# CSC 443: Web Programming

**Haidar Harmanani**
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

```ruby
# Output "I love Ruby"
say = "I love Ruby"
puts say
```

```ruby
# Output "I *LOVE* RUBY"
say['love'] = "*love*"
puts say.upcase
```

```ruby
# Output "I *love* Ruby"
# five times
5.times { puts say }
```

LAU Computer Science

# The Ruby Language

- Yet another scripting language
- Quite clean syntax
- Has OO features
- Lots of useful packages for GUI building
- Supports many of the "modern" programming language ideas

# History and Motivation of Ruby

- Inspired by Perl, Smalltalk and Lisp, Yukihiro Matsumoto ("Matz") started to work on Ruby in 1993.
- It is named as a gemstone because the creator was inspired by Perl
  - In 1993, Summer: First “Hello, world!” program
  - First public release in 1995.
  - 2003, August 4: 1.8.0 is released.
  - 2007,March:1.8.6 is released.
  - 2010: 1.9.1 released
  - 2014: 2.2.0 released
  - 2016: 2.3.2 released
  - 2016: 2.4.0-preview3 released

# A Small Example (1)

```
#This is just to show you an example
class MyFirstRubyProgram
  def SayHello(name="CSC443")
    puts "Hello, #{name}!"
  end
end
```

Comment

Define a class

instance method with default parameter

```
MyFirstRubyProgram.new.SayHello("World!")
```

Instantiate the class and call a method

# A Small Example (1)

- ruby: execute from command line
  - `ruby` MyFirstRubyProgram`.rb`

# A Small Example (1)

- Using irb

Terminal — ruby — 80×24

```
maps:~ haidar$ irb
>> class MyFirstRubyProgram
>> def SayHello(name)
>> puts "Hello, #{name}!"
>> end
>> end
=> nil
>> MyFirstRubyProgram.new.SayHello("OOP 2005")
Hello, OOP 2005!
=> nil
>>
```



LAU Computer Science

# History and Motivation of Ruby

- Available on multiple platforms such as Linux, MacOS X, Windows
- According to Matz its primary application domains are Text processing, CGI programming, Network programming, GUI programming, XML programming, Prototyping, Programming education
- Ruby has adopted features from languages such as Perl, Lisp, Smalltalk
- It is very popular in Asia, especially in Japan

# Ruby in a Nutshell

- Paradigm: Pure OO language
- Simple and without surprises: Easy to learn and understand
- Potential: Powerful and expressive
- Add-on's: Rich library support
- Productive: Rapid development
- Non commercial: Open Source
- Robust: Garbage Collector on Board
- Flexible: Untyped, dynamic language
- And of course: It's cool!

# Basic syntax rules

- Comments start with a # character, go to EOL
- Each expression is delimited by ; or newlines
  - Using newlines is the Ruby way
- Local variables, method parameters, and method names should all start with a lowercase letter or _
- Global variables are prefixed with a $ sign
- Class names, module names, and constants should start with an uppercase letter
- Class instance variables begin with an @ sign

# Variable Names and Scopes

| | |
|---|---|
| `foo` | Local variable |
| `$foo` | Global variable |
| `@foo` | Instance variable in object |
| `@@foo` | Class variable |
| `MAX_USERS` | “Constant” (by convention) |

# Some Ruby Syntax

No variable declarations

Newline is statement separator

```
sum = 0
i = 1
while i <= 10 do
    sum += i*i
    i = i + 1
end
puts "Sum of squares is #{sum}\n"
```

`do ... end` instead of `{ ... }`

Optional parentheses in method invocation

Substitution in string value

# Some Ruby Syntax

- Single quotes (only `\'` and `\\`)

```
'Bill\'s "personal" book'
```

- Double quotes (many escape sequences)

```
"Found #{count} errors\nAborting job\n"
```

- `%q` (similar to single quotes)

```
%q<Nesting works: <b>Hello</b>>
```

- %Q (similar to double quotes)

```
%Q|She said "#{greeting}"\n|
```

- "Here documents"

```
<<END
First line
Second line
END
```

# Everything's an Object

"xyz"

# Everything's an Object

```
"xyz".upcase

#=> "XYZ"
```

# Everything's an Object

```
2 + 2                    #=> 4
```

Computer Science

# Everything's an Object

```
2 + 2                    #=> 4
```

Is really:

```
2.+(2)                   #=> 4
```

# Everything's an Object

```
[1, :two, "third"].last

#=> "third"
```

# Everything's an Object

```
{:a=>1, :b=>2}.keys

#=> [:a, :b]
```

Computer Science

# Simple I/O puts - put string?

```
#!/usr/bin/ruby
puts 42                      # print out an integer number
puts 1.0 * 4                 # resort to floating arithmetic
puts 11 / 2                  # integer division
puts 'Hello World!'          # there is an implied newline
puts "and from me\n\n"       # but we can add more
puts 159.to_s + ' rocks!'    # need the builtin  to_s
puts 'again and' *3          # repeats
```

% **test00.rb**

```
42
4.0
5
Hello World!
and from me

159 rocks!
again andagain andagain and
```

*We can backslash escape if we want quotes in the quotes eg* ***'you\'re in the Atrium'***

LAU Computer Science

# Assignment and Variables

```
#!/usr/bin/ruby

my_string = "hello"
another_string = "world"

puts my_string + another_string

var1 = 42
var2 = var1 * 10

my_string = 42

puts var1 + var2

#puts my_string + another_string
puts my_string.to_s + another_string
```

% **test02.rb**

helloworld

462

42world

- Variables in ruby are in fact **references**
- Ruby has **implicit typing**
- We can't concat a number and a string without explicit conversion

LAU Computer Science

# Ruby's scanf returns tuples

```
#!/usr/bin/ruby
require 'scanf'
a, b, c = scanf( "%d%f%s" )
puts 'a=' + a.to_s
puts 'b=' + b.to_s
puts 'c=' + c.to_s
some_str = "a string with 42 in it"
wrd1, wrd2, wrd3, d = some_str.scanf("%s%s%s%d")
puts 'd=' + d.to_s
printf( "a=%d, b=%f, c=%d, d=%d\n", a, b, c, d );
```

```
% test05.rb
1 2 3
a=1
b=2.0
c=3
d=42
a=1, b=2.000000, c=3, d=42
```

Computer Science

# Ruby Math Object

```
#!/usr/bin/ruby

puts rand  #  uniform deviate [0.0,1)
puts rand(4)   #  0,1,2 or 3   [0,4)

puts( Math::PI)
puts( Math::E)
puts( Math.cos( Math::PI/3) )
puts( Math.tan( Math::PI/4) )
puts( Math.log( Math::E**2) )
puts( (1 + Math.sqrt(5) ) / 2 )
```

% **test06.rb**

```
0.925032119033858
2
3.14159265358979
2.71828182845905
0.5
1.0
2.0
1.61803398874989
```

# Useful Ruby Functions

***.to_s***

.***to_i***

.***to_f***

.***ljust***(line_width)

.***center***(line_width)

.***rjust***(line_width)

.***reverse***

.***length***

.***upcase***

***.downcase***

***.swapcase***

***.capitalize***

***.strip*** *# strips whitespace*

.***rstrip***

***.lstrip***

***.gsub***( regex1, regex2 )

.***split*** *# return array of strings*

LAU Computer Science

# Control Flow

```
myvar = 2
if myvar == 1
  puts "was one"
elsif myvar == 2
  puts 'was almost one'
else
  puts 'was not one'
end
```

```
input = ''
while input != 'exit'
  puts input
  input = gets.chomp
end

input = ''
until input == 'exit' do
  puts input
  input = gets.chomp
end
```

```
loop do
  if true then
    break
  end
end

case
when input == 'exit' then
  puts 'exiting'
when input == '' then
  puts 'blank'
else
  puts 'something else'
end
```

LAU Computer Science

# Control Flow

```
if x < 10 then
    ...
elsif x < 20
    ...
else
    ...
end
```

```
while x < 10 do
    ...
end
```

```
array = [14, 22, 34, 46, 92]
for value in array do
    ...
end
```

# Ruby Methods

```
#!/usr/bin/ruby

def myfunc  myarg1, myarg2
  puts myarg1.to_s + " " + myarg2.to_s
end

def adder arg1, arg2
  sum = arg1 + arg2
  arg1 = arg1 * 2
  sum   #   this ends up being returned
end


myfunc "hollow", "world"

var1 = 1
var2 = 2
puts adder(var1, var2)
puts var1, var2
```

% **test09.rb**

hollow world

3

1

2

- Ruby methods return the last expression evaluated
- Arguments are pass-by-value
- We use tuple syntax for multiple arguments
- Local scoped variables behave as you would expect

LAU Computer Science

# Nice Touches – "?"

*By convention*, methods end in "?" for booleans:

```
a = [1, 2]

a.include?(1)     #> true
a.empty?          #> false
```

Computer Science

# Nice Touches – "!"

*By convention*, destructive methods end in "!"

```
x = "abc"

x.upcase
 # x still "abc", "ABC" returned
x.upcase!
 # x is now "ABC", return is tricky
```

Computer Science

# Arrays or Lists...?

```
names = [ 'Ron', 'Hermione', 'Harry' ]
names.push 'Draco'
puts names
puts names.to_s
puts names.join(', ')
puts names[2]

puts
unwanted = names.pop
puts unwanted
puts names.last

puts
i = 0
names.each do |n|
  puts 'name ' + i.to_s + ' ' + n
  i = i + 1
end

puts
3.times do
  puts 'and verily, verily, I say unto thee...'
end
```

```
% test10.rb
Ron
Hermione
Harry
Draco
RonHermioneHarryDraco
Ron, Hermione, Harry, Draco
Harry

Draco
Harry

name 0 Ron
name 1 Hermione
name 2 Harry

and verily, verily, I say unto thee...
and verily, verily, I say unto thee...
and verily, verily, I say unto thee…
```

- See also **.push**, **.pop**, **.last**, **.length**

# Equivalent Code

```
array = [14, 22, 34, 46, 92]
for value in array do
    print(value, "\n")
end
```

```
array = [14, 22, 34, 46, 92];
array.each do |value|
    print(value, "\n")
end
```

# Recursion

```
#!/usr/bin/ruby

def factorial number
  if number < 0
    return 'Factorial undefined for negative
   number'
  end

  if number <= 1
    1
  else
    number * factorial( number-1 )
  end
end

puts factorial 6

puts factorial( 42 )
```

```
% test11.rb
720
1405006117752879898543142606244511569936384000000000
```

- Uses a **BIGNUM** automatically

LAU Computer Science

# Recursion Factorial

```
def fac(x)
  if x <= 1 then
    return 1
  end
  return x*fac(x-1)
end
```

# File I/O

```
#!/usr/bin/ruby

filename = 'junk.txt'

mystring = <<-MYMARKER
The wretched cat could not sit on the mat
any more, as the mat had been well and truly
burned by the ungrateful students of 159-331.
MYMARKER

puts mystring

File.open filename, 'w' do |fptr|
  fptr.write mystring
End  #  closes the file for us automatically

another_string = File.read filename

puts
puts  ( mystring == another_string ) ? 'they are the same' : 'they
    differ'
```

% **test12.rb**

```
The wretched cat could not sit on the mat
any more, as the mat had been well and truly
burned by the ungrateful students of 159-331.

they are the same
```

- The << “Here” syntax is a way of embedding large amounts of text in a program
- The | | syntax is like the foreach notion, binding to a scalar - a filepointer object in this case
- Note how simple reading the entire file is

LAU Computer Science

# Arrays & Hashes

```
myarray = Array.new  # creates an empty array
myarray += [42]
myarray += [78]
puts  myarray
t1 = Time.new
t2 = t1 + 60
puts t1, t2
myarray = []
myhash = {}
myarray[0] = 0
myarray[1] = 1
myarray[2] = 2
myhash['one'] = 1
myhash['two'] = 2
myhash['three'] = 3
puts
myarray.each do |w1|
  puts w1
end
puts
myhash.each do |w2|
  puts w2
end
```

```
% test13.rb
42
78
Mon May 14 14:41:48 NZST 2007
Mon May 14 14:42:48 NZST 2007

0
1
2

three
3
two
2
one
1
```

LAU Computer Science

# Arrays and Hashes

```
x = Array.new          // or x = Array.new(10)
x << 10               // add an element to x
x[0] = 99
y = ["Alice", 23, 7.3]
x[1] = y[1] + y[-1]


person = Hash.new
person["last_name"] = "Rodriguez"
person[:first_name] = "Alice"
order = {:item => "Corn Flakes", :weight => 18}
order = {item: "Corn Flakes", weight: 18}
```

# Classes

```
class Person
  attr_reader :name
  def initialize(name)
    @name = name
  end
end

Person.new("Joe").name   #=> "Joe"
```

- Use @ to denote an instance variable
- The constructor is called *initialize*

# Ruby Classes

```
Class Person
  attr_accessor :age
end

j = Person.new()
j.age = 30
j.age                    #=> 30
```

# Ruby Classes

```
Class Person
  def height=(h)
    @height = h
  end
  def height
    @height
  end
end
```

```
j = Person.new()
j.height  #=> nil
j.height = 72
j.height  #=> 72
```

# Classes are Open

```
# String is a delivered class
class String
  def introduce
    puts "My name is #{self}"
  end
end

"George".introduce
#=> “My name is George”
```

Computer Science

# Duck Typing

```
class Duck
  def walk
    "waddle"
  end
  def talk
    "quack"
  end
end
```

LAU Computer Science

# Duck Typing

```
class Goose
  def walk
    "waddle"
  end
  def talk
    "honk"
  end
end
```

Computer Science

# Duck Typing

```
duck = Duck.new
duck.walk        #=> "waddle"
duck.talk        #=> "quack"

bird = Goose.new
bird.walk        #=> "waddle"  (Duck?)
bird.talk        #=> "honk"    (No. ☹)

Rabbit.new().walk #=> "hop"  (not a duck)
```

# More Ruby Classes

```
#!/usr/bin/ruby

class Die
  def initialize
    roll
  end

  def roll
    @current = 1 + rand(6)
  end

  def shows
   @current
  end
end

pair = [Die.new, Die.new]
pair.each do |d|
  puts d.roll
end
puts pair[0].shows
```

```
% test14.rb
3
2
3
```

LAU Computer Science

# Simple Class

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

  def x
    @x
  end

  def x=(value)
    @x = value
  end
end
```

```
p = Point.new(3,4)
puts "p.x is #{p.x}"
p.x = 44
```

# 1st class functions

```
double = Proc.new do |num|
  num * 2
end

triple = Proc.new do |num|
  num * 3
end

square = Proc.new do |num|
  num * num
end

cube = Proc.new do |num|
  num * num * num
end

myfuncs = Array.new
myfuncs[0] = double
myfuncs[1] = triple
myfuncs[2] = square
myfuncs[3] = cube
myfuncs[4] =  Proc.new do |num|   num + 42   end

puts myfuncs[ rand(5) ].call(2)
```

- Can have array of functions
- See *Proc* class for more info
- Need the *.call* handle

```
% test15.rb
4
% test15.rb
8
% test15.rb
4
% test15.rb
8
% test15.rb
6
% test15.rb
44
% test15.rb
44
% test15.rb
8
% test15.rb
44
```

LAU Computer Science

```
def compose proc1, proc2
  Proc.new do |num|
    proc2.call( proc1.call(num) )
  end
end

double_then_square =
      compose double, square

puts double_then_square.call(4)
```

- Assuming the definitions of double, square etc from previous slide:
- We can def compose
- This gives **64** as output
- Mechanism allows us to do currying…
- See also **lambda**…

LAU Computer Science

# Arguments: Defaults, Variable #

```
def inc(value, amount=1)
  value+amount
end

def max(first, *rest)
  result = first
  for x in rest do
    if (x > result) then
      result = x
    end
  end
  return result
end
```

# Blocks, Iterators, Yield

```
odd_numbers(3) do |i|
    print(i, "\n")
end
```

Block: code passed to method

```
def odd_numbers(count)
    number = 1
    while count > 0 do
        yield(number)
        number += 2
        count -= 1
    end
end
```

Iterator

Invoke method's block

# Iterators are Reusable

```
def sum_odd(count)
    sum = 0
    odd_numbers(count) do |i|
        sum += i
    end
    return sum
end

def odd_numbers(count)
    number = 1
    while count > 0 do
        yield(number)
        number += 2
        count -= 1
    end
end
```

# Module Example

```
class MyClass
  include Enumerable
  ...
  def each
    ...
  end
end
```

New methods available in MyClass:

**min**, **max**, **sort**, **map**, **select**, …

# System calls

```
#!/usr/bin/ruby

prefix = "pfx"
number = 42
ending = "txt"

filename = sprintf("%s%06d.%s", prefix, number, ending );

mycommand = "echo hello ken > " + filename

system( mycommand )
```

- **test16.rb** uses the Unix system command "**echo**" to write "*hello ken*" to a file named *pfx000042.txt* using the stdout redirect "**>**"

LAU Computer Science