CMSC 330: Organization of Programming Languages

Introduction to Ruby:
Declarations, Types, Control
Ruby

- An object-oriented, imperative, dynamically typed (scripting) language
  - Similar to other scripting languages (e.g., Python)
  - Notable in being fully object-oriented, and embracing higher-order programming style
    - Functions taking function(al code) as arguments

- Created in 1993 by Yukihiro Matsumoto (Matz)
  - “Ruby is designed to make programmers happy”

- Adopted by Ruby on Rails web programming framework in 2005 (a key to Ruby’s popularity)
Books on Ruby

• Earlier version of Thomas book available on web
  ➢ See course web page
Applications of Scripting Languages

- Scripting languages have many uses
  - Automating system administration
  - Automating user tasks
  - Quick-and-dirty development

- Motivating 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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6A</th>
<th>6B</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2MIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPTH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSBL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S-S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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,000,01,0000001,572059,72264,572059,12.6,572059,572059,572059,0,0,0,0,572059,175306,343213,2006,14762,383,21728,14661,572059,527044,158617,340061,1560,14605,291,1638,10272,45015,16689,3152,446,157,92,20090,4389,572059,268827,3362,3048,3170,3241,3504,3286,3270,3475,3939,3647,3525,3044,2928,2913,2769,2752,2933,2703,4056,5501,5217,4969,13555,24995,24216,23726,20721,18802,16523,12318,4345,5810,3423,4690,7105,5739,3260,2347,303232,3329,3057,2935,3429,3326,3456,3257,3754,3192,3523,3336,3276,2989,2838,2824,2624,2807,2871,4941,6588,5625,5563,17177,27475,24377,22818,21319,20851,19117,15260,5066,6708,4257,6117,10741,9427,6807,6175,572059,536373,370675,115963,55603,60360,57949,129440,122518,3754,3168,22448,9967,4638,14110,16160,165698,61049,47694,13355,71578,60875,10703,33071,35686,7573,28113,248590,108569,47694,60875,140021,115963,58050,21654,36396,57913,10355,4065,6290,47558,25229,23329,24058,13355,10703,70088,65737,37112,21742,12267,9475,9723,2573,2314,760,28625,8207,7469,738,19185,18172,1013,1233,4351,3610,741,248590,199456,94221,46274,21443,24831,47947,8705,3979,4726,39242,25175,14067,105235,82928,22307,49134,21742,11776,211,11565,9966,1650,86,1564,8316,54,8262,27392,25641,1751,248590,1159,63,4999,22466,26165,24062,16529,12409,7594,1739,132627,11670,32445,23225,21661,16234,12795,10563,4034,248590,115963,48738,28914,19259,10312,4748,3992,132627,108569,19284,2713,1209,509,218,125

...
Ruby started with special purpose, but has grown into a general-purpose language
  • As have related languages, like Python and Perl
    ➢ The Swedish pension system was once written in Perl!
But Ruby has distinctive features when compared to traditional general-purpose languages
  • Such as lightweight syntax, dynamic typing, evaluating code in strings, …
We will call them scripting languages, still, but also dynamic languages
A Simple Example

Let’s start with a simple Ruby program

```ruby
# This is a ruby program
x = 1
n = 5
while n > 0
  x = x * n
  n = n - 1
end
print(x)
print("\n")
```

```
ruby1.rb: 9
```

```
% ruby -w ruby1.rb
120
% 
```
# This is a ruby program
x = 1
n = 5
while n > 0
    x = x * n
    n = n - 1
end
print(x)
print("\n")
Run Ruby, Run

There are two basic 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
  - Ruby filenames should end with ‘.rb’ extension
- **irb** – launch interactive Ruby shell
  - Can type in Ruby programs one line at a time, and watch as each line is executed
    ```ruby
    irb(main):001:0> 3+4
    => 7
    ```
  - Can load Ruby programs via `load` command
    - *Form:* `load string`
    - String must be name of file containing Ruby program
    - E.g.: `load ‘foo.rb’`

Ruby is installed on Grace cluster
Some Ruby Language Features

- Implicit declarations
  - Java, C have explicit declarations

- Dynamic typing
  - Java, C have (mostly) static typing

- Everything is an object
  - No distinction between objects and primitive data
  - Even “null” is an object (called `nil` in Ruby), as are classes

- No outside access to private object state
  - Must use getters, setters

- No method overloading

- Class-based and Mixin inheritance
Implicit vs. Explicit Declarations

In Ruby, variables are implicitly declared
- First use of a variable declares it and determines type
  \[ x = 37; \] // no declaration needed – created when assigned to
  \[ y = x + 5 \]
  - \( x, y \) now exist, are integers

Java and C/C++ use explicit variable declarations
- Variables are named and typed before they are used
  \[ \text{int } x, y; \] // declaration
  \[ x = 37; \] // use
  \[ y = x + 5; \] // use
# Tradeoffs?

<table>
<thead>
<tr>
<th>Explicit Declarations</th>
<th>Implicit Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td>More text to type</td>
<td>Less text to type</td>
</tr>
<tr>
<td>Helps prevent typos</td>
<td>Easy to mistype variable name</td>
</tr>
</tbody>
</table>

```python
var = 37
If (rare-condition)
    y = vsr + 5
```

**Typo!**

Only caught when this line is actually run. Bug could be latent for quite a while.
Static Type Checking (Static Typing)

Before program is run
- Types of all expressions are determined
- Disallowed operations cause compile-time error
  - Cannot run the program

Static types are often explicit (aka manifest)
- Specified in text (at variable declaration)
  - C, C++, Java, C#
- But may also be inferred – compiler determines type based on usage
  - OCaml, C# and Go (limited)
Dynamic Type Checking

- During program execution
  - Can determine type from run-time value
  - Type is checked before use
  - Disallowed operations cause run-time exception
    - Type errors may be latent in code for a long time

- Dynamic types are not manifest
  - Variables are just introduced/used without types
  - Examples
    - Ruby, Python, Javascript, Lisp
Static and Dynamic Typing

- Ruby is dynamically typed, C is statically typed

```ruby
# Ruby
x = 3
x = "foo"  # gives x a new type
x.foo      # NoMethodError
# at runtime
```

```c
/* C */
int x;
x = 3;
x = "foo"; /* not allowed */
/* program doesn’t compile */
```

- Notes
  - Can always run the Ruby program; may fail when run
  - C variables declared, with types
    - Ruby variables declared *implicitly*
    - Implicit declarations most natural with dynamic typing
Tradeoffs?

- **Static type checking**
  - More work for programmer (at first)
    - Catches more (and subtle) errors at compile time
  - Precludes some correct programs
    - May require a contorted rewrite
  - More efficient code (fewer run-time checks)

- **Dynamic type checking**
  - Less work for programmer (at first)
    - Delays some errors to run time
  - Allows more programs
    - Including ones that will fail
  - Less efficient code (more run-time checks)
Java: *Mostly* Static Typing

- In Java, types are mostly checked statically
  ```java
  Object x = new Object();
x.println(“hello”);  // No such method error at compile time
  ```

- But sometimes checks occur at run-time
  ```java
  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)
  ```
Quiz 1: Get out your clickers!

- True or false: This program has a type error

```ruby
# Ruby
x = 3
y = "foo"
x = y
```

A. True
B. False
Quiz 1: Get out your clickers!

» True or false: This program has a type error

```ruby
# Ruby
x = 3
y = "foo"
x = y
```

A. True  
B. False  

» True or false: This program has a type error

```c
/* C */
void foo() {
    int x = 3;
    char *y = "foo";
    x = y;
}
```

A. True  
B. False
Quiz 1: Get out your clickers!

- **True or false:** This program has a type error

```ruby
# Ruby
x = 3
y = "foo"
x = y
```

- **True or false:** This program has a type error

```c
/* C */
void foo() {
    int x = 3;
    char *y = "foo";
    x = y;
}
```
Control Statements in Ruby

- A control statement is one that affects which instruction is executed next
  - While loops
  - Conditionals

```ruby
i = 0
while i < n
  i = i + 1
end
```

```ruby
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
```
Conditionals and Loops Must End!

- All Ruby conditional and looping statements must be terminated with the `end` keyword.

Examples

- `if grade >= 90 then
   puts "You got an A"
end`

- `if grade >= 90 then
   puts "You got an A"
else
   puts "No A, sorry"
end`

- `i = 0
  while i < n
    i = i + 1
  end`
What is True?

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

```c
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”

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

- `until cond body end`
  - Same as “while not cond body end”

```ruby
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? Why or why not?
    - **Good**: can make program more readable, expressing programs more directly. In natural language, many ways to say the same thing, which supports brevity and adds style.
    - **Bad**: many ways to do the same thing may lead to confusion and hurt maintainability (if future programmers don’t understand all styles)
Quiz 2: What is the output?

```ruby
x = 0
if x then
  puts "true"
elsif x == 0 then
  puts "== 0"
else
  puts "false"
end
```

A. Nothing – there’s an error
B. "true"
C. "== 0"
D. "false"
Quiz 2: What is the output?

```ruby
x = 0
if x then
    puts "true"
elsif x == 0 then
    puts "== 0"
else
    puts "false"
end
```

x is neither false nor nil so the first guard is satisfied

A. Nothing – there’s an error
B. “true”
C. “== 0”
D. “false”
Other Useful Control Statements

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

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

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

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

case x
  when 1, 3..5
    code block (details later)
  when 2, 6..8
    does not need break
end

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