CMSC 330: Organization of Programming Languages

Introduction to Ruby
Ruby

- An **object-oriented, imperative, dynamically typed (scripting) language**
  - Similar to Python, Perl
  - Fully object-oriented
- 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
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#, Rust, 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
  - **Note**: Ruby v3 adds support for static types, mixed with its native dynamic ones. We’ll discuss this more, later in the course.
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 */
```
## Tradeoffs?

<table>
<thead>
<tr>
<th>Static type checking</th>
<th>Dynamic type checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>More work for programmer (at first)</td>
<td>Less work for programmer (at first)</td>
</tr>
<tr>
<td>Catches more (and subtle) errors at compile time</td>
<td>Delays some errors to run time</td>
</tr>
<tr>
<td>Precludes some correct programs</td>
<td>Allows more programs (including ones that will fail)</td>
</tr>
<tr>
<td>More efficient code (fewer run-time checks)</td>
<td>Less efficient code (more run-time checks)</td>
</tr>
</tbody>
</table>
Java: Mostly Static Typing

• In Java, types are mostly checked statically
  Object x = new Object();
  x.println("hello");  // No such method error at compile time

• But sometimes checks occur at run-time
  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

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

```ruby
# Ruby
x = "hello"
y = 2.5
y = x
```

A. True
B. False
Quiz 1

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

```ruby
# Ruby
x = "hello"
y = 2.5
y = x
```

A. True
B. False
Quiz 2

• True or false: This program has a type error

```c
/* C */
void foo() {
    int a = 10;
    char *b = "hello";
    a = b;
}
```

A. True
B. False
Quiz 2

- True or false: This program has a type error

```
/* C */
void foo() {
    int a = 10;
    char *b = "hello";
    a = b;
}
```

A. True
B. False
Control Statements in Ruby

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

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

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

  **Guard**

• **True**: anything except
  • false
  • nil

• Warning to C programmers: 0 is not false!
Quiz 3: What is the output?

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

A. Nothing – there’s an error
B. “false”
C. “== 0”
D. “true”
Quiz 3: 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. “false”
C. “== 0”
D. “true”
In Ruby, everything is an Object

• Ruby is object-oriented
• 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

> 1.class Integer

> 1.methods [:to_s, :to_i, :abs, ...]

Object is the superclass of every class

> 1.class.ancestors [Integer, Numeric, Comparable, Object, Kernel, BasicObject]
Objects Communicate via Method Calls

+ is a method of the Integer class

\[
\begin{align*}
1 + 2 & \Rightarrow 3 \\
1.+(2) & \Rightarrow 3 \\
\end{align*}
\]

1 + 2 is **syntactic sugar** for 1.+(2)

\[
\begin{align*}
1.\text{add}(2) & \Rightarrow 1.+(2) \Rightarrow 1 + 2 \\
\end{align*}
\]

1.to_s = ”1”
1.to_s() = ”1”  no parens needed if no args
The nil Object

- Ruby uses **nil** (not null)
  - All uninitialized fields set to nil
  - `irb(main):004:0> @x`  
    => nil
- **nil** is an object of class **NilClass**
  - Unlike null in Java, which is a non-object
  - **nil** is 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
Classes are Objects too

> nil.class
NilClass

> 2.5.class
Float

> true.class
TrueClass

> Float.class
Class
First-class Classes

- Since classes are objects, you can manipulate them however you like
  - Here, the type of $y$ depends on $p$
    - Either a String or a Time object

```ruby
if p then
  x = String
else
  x = Time
end
y = x.new
```
What is the type of variable $x$ at the end of the following program?

- A. Integer
- B. NilClass
- C. String
- D. *Nothing* – there’s a type error
Quiz 4

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

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

```ruby
p = nil
x = 3
if p then
  x = "hello"
else
  x = nil
end
```
Strings in Ruby have class String

- “hello”.class == 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 minus any trailing newline
- 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
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

33
Creating Strings in Ruby (cont.)

- **sprintf**
  
  ```ruby
  count = 100
  s = sprintf("%d: %s", count, Time.now)
  => "100: 2021-01-27 19:56:06 -0500"
  ```

- **to_s** returns a **String** representation of an object
  
  - Like Java’s `toString()`

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

• Ruby symbols begin with a colon
  • :foo, :baz_42, :"Any string at all"
• Symbols are “interned” Strings,
• Symbols are more efficient than strings.
  • The same symbol is at the same physical address

```
"foo" == "foo"     # true
"foo".equal? "foo" # false
:foo == :foo       # true
:foo.equal? :foo   # true
```
Arrays and Hashes

• Ruby data structures are typically constructed from Arrays and Hashes
  • Built-in syntax for both
  • Each has a rich set of standard library methods
  • They are integrated/used by methods of other classes
Array

- Create an empty Array
  
  ```ruby
  t = Array.new
  x = []
  b = Array.new(3)  # b = [nil, nil, nil]
  b = Array.new(5, "a")  # b = ["a", "a", "a", "a", "a"]
  ```

- Arrays may be heterogeneous
  
  ```ruby
  a = [1, "foo", 2.14]
  ```
Array Index

> s = ["a","b","c", 1, 1.5, true]

<table>
<thead>
<tr>
<th></th>
<th>“a”</th>
<th>“b”</th>
<th>“c”</th>
<th>1</th>
<th>1.5</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td>-4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> s[0]
"a"

> s[-6]
"a"
Arrays Grow and Shrink

- **Arrays are growable**
  
  ```
  # b = [ ]; b[0] = 0; b[5] = 0; b
  => [0, nil, nil, nil, nil, 0]
  ```

- **Arrays can also shrink**
  
  - Contents shift left when you delete elements
  ```
  a = [1, 2, 3, 4, 5]
a.delete_at(3)  # delete at position 3; a = [1,2,3,5]
a.delete(2)    # delete element = 2; a = [1,3,5]
  ```
Two-Dimensional Array

```ruby
> a = Array.new(3) { Array.new(3) }
> a[1][1]=100
> a

[ [nil, nil, nil],
  [nil, 100, nil],
  [nil, nil, nil] ]
```
Some Array Operations

\[ a = [1, 2, 3, 4] \]
\[ b = [3, 4, 5, 6] \]

Adding two arrays
\[ a + b \Rightarrow [1, 2, 3, 4, 3, 4, 5, 6] \]

Union
\[ a \mid b \Rightarrow [1, 2, 3, 4, 5, 6] \]

Intersection
\[ a \& b \Rightarrow [3, 4] \]

Subtract
\[ a - b \Rightarrow [1, 2] \]
Arrays can model stacks and queues

```javascript
a = [1, 2, 3]
a.push("a")  # a = [1, 2, 3, "a"]
x = a.pop  # x = "a"
a.unshift("b")  # a = ["b", 1, 2, 3]
y = a.shift  # y = "b"
```

Note that `push`, `pop`, `shift`, and `unshift` all permanently modify the array.
Quiz 5: What is the output?

A. **Error**
B. 2
C. 3
D. 0

```python
a = [1,2,3]
a[1] = 0
a.shift
print a[1]
```
Quiz 5: What is the output?

A. Error
B. 2
C. 3
D. 0

```javascript
a = [1, 2, 3]
a[1] = 0
a.shift
print a[1]
```
Hash

- A **hash** acts like an *array*, whose elements can be indexed by *any kind* of value
  - Every Ruby object can be used as a hash key, because the Object class has a hash method

- Elements are referred to like array elements

```ruby
italy = Hash.new # or italy={}  
italy["population"] = 58103033  
italy[1861] = "independence"  
p = italy["population"] # pop is 58103033  
planet = italy["planet"] # planet is nil
```
Hash methods

- new(v) returns hash whose default value is v
  - h = Hash.new("fish");
  - h["go"] # returns "fish"

- values: returns array of a hash’s values
- keys: returns an array of a hash’s keys
- delete(k): deletes mapping with key k
- has_key?(k): is true if mapping with key k present
  - has_value?(v) is similar
Hash creation

Convenient syntax for creating literal hashes

- Use \{ key => value, ... \} to create hash table

```ruby
credits = {
  "cmsc131" => 4,
  "cmsc330" => 3,
}

x = credits["cmsc330"]  # x now 3
credits["cmsc311"] = 3
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmsc131</td>
<td>4</td>
</tr>
<tr>
<td>cmsc330</td>
<td>3</td>
</tr>
</tbody>
</table>
Hashes of Hashes

\[
\begin{align*}
\text{h} & = \text{Hash.new(0)} \\
\text{h}[1] & = \text{Hash.new(0)} \\
\text{h}[1][2] & = 5 \\
\text{h}[2] & = \text{Hash.new(0)} \\
\text{h}[2][1] & = 1 \\
\text{h}[3] & = \text{Hash.new(0)} \\
\text{h}[3][3] & = 3
\end{align*}
\]

\( h \) is \{ 1\=> \{2\=> 5\}, 2\=> \{1\=> 1\}, 3\=> \{3\=> 3\} \}
Quiz 6: What is the output?

A. Error
B. bar
C. bazbar
D. baznilbar

```python
a = {'foo': 'bar'}
a['bar'] = 'baz'
print(a[1])
print(a['foo'])
```
A. Error
B. bar
C. bazbar
D. baznilbar

```ruby
a = {"foo" => "bar"}
a["bar"] = "baz"
print a[1]
print a["foo"]
```

Quiz 6: What is the output?
Quiz 7: What is the output?

A. Green
B. (nothing)
C. Blue
D. Error

```python
a = { "Yellow" => [] }
a["Yellow"] = {}
a["Yellow"]["Red"] = ["Green", "Blue"]
print a["Yellow"]["Red"][1]
```
Quiz 7: What is the output?

A. Green  
B. (nothing)  
C. Blue  
D. Error

```ruby
a = { "Yellow" => [] }
a["Yellow"] = {}
a["Yellow"]["Red"] = ["Green", "Blue"]
print a["Yellow"]["Red"][1]
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Yellow&quot;</td>
<td>[]</td>
</tr>
<tr>
<td>&quot;Red&quot;</td>
<td>[&quot;Green&quot;, &quot;Blue&quot;]</td>
</tr>
</tbody>
</table>

```ruby
a["Yellow"]["Red"][1]
```
Methods in Ruby

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

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

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

puts(x)

List parameters at definition

May omit parens on call

Invoke method

Like print, but adds newline

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).

Note: Methods need not be part of a class.
Methods: 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
  ```

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

• Methods can return multiple results (as an Array)

  ```ruby
  def dup(x)
    return x,x
  end
  ```
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
    "(\(\) + @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

• An object’s 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
Defining Your Own Classes

```ruby
class Point
  def initialize(x)
    @x = x
  end
  def x=(x)
    @x = x
  end
  def x
    @x
  end
  private
  def prt
    "#{@x}"
  end
  # Make the below methods public
  public
  def to_s
    prt
  end
end

> p = Point.new(10)
#<Point:0x000007f8 @x=10>

> p.x= 100
100

> p.prt
NoMethodError
(private method `prt' called)
```
class Point
  def initialize(x)
    @x = x
  end

  def x=(x)
    @x = x
  end

  def x
    @x
  end

  private
  def prt
    "#{@x}" end

  # Make the below methods public
  public
  def to_s
    prt
  end
end

class Point
  attr_accessor :x
  attr_reader :y
  attr_writer :z

  private
  def prt
    "#{@x}, #{@y}" end

  # Make the below methods public
  public
  def to_s
    prt
  end
end
Quiz 8: What is the output?

A. I smelled Alice for nil seconds
B. I smelled #{thing}
C. I smelled Alice
D. Error

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smell("Alice",3)
```
Quiz 8: What is the output?

A. I smelled Alice for nil seconds
B. I smelled #{thing}
C. I smelled Alice
D. Error – call from Dog expected two args

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smell(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smell("Alice",3)
```
Quiz 9: What is the output?

A. I smelled Alice for seconds
B. I smelled #{thing} for #{dur} seconds
C. I smelled Alice for 3 seconds
D. Error

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smelltime(thing,dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smelltime("Alice",3)
```
Quiz 9: What is the output?

A. I smelled Alice for seconds
B. I smelled #{thing} for #{dur} seconds
C. I smelled Alice for 3 seconds
D. Error

```ruby
class Dog
  def smell(thing)
    "I smelled #{thing}"
  end
  def smelltime(thing, dur)
    "#{smell(thing)} for #{dur} seconds"
  end
end
fido = Dog.new
puts fido.smelltime("Alice", 3)
```
Update Existing Classes (Including Builtins!)

10.double => NoMethodError
(undefined method `double' for 10:Integer)

Add a method to the Integer class

class Integer
  def double
    self + self
  end
end

10.double => 20
Method naming style

- Names of methods that return `true` or `false` should end in `?`
- Names of methods that modify an object’s state should end in `!`

Example: suppose `x = [3,1,2]` (this is an array)
- `x.member? 3` returns true since 3 is in the array `x`
- `x.sort` returns a new array that is sorted
- `x.sort!` modifies `x` in place
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
Inheritance

- 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))
```

- `b.is_a? A` is true
- `b.instance_of? A` is false
Quiz 10: What is the output?

A. Dirty, no good Billy the kid
B. Dirty, no good
C. Billy the Kid
D. Error

```ruby
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
```
Quiz 10: What is the output?

A. Dirty, no good Billy the kid
B. Dirty, no good
C. Billy the Kid
D. Error

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
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
```

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

define a class (“singleton”) method
Quiz 8: What is the output?

A. 0  
B. 5  
C. 3  
D. 7

```ruby
class Rectangle
  def initialize(h, w)
    @@h = h
    @w = w
  end
  def measure()
    return @@h + @w
  end
end
r = Rectangle.new(1,2)
s = Rectangle.new(3,4)
puts r.measure()
```

Output:

```
7
```
Quiz 8: What is the output?

A. 0  
B. 5  
C. 3  
D. 7

```ruby
class Rectangle
  def initialize(h, w)
    @@h = h
    @w = w
  end
  def measure()
    return @@h + @w
  end
end
r = Rectangle.new(1,2)
s = Rectangle.new(3,4)
puts r.measure()
```

75
What is a Program?

- In C/C++, a program is...
  - A collection of declarations and definitions
  - With a distinguished function definition
    ```c
    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
    ```java
    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

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

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