameters
  - Parameters used in the body of the method
  - `message`, `n` in our example
- Actual parameters
  - Arguments passed in to the method at a call
  - `"hello"`, `3` in our example



Control Statements

- A control statement is one that affects which instruction is executed next
  - We've seen two so far in Ruby
    - `while` and function call
- Ruby also has 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
```


Guard Statements

- The guard of a conditional is the expression determines which branch is taken
  - `if grade >= 90 then …`
  - The true branch is taken if it does not evaluate to
    - `false`
    - `nil`
  - Warning: 0 is not false!