CS 360 Internet Programming

Ruby

Introduction to Ruby Programming

Daniel Zappala
Computer Science
Brigham Young University

Hello World

```
1  puts "Hello, World!"
```

Hello World Method

```
1  def hello
2    puts "Hello, World!"
3  end
4
5  hello
```

Hello World Class

```
1  class Hello:
2    def hello
3      puts "Hello, World!"
4    end
5  end
6
7  h = Hello
8  h.hello()
```

Language Features

- interpreted
- object-oriented
- duck typing
  - objects are strongly and dynamically typed
  - types defined by what an object can do
  - e.g. does it support the << method?
  - see Chapter 23 of Programming Ruby
- classes, modules, exceptions, high level data types, unit testing
- easy to integrate with C, C++
- interfaces to GUIs
- packaging system (RubyGems)

Purely Object Oriented

```
1  song1 = Song.new("A Networking World")
2  song1 = Song.new("Me and My Compiler")
3  song1.play
4
5  "this is a string".length
6  n = -1942.abs
```

- Java: n = Math.abs(number);

Code Indentation

- unlike Python, code blocks are *not* determined by indentation
- instead use the *end* keyword to designate the end of a block
- this works just as well, but use two space indenting!

```
1  def hello
2  puts "Hello, World!"
3  end
4
5  hello
```

Methods

```
1  def say_goodnight(name)
2    result = "Good night, " + name + "."
3    return result
4  end
5
6  # time for bed
7  puts say_goodnight("room")
8  puts(say_goodnight("moon"))
9  puts say_goodnight "cow jumping over the moon"
```

- parenthases are optional in method calls
- precedence rules are difficult to remember, so use them in all but the most obvious cases

Strings

- double-quoted or single-quoted
  - single-quoted are mostly unchanged by Ruby
  - double-quoted allow substitution (\n) and expression interpolation

```
1  def say_goodnight(name)
2    result = "Good night, #{name}."
3    return result
4  end
```

Return Values

- Ruby returns the value of the last expression evaluated

```
1  def say_goodnight(name)
2    "Good night, #{name}."
3  end
```

Naming Conventions

- local variables, method parameters, method names start with a lowercase letter or an underscore
- class names, module names, and constants start with an uppercase letter
- global variables begin with $
- instance variables begin with @
- class variables begin with @@
- multiword variables are written with underscores
- multiword class names are written in mixed case

Arrays and Hashes

- indexed collections of objects
  - array: key is an integer
  - hash: key is any object
- arrays are more efficient, hashes are more flexible

Arrays

- may contain any type of objects
- simple initialization and direct access
- can initialize string arrays with a shortcut

```
1  a = [ 1, 'cat', 3.14]
2  a[0]
3  a[2] = 3.14159
4
5  a = [ 'ant', 'bee', 'cat', 'dog', 'elk' ]
6  a = %w{ ant bee cat dog elk }
```

Hashes

- keys must be unique
- keys and values may be arbitrary objects
- returns nil object if no corresponding entry

```
1  inst_section = {
2    'cello'     => 'string',
3    'clarinet'  => 'woodwind',
4    'drum'      => 'percussion',
5    'oboe'      => 'woodwind',
6    'trumpet'   => 'brass',
7    'violin'    => 'string'
8  }
9
10 inst_section['oboe']
11 inst_section['bassoon']
```

If and While Statements

```
1  if count > 10
2    puts "Try again"
3  elsif tries == 3
4    puts "You lose"
5  else
6    puts "Enter a number"
7  end
8
9  while weight < 100 and num_pallets <= 30
10   pallet = next_pallet()
11   weight += pallet.weight
12   num_pallets += 1
13 end
```

Statement Modifiers

- shortcut for one-line if and while bodies

```
1  puts "Danger, Will Robinson" if radiation > 3000
2
3  square = 2
4  square = square*square while square < 1000
```

Blocks

- a chunk is any code between braces or a do-end block
  - use braces for a single-line block
  - use do-end for a multi-line block
- can associate a block with a method invocation, almost as if it was a parameter to the method
- can use a block to
  - implement callbacks
  - pass around code (like a function pointer)
  - implement iterators

```
1 { puts "Hello" }
2 do
3   club.enroll(person)
4   person.socialize
5 end
```

Blocks as Method Parameters

- add the block after the method call
- call the block inside the method using yield

```
1  def call_block
2    puts "Start of method"
3    yield
4    yield
5    puts "End of method"
6  end
7
8  call_block { puts "In the block" }
```

Block Arguments and Iterators

- can pass arguments to a block
- may objects have builtin iterators

```
1  animals = %w{ ant bee cat dog elk }
2  animals.each {|animal| puts animal}
3
4  5.times { print "*" }
5  3.upto(6) {|i| print i }
6  ('a'..'e').each {|char| print char }
```

- output
  - puts: writes arguments, with newline after each one
  - print: writes arguments, with no newline
  - printf: uses C format strings
- input
  - gets: gets a line from standard input