Overview of Ruby
Reminders and Announcements

• Make sure you are “in” the class
  – Registered on Testudo
  – Submit server account
• Project 1 is posted
  – It is due on June 6
  – Start immediately
• Lecture slides will be posted after each class
• Use the website resources and course forum
• Leave 24 hours for email responses
Review

• Why study programming languages?
• Types of programming languages
• Compiled vs. interpreted languages
• Standardization and internationalization
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”)
Applications of Scripting Languages

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

• Major application: Text processing

Go away or I will replace you with a very small shell script.
## 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>AVG</th>
<th>MX</th>
<th>2MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87</td>
<td>66</td>
<td>77</td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>67</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>93</td>
<td>69</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>95</td>
<td>69</td>
<td>82</td>
</tr>
<tr>
<td>5</td>
<td>94</td>
<td>73</td>
<td>84</td>
</tr>
<tr>
<td>6</td>
<td>89</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>89</td>
<td>69</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>86</td>
<td>70</td>
<td>78</td>
</tr>
<tr>
<td>9</td>
<td>76</td>
<td>70</td>
<td>73</td>
</tr>
<tr>
<td>10</td>
<td>87</td>
<td>71</td>
<td>79</td>
</tr>
</tbody>
</table>

### Table Columns
- **DY**: Day
- **AVG**: Average Temperature
- **MX**: Maximum Temperature
- **2MIN**: Two-Minute Temperature
- **AVG DEP**: Average Depth
- **HDD**: Heating Degree Days
- **CDD**: Cooling Degree Days
- **WTR**: Water
- **SNW**: Snow
- **DPTH**: Drought Depth
- **SPD**: Speed
- **SPD DIR**: Speed Direction
- **MIN**: Minimum Temperature
- **PSBL**: Probability
- **S-S WX**: Sea Surface Weather
- **SPD DR**: Speed Details

...
### Raw Census 2000 Data for DC

```
000, 01, 0000001, 572059, 72264, 572059, 72264, 12.6, 572059, 572059, 572059, 0, 0, 0, 0, 0.57, 2059, 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, 175306, 3362, 3048, 3170, 3241, 3504, 3286, 3270, 3475, 3939, 3647, 3525, 3044, 2928, 2913, 2769, 2752, 2933, 2703, 4056, 5501, 5217, 4969, 13555, 24995, 254216, 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, 5629, 5563, 17177, 27475, 24377, 22818, 21319, 20851, 19117, 15260, 5066, 6708, 4257, 6117, 10741, 9427, 6807, 6175, 572059, 573673, 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, 22329, 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, 115963, 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...
```
A Simple Example

Let’s start with a simple Ruby program

test.rb:

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

```bash
% ruby test.rb
42
%
```
Language Basics

comments begin with #, go to end of line

variables need not be declared

no special main() function or method

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

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

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

• `./filename`  # run program
  – The first line 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 of Ruby interpreter
    • May be safer: `#!/usr/bin/env 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 vs. Implicit

- Overhead
- Ease of programming
- Error-proneness
- Documentation
Methods in Ruby

Methods are declared with `def...end`

List parameters at definition

Invoke method

May omit parens on call

```
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 lowercase letter and be defined before they are called)
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 instruction is executed next
  – We’ve seen two so far in Ruby
    • `while` and function call

• Ruby also has conditionals

```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
```
What is True?

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

```plaintext
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 \textit{cond} then \textit{stmt-f} else \textit{stmt-t} end
  – Same as “if not \textit{cond} then \textit{stmt-t} else \textit{stmt-f} end”

\begin{verbatim}
unless grade < 90 then
  puts "You got an A"
else
  unless grade < 80 then
    puts "You got a B"
  end
end
\end{verbatim}

• until \textit{cond} \textit{body} end
  – Same as “while not \textit{cond} \textit{body} end”

\begin{verbatim}
until i >= n
  puts message
  i = i + 1
end
\end{verbatim}
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
Other Useful Ruby Constructs

```ruby
IO.foreach(filename) do |line|
  puts line
end

for e in [4, "text", 3.45]
  puts e.to_s
end

name = "Bob"
puts "Hi, #{name}"
```

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

while i > n
  puts "hi"
  break
  next
  redo
end
```
Two Ways to Create Blocks

```ruby
names = ["Jim", "Sue", "Bill"]
names.each do |name|
  puts name
end
```

```ruby
names = ["Jim", "Sue", "Bill"]
names.each { |name|
  puts name
}
```