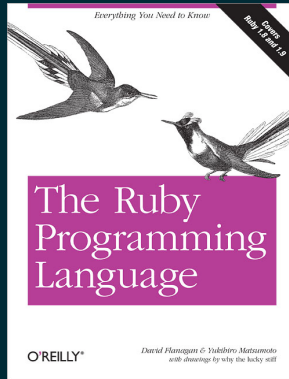


Ruby Intro

CSCI400

Valuable Reference

The Ruby Programming Language, Flanagan and Matsumoto
(creator of Ruby)



Getting Started

Option 1: REPL

- 1 Open irb (interactive ruby)
- 2 Try out a ruby command, e.g.
 - `puts "Who you callin' pinhead?"`
Who you callin' pinhead?
=> nil
 - `puts` returns nil

Option 2: Scripting

- ❶ Open a text editor
- ❷ Write ruby commands, e.g.
 - `puts "My god...it's full of stars!"`
- ❸ Save file as `demo1.rb`
- ❹ Open git bash: `ruby demo1.rb`
 - Must be in same directory as `demo1.rb`

Ruby: General Structure

Aside: Expressions vs. Statements

- *Expressions produce a value*
- *Statements generally do something*
 - May contain expressions (distinction sometimes blurry)
 - **Expression vs. statements in Python**

Ruby Program Structure

- Basic unit is *expression*
- Primary expressions
 - `true`, `false`, `nil`, `self`, number/string literals, variable references
- Expression types: arithmetic, boolean

Ruby Program Structure

Code can be organized using:

- Blocks
- Methods
- Classes
- Modules

Program Execution

- Ruby is a *scripting* language
 - No special `main` method
 - Script start at a line 1, execute all lines in order
- A method/class is defined when the definition is read/executed
 - Method calls must come *after* the method's definition

Ruby Expression Structure

- Whitespace: mostly ignored
- Expression separators: newline
- If statement doesn't fit on one line...
 - 1 Insert newline after operator, period, or comma
 - 2 or escape the newline *

*Think: How does interpreter recognize tokens/statements?

Ruby: Basic Constructs

Block Structure

```
# Block surround by {}  
10.times { puts "hello" }
```

```
x = 5  
unless x == 10  
  # this is the start of an `unless` block  
  print x # 'body' of the block  
end # this ends an `unless` block
```

Blocks can be nested. Indent for clarity.

Ruby Comments

```
# This is a comment
```

or

```
=begin  
This is a longer comment. =begin/=end must be  
at the start of a line.  
=end
```

Variables/Methods

- Valid variable name identifiers
 - letters, numbers, _
 - Cannot start with number
 - *Global var*: start with \$
 - *Instance, class vars*: start with @, @@ resp.)
- Conventions
 - ?: End method name with ? if returns boolean
 - !: End method name with ! if dangerous
- Constants, classes, modules: begin with A-Z

Ruby Data Types: Numbers

Numeric

- Integer – allows base 8, 16, 2
 - Fixnum: Fit in 31 bits
 - Bignum: Arbitrary size
- Float
 - Includes scientific notation
- Complex
- BigDecimal
 - Uses decimal rather than binary representation
- Rational

A few number-related details

- `div`: integer division
 - `7.div 3`
- `fdiv`: floating point division
 - `7.fiv 3`
- `quo`: rational division
 - In `irb`, try: `(1.quo 3).class`
- $-7/3 = -3$ in Ruby, -2 in Java/C++ ([explanation](#))
- Float limits: See `INFINITY`, `MAX` in [this link](#)
- Numbers are immutable (as you'd expect)

Ruby Data Types: Strings

- String literals: *single quote*
 - `'A ruby string'`
 - `'Don\'t call me Shirley.'`
 - Only escape `'` or `\`
 - Newlines are embedded if multi-line
- String literals: *double quote*
 - Normal escape sequences (`\t`, `\n`, etc.)
 - String interpolation

```
w = 5
h = 4
puts "The area is #{w*h}"
```

More on strings (1)

- In Ruby, strings are *mutable*
- `+`: concatenation (interpolation often preferred)

```
age = 32  
puts "I am " + age.to_s
```

- `<<`: append

```
s = "Hello"  
s << " World"  
puts s
```

More on strings (2)

- Substring
 - `puts s[0, 5]`
- `*`: repeats string
 - `puts "Alright" * 14`
- `length`: returns length of `s`
 - `puts s.length`

Characters

- Strings of length 1
 - Changed from Ruby 1.8 \rightarrow 1.9

Methods

- No () needed for function calls. Try:
 - `"hello".center 20`
 - `"hello".delete "lo"`
- Note: if using (), *don't put space after function name*
 - `f (3+2)+1 != f (3+2)+1`
- Best practice?
 - Some thoughts

String access

- Cases with examples
 - `[i]` # `puts s[0]`, `puts s[-1]`
 - `[i, len]` # `puts s[0, 4]`
 - `[i..j]` # `puts s[0..3]`

String Access: Quick Exercise

- Try:
 - `s = "Sunday, Monday, Tuesday, Wednesday, Thursday, Friday"`
 - Extract "Sunday", "Monday", and "Friday"
 - Figure out how to turn `s` into `["Sunday", "Monday", ... "Friday"]` (Hint)
- Play with [the Ruby basics file](#)
- Nothing to submit, no right answers – just play!

Get Started

Do Ruby intro homework