Ruby Notes

Set up

* Create a GitHub repository
* Clone it in cloud9 using an SSH key
* Be sure to commit changes and push at the end and share repository with class
* Show students how to use Gist

Three principles

* Everything is an object
* Every operation is a method call on some object and returns a value
* All programming is metaprogramming
  + Classes and methods can be added or changed at any time

Compared to Java

* Similarities
  + Memory is managed by garbage collector
  + Objects are strongly typed
  + Public, private and protected methods
  + Doc tools similar to javadocs (rdocs)
* Differences
  + Not compiled
  + Different GUI toolkits
  + Use end instead of curly brackets
  + All member variables are private
  + Everything is an object (almost)
  + No static type checking
  + Variable names are labels. There are no types associated with them.
  + No type declaration needed
  + No casting
  + Constructor is named initialize
  + Instantiate an object using a new method. Example: foo = Foo.new("hi")
  + Use nil instead of null
  + Equality comparisons are different
    - use == to test equality (.equals in Java)
    - use equals? to test for the same object (== in Java)

Basics

* Statement are separated with newlines
* Indentation is not significant
* All major arithmetic operators plus \*\* for exponentiation
* Basic logic operators: <, >, <=, >=, ==, !=
* Boolean values: true and false
* Logical operators: && || !
* Combined assignment operators: += -+ \*= /=
* Does not have ++ and —
* Printing
  + puts outputs string followed by a newline
  + print outputs a string without a newline
* Comments
  + Anything following a # is a comment
  + Multi-line comments start with =begin and end with =end
* Variables
  + Don’t need a type - just start using
  + convention: names should begin with a lowercase letter and words should be separated with underscores
* User input
  + gets is the method that Ruby uses to get input from a user
  + Ruby automatically adds a newline at the end of each value input
  + chomp removes that value
  + Example: variable\_name = gets.chomp
* To insert values into strings use #{stuff to stringify}
  + This is called string interpolation
  + Example: puts "Your name is #{first\_name}"
* Conversions
  + to\_s converts numbers to strings
  + to\_i converts something to an integer
  + to\_a converts something to an array
* Concatenation operator <<
  + also known as "the shovel" can add an element to the end of an array or concatenate one string to another
  + example:  
    [1, 2, 3] << 4
  + another example:  
    "Cindy" << "Howard"
* is\_a? tests for variable type
  + Example: [x.is](http://x.is)\_a? Integer

Strings

* can use single-quotes or double-quotes
* can also use %q{test} or %Q{test}
* methods
  + length
  + reverse
  + upcase and downcase
  + capitalize
  + split - takes a string and returns an array
    - provide a delimiter
    - Example: words = text.split(" ")
* ending a method with ! causes the method to be executed in-place
* include? - testing for a substring
  + methods that end with ? return a Boolean value
  + example: if user\_input.include? "s"
* gsub! - global substitution
* \* num will repeat a string *num* times
* + can be used for concatenation
* can index strings using subscripts in square brackets

Conditional Statements

* Use if-elsif-else-end
  + Don't need to indent statements (like Python) but it's considered good style
* unless tests for false cases
  + Example:  
    hungry = false  
    unless hungry  
     puts "Write code"  
    else  
     puts "Eat something"  
    end
* Single line if and unless
  + put expression first then if
  + Example:  
    puts "Fall" if true
* ternary conditional expression
  + example:   
    puts 3 < 4 ? "3 is less than 4!" : "3 is not less than 4."
* Case statements
  + use case - when
  + Example:  
    case language  
    when "JS"  
     puts "Websites!"  
    when "Python"  
     puts "Science!"  
    when "Ruby"  
     puts "Web apps!"  
    else  
     puts "I don't know!"  
    end
  + Shorter version when - then  
    case greeting  
     when "English" then puts "Hello!"  
     when "French" then puts "Bonjour!"  
     when "German" then puts "Guten Tag!"  
     when "Finish" then puts "Haloo!"  
     else puts "I don't know that language"  
    end

Loops

* while loop - start with while and end with end
* until loop
  + executes while the condition is false
  + example:  
    counter = 1  
    until counter > 10  
     puts counter  
     counter = counter + 1  
    end
* for loop
  + syntax for *var in range*
  + range is specified by using starting value, two or three dots and ending value
    - two dots .. means including last value
    - three dots … means excluding last value
  + Example:   
    for num in 1..15  
     puts num  
    end
* loop iterator
  + loop {"Hello World"} - creates an infinite loop
    - curly brackets substitute for do and end
  + use break if *condition* to exit the method
    - example:  
      i = 20  
      loop do  
       i -= 1  
       print "#{i}"  
       break if i <= 0  
      end
  + use next if to skip a loop iteration
    - example:  
      i = 20  
      loop do  
       i -= 1  
       next if i % 2 == 0  
       print "#{i}"  
       break if i <= 0  
      end
  + times iterator - like a condensed for loop
    - syntax: *num.times {#do something}*
    - example:  
      3.times {puts "Time for bed!"}
  + can also use .upto and .downto to count ranges of letters and numbers
    - Example:  
      95.upto(100) { |num| print num, " " }

Arrays

* use [] to define an array
  + example: my\_array = [1, 2, 3, 4, 5]
  + don't need the square brackets
* Elements do not need need to be the same type
* each iterator can apply an expression to each element of an object
  + syntax: *object.each { |item| #do something}*
  + You can use do and end instead of { }
  + Example:  
    my\_array.each do |x|  
     x += 10  
     print "#{x}"  
    end
  + Another example  
    languages = ["C++", "Java", "JavaScript", "Python", "Ruby"]  
    languages.each {|lang| puts lang}
* methods
  + max and min
  + sort or sort!
  + length
  + push
* Can access array elements using
  + forward subscript (positive number)
  + backward subscript (negative number)
  + range (number … number)
    - number can be forward or backward subscript
* Multidimensional arrays
  + Arrays of arrays
  + Example:  
    s = [["ham", "swiss"], ["turkey", "cheddar"], ["roast beef", "gruyere"]]  
    s.each{ |row| row.each{ |stuff| puts stuff}}
  + Access elements using two subscripts
    - Example: s[0][1]
* Sorting in reverse order
  + the combined comparison operator (<=>) aka spaceship operator compares two Ruby objects.
    - It returns 0 if the first operand (item to be compared) equals the second
    - 1 if first operand is greater than the second
    - -1 if the first operand is less than the second.
  + Example:   
    arr.sort! {|item1, item2| item2 <=> item1}

Hashes

* Like Python dictionaries
* Syntax:  
  hash = { key1 => value1,  
   key2 => value2,  
   key3 => value3  
  }
  + => is called the hash rocket
* Can create an empty Hash using Hash.new or {}
  + Example: my\_hash = Hash.new or my\_hash = {}
  + Can set a default value for new entries
    - Default value is the value returned when accessing keys that don't exist in the hash
    - If not set default is nil
    - Example: frequencies = Hash.new(0)
* Add values to Hash using bracket notation
  + Example: my\_hash['bird'] = 'Finch'
* Access the value by putting the key name in square brackets
  + Example: my\_hash['bird']
* Can use an each loop to iterate over hashes
  + Example:  
    my\_hash.each { |key, value| puts "#{key}: #{value}" }
* Can sort by key or value using hash.sort\_by{|key,value| value}
* Often use symbols as keys for hashes
  + symbol is text preceded by a colon
  + a symbol is not a string
  + only one copy can exist at a time
  + immutable
  + use as key symbol: => value
* New syntax is like this:  
  new\_hash = { one: 1,  
   two: 2,  
   three: 3  
  }
* Methods
  + The .select method takes a block consisting of a key/value pair and an expression for selecting matching keys and values.
    - Example: good\_movies = movie\_ratings.select{ |k, v| v > 3}
  + each\_key returns an array of keys
  + each\_value returns an array of values

Methods

* method is a reusable piece of code
* To define a method use def and terminate with end
  + Example:  
    def h  
     put "Hello World!"  
    end
* To call a method use its name
* Parentheses are usually optional in Ruby
  + it's a good idea to put your parameters and arguments in parentheses for readability.
  + But if there are no arguments you don’t need parentheses
* Splat arguments are arguments preceded by a \* that there is a variable number of arguments
* When calling a method, the number of arguments matters
* Can use default arguments
  + Example: def hello(name = "World")
* It's a Ruby best practice to end method names that produce Boolean values with a question mark
* implicit return statement - If a value is not explicitly returned, Ruby's methods will return the result of the last evaluated expression.
* Ruby is less concerned about what kind of thing an object is and only really cares about what method calls it responds to.
  + use respond\_to?(*symbol*) where symbol is the name of a method written as a symbol
    - Example: ar.respond\_to(:push)

Blocks, Procs and Lambdas

* A block is just a bit of code between do..end or {}.
  + It's not an object on its own
  + it can be passed to methods like .each or .select
  + Other methods that use blocks
    - times
    - collect (operates on each element of an array)
    - Example:  
      ints = odds\_n\_ends.select{|x| x.is\_a? Integer}
  + Blocks can be used to sort in reverse order
    - Example:  
      fruits = ["orange", "apple", "banana", "pear", "grapes"]  
      fruits.sort!{|fruit1, fruit2| fruit2 <=> fruit1}
  + A proc is a saved block we can use over and over.
  + A lambda is just like a proc, only it cares about the number of arguments it gets and it returns to its calling method rather than returning immediately.
    - A lambda is an object

Ruby in 20 Minutes

* Kicking off a script - use if \_\_FILE\_\_ == $0
  + \_\_FILE\_\_ is the magic variable that contains the name of the current file. $0 is the name of the file used to start the program. This check says “If this is the main file being used…” This allows a file to be used as a library, and not to execute code in that context, but if the file is being used as an executable, then execute that code.

Object-Oriented Programming

* Ruby supports reflection - the ability to ask objects about themselves
* The notation a.b means "call method b on object a
  + The variable a is the receiver
  + If it can't handle the method call, it will pass it to its superclass
* Define using class and end
* initialize is the constructor method
* instance variables are declared using @
* To instantiate an object use *ClassName*.new
* Class variables start with two @
* include a to\_s method (to string)
* Setters and getters
  + Ruby needs methods in order to access attributes
    - typically named the name of the attribute for getters and attribute= for setters
    - Can create them automatically using attr\_reader to access a variable and attr\_writer to change it.
    - Or if granting both reading and writing use attr\_accessor
  + Example:  
    class Person  
     attr\_reader :name  
     attr\_writer :name  
     def initialize(name)  
     @name = name  
     end  
    end
* Examples: pizza.rb , machine.rb and bank.rb
* Inheritance
  + class DerivedClass < BaseClass  
     # Some stuff!  
    end
  + Multiple inheritance is not allowed
* Can override methods by redefining
* Access parent methods by using keyword super
  + Used inside of a method and tells Ruby to look in the superclass of the current class and find a method with the same name.
* Methods are public by default
* This method of not caring about the actual type of a variable, just relying on what methods it supports is known as “Duck Typing”, as in “if it walks like a duck and quacks like a duck…”.
* Example: ducks.rb

Modules

* think of a module as a toolbox that contains a set methods and constants
* Like classes but modules can't create instances and can't have subclasses.
* They're just used to store things
* To make a module use the module keyword
* One of the main purposes of modules is to separate methods and constants into named spaces (namespacing)
* To include a module use require
* Use include to embed the module and don't have to qualify names
* There are modules that are part of the Ruby language
  + Math
  + Date
* Example: week.rb and decade.rb
* Mixins
  + Allow you to add functionality to a class
  + Example: mixins.rb

Book Notes

* Regular expressions – need to spend time on this?
* Poetry Mode
  + The ability to omit parentheses and curly braces when the parsing is unambiguous.
  + These are all the same:

link\_to('Edit', {:controller => 'students', :action => 'edit'})

link\_to 'Edit', :controller => 'students', :action => 'edit'

link\_to 'Edit', controller: 'students', action: 'edit'

* Ruby objects have types, but the variables that refer to them don't
* Ruby uses lexical scoping (not sure if we need this)