CMSC 330: Organization of Programming Languages

Introduction to Ruby
Ruby

- An object-oriented, imperative, dynamically typed (scripting) language
  - Created in 1993 by Yukihiro Matsumoto (Matz)
  - “Ruby is designed to make programmers happy”
  - Core of Ruby on Rails web programming framework (a key to its popularity)
  - Similar in flavor to many other scripting languages
    - Much cleaner than perl
  - Full object-orientation (even primitives are objects!)
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  5873 Makefile
   270  725   5578 Makefile.am
   866 2743  57320 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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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></td>
<td>AVG</td>
<td>MX</td>
<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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<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>1</td>
<td>87</td>
<td>66</td>
<td>77</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>2.5</td>
<td>9</td>
<td>200</td>
<td>M</td>
<td>M</td>
<td>7</td>
<td>18</td>
<td>12</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>92</td>
<td>67</td>
<td>80</td>
<td>4</td>
<td>0</td>
<td>15</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>3.5</td>
<td>10</td>
<td>10</td>
<td>M</td>
<td>M</td>
<td>3</td>
<td>18</td>
<td>17</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>93</td>
<td>69</td>
<td>81</td>
<td>5</td>
<td>0</td>
<td>16</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>4.1</td>
<td>13</td>
<td>360</td>
<td>M</td>
<td>M</td>
<td>2</td>
<td>18</td>
<td>17</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>95</td>
<td>69</td>
<td>82</td>
<td>6</td>
<td>0</td>
<td>17</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>3.6</td>
<td>9</td>
<td>310</td>
<td>M</td>
<td>M</td>
<td>3</td>
<td>18</td>
<td>12</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>94</td>
<td>73</td>
<td>84</td>
<td>8</td>
<td>0</td>
<td>19</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>5.9</td>
<td>18</td>
<td>10</td>
<td>M</td>
<td>M</td>
<td>3</td>
<td>18</td>
<td>25</td>
<td>360</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>89</td>
<td>70</td>
<td>80</td>
<td>4</td>
<td>0</td>
<td>15</td>
<td>0.02</td>
<td>0.0</td>
<td>0</td>
<td>5.3</td>
<td>20</td>
<td>200</td>
<td>M</td>
<td>M</td>
<td>6</td>
<td>138</td>
<td>23</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>89</td>
<td>69</td>
<td>79</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>3.6</td>
<td>14</td>
<td>200</td>
<td>M</td>
<td>M</td>
<td>7</td>
<td>1</td>
<td>16</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>86</td>
<td>70</td>
<td>78</td>
<td>3</td>
<td>0</td>
<td>13</td>
<td>0.74</td>
<td>0.0</td>
<td>0</td>
<td>4.4</td>
<td>17</td>
<td>150</td>
<td>M</td>
<td>M</td>
<td>10</td>
<td>18</td>
<td>23</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>76</td>
<td>70</td>
<td>73</td>
<td>-2</td>
<td>0</td>
<td>8</td>
<td>0.19</td>
<td>0.0</td>
<td>0</td>
<td>4.1</td>
<td>9</td>
<td>90</td>
<td>M</td>
<td>M</td>
<td>9</td>
<td>18</td>
<td>13</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>87</td>
<td>71</td>
<td>79</td>
<td>4</td>
<td>0</td>
<td>14</td>
<td>0.00</td>
<td>0.0</td>
<td>0</td>
<td>2.3</td>
<td>8</td>
<td>260</td>
<td>M</td>
<td>M</td>
<td>8</td>
<td>1</td>
<td>10</td>
<td>210</td>
<td></td>
</tr>
</tbody>
</table>

...
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, 9536373, 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, 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...
A Simple Example

Let’s start with a simple Ruby program

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

```
ruby -w ruby1.rb
42
```
Language Basics

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

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
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
    - `int x, y;` // declaration
    - `x = 37;` // use
    - `y = x + 5;` // use
Tradeoffs?

**Explicit Declarations**

- More text to type
- Helps prevent typos

**Implicit Declarations**

- Less text to type
- Easy to mistype variable name

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

C

```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
```

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

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

- `i = 0
  while i < n
    i = i + 1
  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
  ...
  ```

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

\[
\begin{align*}
x &= 0 \\
\text{if } x \text{ then} & \quad \text{(this condition is satisfied)} \\
\quad & \text{puts "true"} \\
\text{elsif } x == 0 \text{ then} & \quad \text{(this condition is satisfied)} \\
\quad & \text{puts "== 0"} \\
\text{else} & \quad \text{(this condition is not satisfied)} \\
\quad & \text{puts "false"} \\
\text{end} \\
\end{align*}
\]

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

\[ x \text{ is neither } false \text{ nor } nil \text{ so the first guard is satisfied} \]
Methods in Ruby

Methods are declared with `def...end`.

List parameters at definition.

May omit parens on call.

Invoke method.

Like print, but adds newline.

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

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

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

Variable names that begin with uppercase letter are **constants** (only assigned once).
Terminology

- **Formal** parameters
  - Variable parameters used in the method
  - `def sayN(message, n)` in our example

- **Actual** arguments
  - Values passed in to the method at a call
  - `x = sayN("hello", 3)` in our example

- **Top-level** methods are “global”
  - Not part of a class. `sayN` is a top-level method.
Method Return Values

- Value of the `return` is the value of the last executed statement in the method
  - These are the same:

```ruby
def add_three(x)
    return x+3
end
```

- Methods can return multiple results (as an Array)

```ruby
def dup(x)
    return x,x
end
```
Everything is an Object

- All values are (references to) objects
  - Java/C/C++ distinguish primitives from objects
- Objects communicate via method calls
- Each object has its own (private) state
- Every object is an instance of a class
  - An object’s class determines its behavior:
  - The class contains method and field definitions
    - Both instance fields and per-class (“static”) fields
Everything is an Object

Examples

• (-4).abs
  ➢ integers are instances of class Fixnum

• 3 + 4
  ➢ infix notation for “invoke the + method of 3 on argument 4”

• "programming".length
  ➢ strings are instances of String

• String.new
  ➢ classes are objects with a new method

• 4.13.class
  ➢ use the class method to get the class for an object
  ➢ floating point numbers are instances of Float
Classes

- 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`
Objects and Classes

- Objects are data
- Classes are types (the kind of data which things are)
- Classes are also objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Class (aka type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Integer</td>
</tr>
<tr>
<td>-3.30</td>
<td>Float</td>
</tr>
<tr>
<td>&quot;CMSC 330&quot;</td>
<td>String</td>
</tr>
<tr>
<td>String.new</td>
<td>String</td>
</tr>
<tr>
<td>['a', 'b', 'c']</td>
<td>Array</td>
</tr>
<tr>
<td>Integer</td>
<td>Class</td>
</tr>
</tbody>
</table>

- Integer, Float, and String are objects of type Class
  - So is Class itself!
The nil Object

- Ruby uses a special object **nil**
  - All uninitialized fields set to **nil** (@ prefix used for fields)
    
    ```
    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 methods like to_s, but not other methods
    
    ```
    irb(main):006:0> nil + 2
    NoMethodError: undefined method `+' for nil:NilClass
    ```
Quiz 3

What is the type of variable $x$ at the end of the following program?

```
p = nil
x = 3
if p then
   x = nil
else
   x = "hello"
end
```

A. String
B. Integer
C. NilClass
D. *Nothing* – there’s a type error
Quiz 3

What is the type of variable \( x \) at the end of the following program?

\[
\begin{array}{l}
p = \text{nil} \\
x = 3 \\
\text{if } p \text{ then} \\
\quad x = \text{nil} \\
\text{else} \\
\quad x = \text{“hello”} \\
\end{array}
\]

A. String

B. Integer

C. NilClass

D. *Nothing* – there’s a type error
Creating Strings in Ruby

- Substitution in double-quoted strings with \{\ }
  - course = "330"; msg = "Welcome to #{course}"
  - "It is now #{Time.new}"
  - The contents of \{\ } may be an arbitrary expression
  - Can also use single-quote as delimiter
    - No expression substitution, fewer escaping characters

- Here-documents
  
s = <<END
  This is a text message on multiple lines
  and typing \n is annoying
  END
Creating Strings in Ruby (cont.)

- Ruby has `printf` and `sprintf`
  - `printf("Hello, %s\n", name);`
  - `sprintf("%d: %s", count, Time.now)`
    - Returns a String

- `to_s` returns a **String** representation of an object
  - Can be invoked implicitly – write `puts(p)` instead of `puts(p.to_s)`
    - Like Java’s `toString()`

- `inspect` converts any object to a string
  ```ruby
  irb(main):033:0> p.inspect
  => "#<Point:0x54574 @y=4, @x=7>"
  ```
Standard Library: String

- The **String** class has many useful methods
  - `s.length` # length of string
  - `s1 == s2` # structural equality (string contents)
  - `s = "A line\n"; s.chomp` # returns "A line"
    - Return new string with s's contents except newline at end of line removed
  - `s = "A line\n"; s.chomp!`
    - Destructively removes newline from s
    - **Convention:** methods ending in ! modify the object
    - **Another convention:** methods ending in ? observe the object
Defining Your Own Classes

class Point
  def initialize(x, y)
    @x = x
    @y = y
  end

  def add_x(x)
    @x += x
  end

  def to_s
    return "(\(\@x.to_s\) , \(\@y.to_s\) )"
  end
end

p = Point.new(3, 4)
p.add_x(4)
puts(p.to_s)
No Outside Access To Internal State

- Instance variables (with @) can be directly accessed only by instance methods.
- Outside class, they require accessors:

  A typical getter
  ```ruby
def x
    @x
  end
  ```
  A typical setter
  ```ruby
def x= (value)
    @x = value
  end
  ```

- Very common, so Ruby provides a shortcut:
  ```ruby
class ClassWithXandY
  attr_accessor :x, :y
end
```

  Says to generate the x= and x and y= and y methods.
No Method Overloading in Ruby

- Thus there can only be one `initialize` method
  - A typical Java class might have two or more constructors
- No overloading of methods in general
  - You can code up your own overloading by using a variable number of arguments, and checking at run-time the number/types of arguments
- Ruby does issue an exception or warning if a class defines more than one `initialize` method
  - But last `initialize` method defined is the valid one
Quiz 4: What is the output?

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "I smelled #{thing} for #{dur} seconds"
  end
end

fido = Dog.new
puts fido.smell("Alice")
```

A. I smelled Alice for nil seconds
B. Error
C. I smelled #{thing}
D. I smelled Alice
Quiz 4: What is the output?

class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "I smelled #{thing} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smell(”Alice”)

A. I smelled Alice for nil seconds
B. Error
C. I smelled #{thing}
D. I smelled Alice
Quiz 5: What is the output?

class Dog
  def smell(thing)
    "I smelled #{thing}"n
  end
  def smell(thing,dur)
    "I smelled #{thing} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smell(“Alice”,3)

A. I smelled Alice for 3 seconds
B. Error
C. I smelled #{thing} for #{dur} seconds
D. I smelled Alice for 3 seconds
Quiz 5: What is the output?

class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "I smelled #{thing} for #{dur} seconds"
  end
end

fido = Dog.new
puts fido.smell("Alice",3)

A. I smelled Alice for 3 seconds
B. Error
C. I smelled #{thing} for #{dur} seconds
D. I smelled Alice for 3 seconds
Recall that every class inherits from **Object**

```ruby
class A #< Object
  def add(x)
    return x + 1
  end
end

class B < A
  def add(y)
    return (super(y) + 1)
  end
end

b = B.new
puts(b.add(3))
```

- **extend superclass**
- **invoke add method of parent**

```
b.is_a? A
true
b.instance_of? A
false
```
Quiz 6: What is the output?

class Gunslinger
  def initialize(name)
    @name = name
  end

  def full_name
    "#{@name}"
  end
end

class Outlaw < Gunslinger
  def full_name
    "Dirty, no good #{super}"
  end
end

d = Outlaw.new("Billy the Kid")
puts d.full_name

A. Dirty, no good
B. Dirty, no good Billy the kid
C. Billy the Kid
D. Error
Quiz 6: What is the output?

class Gunslinger
  def initialize(name)
    @name = name
  end
  def full_name
    "#{@name}"
  end
end
class Outlaw < Gunslinger
  def full_name
    "Dirty, no good #{super}"
  end
end
d = Outlaw.new("Billy the Kid")
puts d.full_name

A. Dirty, no good
B. Dirty, no good Billy the kid
C. Billy the Kid
D. Error
Global Variables in Ruby

- Ruby has two kinds of global variables
  - Class variables beginning with `@@` (**static** in Java)
  - Global variables across classes beginning with `$`

```ruby
class Global
  @@x = 0
  def Global.inc
    @@x = @@x + 1; $x = $x + 1
  end
  def Global.get
    return @@x
  end
end

$x = 0
Global.inc
$x = $x + 1
Global.inc
puts(Global.get)
puts($x)
```

define a class (**“singleton”**) method
Quiz 7: What is the output?

```ruby
class Animal
  def initialize(h, w)
    @@h = h
    @w = w
  end
  def measure()
    return @@h + @w
  end
End

giraffe = Animal.new(1,2)
elephant = Animal.new(3,4)
puts giraffe.measure()
```

A. 0
B. 3
C. 5
D. 7
Quiz 7: What is the output?

class Animal
  def initialize(h, w)
    @@h = h
    @w = w
  end
  def measure()
    return @@h + @w
  end
End

giraffe = Animal.new(1,2)
elephant = Animal.new(3,4)
puts giraffe.measure()

A. 0
B. 3
C. 5
D. 7
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 some class (say, MyClass) containing a method
    - public static void main(String[] args)
  - When you run java MyClass, the main method of class MyClass is invoked
A Ruby Program is...

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

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

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

defines a method of Object (i.e., top-level methods belong to Object)

invokes `self.sayN`

invokes `self.puts` (part of Object)