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
  - \[
    \text{let rec map f = function} \[\] \rightarrow \[\]
    \quad | \ x::l \rightarrow (f \ x)::(map \ f \ l)
  \]
  - Takes function \(f\) as an argument, and returns a function
  - Same as
    \[
    \text{let rec map f somelist} = \text{match somelist with}
    \quad | \ [] \rightarrow \[]
    \quad | \ x::xs \rightarrow (f \ x)::(map \ f \ xs)
    \]

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 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
Explicit vs. Implicit Declarations

- Java and C/C++ use explicit variable declarations
  - Variables are named and typed before they are used
    \[
    \text{int } x, y; \quad x = 37; \quad y = x + 5;
    \]
- In Ruby, variables are implicitly declared
  - First use of a variable declares it and determines type
    \[
    x = 37; \quad y = x + 5;
    \]
  - \(x, y\) exist, will be integers

Tradeoffs?

Explicit Declarations
- Higher overhead
- Helps prevent typos

Implicit Declarations
- Lower overhead
- Easy to mistype variable names
- Forces programmer to document types
- Figures out types of variables automatically

Methods in Ruby

- Methods are declared with `def...end`
- Parameters are optional
- May omit parentheses on call

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

x = say("hello", 3)
ppts(x)

(Methods should begin with a 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 ...
  - if grade >= 90 then ...
  - The true branch is taken if it does not evaluate to
    - `false`
    - `nil`
  - Warning: 0 is not false!
More Control Statements

- unless grade < 90 then
  puts "You got an A" unless grade < 90
- until i >= n
  puts message
  i = i + 1
end

- Why so many control statements?
  - Is this a good idea? Tradeoffs?

Even more control statements

- for elt in [1, "math", 3] 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, .. when 2, .. end
- while i < n
  - break, next, redo

Practice Ruby Control Statements

- Ruby function to print all even numbers from 1 to some given value x

Classes and Objects

- 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

Everything is an Object

- In Ruby, everything is in fact an object
  - (-4).abs
    - integers are instances of 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 and Objects

- Objects are data
- Classes are types
- Fixnum, Float, String, etc., (including Class), are objects of type Class
Two cool things to do with classes

- Since classes are objects, you can manipulate them however you like
  - if p then x = String else x = Time end
  - y = x.new # creates a String or a Time, # depending upon p
- You can get names of all the methods of a class
  - Object.methods
  - => ["send", "name", "class_eval", "object_id", "new", "autoload?", "singleton_methods", ...]

What is a program?

- In Java, a program is...
  - A collection of class definitions
  - With a class MyClass that contains a method
    - public static void main(String[] args)
  - When you run java MyClass, the main method of class MyClass is invoked

Dynamic Typing

- Recall explicit versus implicit declarations
- We say that Ruby is dynamically typed
  - Types are determined and checked at run time
- Compare to C, which is statically typed

```
# Ruby
x = 3
x = "foo"  # gives x a # new type

// C */
int x;
x = 3;
x = "foo"; /* not allowed */
```

The nil object

- Ruby uses a special object nil
  - All uninitialized fields set to nil ($ refers to a class field)
    - 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 that don’t make sense
    - irb(main):006:0> $x = 2
    - NoMethodError: undefined method ‘*’ for nil::NilClass

A Ruby program is

- The class Object
  - When the class is loaded, any expression not in a method body is executed defines a method of Object

```
def sayHi(message, n)
  i = 0
  while i < n
    puts message
    i = i + 1
  end
  return i
end
x = sayHi("hello", 3)
puts(x)
```

Invokes self.sayHi
Invokes self.puts

Types in Java and C++

- Are types in Java and C++ dynamic or static?
  - Answer: both
  - Many things are checked statically
    - Vector v = new Vector();
      v.getHiupcode(); //compile time error
  - Other things are checked at runtime
    - Vector v = new Vector();
      Person p = (Person) v; //runtime error
Tradeoffs?

Static Typing
More work when writing code
Helps prevent some subtle errors
Fewer program types to check

Dynamic Typing
Less work when writing code
Can use objects incorrectly and not realize until exec.
More program types to check

Notes to Java programmers

- Ruby does not support method overloading
  - A typical Java class might have two or more constructors
  - Since Ruby does not support method overloading there can only be one initialize method
- Ruby does issue an exception or warning if classes defines more than one initialize method
  - But last initialize method defined is the valid one

Inheritance

- Recall that every class inherits from Object

Classes and Objects in Ruby

```
class Point
  def initialize(x, y)
    @x = x
    @y = y
  end
  def addX(x)
    @x += x
  end
  def to_s
    return "(\(x + @x.to_s + ",\" + @y.to_s + ")"
  end
end
p = Point.new(3, 4)
p.addX(4)
puts(p.to_s)
```

- Recall classes begin with an uppercase letter
- `inspect` converts any instance to a string
  ```
  irb(main):033:0> p.inspect
  => "<Point:x=54574 @x=4, @y=7>"
  ```
- Instance variables are prefixed with `@`
  - Compare to local variables with no prefix
  - Cannot be accessed outside of class
- The `to_s` method can be invoked implicitly
  - Like Java's `toString()` methods

```
super() in Ruby

- Within the body of a method
  - Call to `super()` acts just like a call to that original method
  - Except that search for method body starts in the superclass of the object that was found to contain the original method

```
Global Variables

- 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 += 1; $x += 1
    end
  Global.inc
  puts $x
  end
  def Global.get
    return $x
  end
end
```

- Define a class (`singleton`) method

---

Strings in Ruby

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

- Here-documents
  ```ruby
  s = <<END
This is a text message on multiple lines and
typing \n is annoying
END
  ```
- Print with `printf`, `sprintf`, `to_s`

---

String library in Ruby

- `s.length`
- `s1 == s2`
- `s.chomp` and `s.chomp!`
  - `!` modifies the object
  - `?` observes the object
- `s.index('st', 0), s.sub('s', 't'), s.gsub('s', 't'), s.split("t")`
- `s.each("t"){ |x| puts x}`
  - Apply code to each tab-separated substring

---

Object Copy vs. Reference Copy

- Consider the following code
  ```ruby
  x = "groundhog" ; y = x
  ```
  - Assume an object/reference model like Java or Ruby
  - Or even two pointers pointing to the same structure
  ```ruby
  x = "groundhog" ; y = x
  ```
  - Which of these occur?

[Diagram showing object copy and reference copy]

---

Object Copy vs. Reference Copy

- For `x = "groundhog" ; y = x`
  - Ruby and Java would both do a reference copy
- But for `x = String.new(x)`
  - Ruby would cause an object copy
  - Unnecessary in Java since Strings are final

---

Physical vs. Structural Equality

- Consider these cases again:
  ```ruby
  x = "groundhog" ; y = new "groundhog"
  ```
  - If we compare `x` and `y`, what is compared?
    - The references, or the contents of the objects they point to?
  - If `references` are compared (physical equality) the first would return false but the second true
  - If `objects` are compared both would return true
String Equality

- In Java, `x == y` is physical equality, always
  - Compares references, not string contents
- In Ruby, `x == y` for strings uses structural equality
  - Compares contents, not references
  - `==` is a method that can be overridden in Ruby!
  - To check physical equality, use the `equal?` method
    - Inherited from the object class
- It’s always important to know whether you’re doing a reference or object copy
  - And physical or structural comparison

Summary

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

Reminders

- Discussion section tomorrow
  - Go over Ruby examples (available on webpage)
  - Go over Project 1 notes
    - Available on website
    - Ignore due date on slides; due Wednesday June 10
  - We will talk about regular expressions in Ruby on Thursday