Introduction and best practice

Luigi De Russis

Gems and conventions

2

RubyGems

What is a gem?

- A RubyGem is a software package
  - commonly called a “gem”
- Gems contain a packaged Ruby application or library
- The RubyGems software itself allows you to easily download, install and manipulate gems on your system
- Each gem has a name, version, and platform

What is a gem?

- Gems can be used to extend or modify functionality within a Ruby application
  - they are used to split out reusable functionality that other can use in their applications as well
  - some gem also provide command line utilities to help automate tasks and speed up your work
- Gems can include C extension
- Since Ruby 1.9.2, RubyGems is included when you install the programming language

Some useful commands…

- gem update
  - update installed gems to the latest version
- gem update --system
  - update the RubyGems system software
- gem install/uninstall GEMNAME
  - install/uninstall one or more gems from the local repository
- gem check [GEMNAME…]
  - check a gem repository for added or missing files
- gem cleanup [GEMNAME…]
  - clean up old versions of installed gems in the local repository
- gem environment
  - display information about the RubyGems environment
- gem list
  - display installed gems

6

Ruby code conventions

Main goal

Code readability

Conventions from the field…

- Two spaces per indent
- One statement per line
  - keep lines fewer than 80 characters
- Use spaces around operators, after commas, colons and semicolons, around `{` and before `}`
- No spaces
  - after `(`, `[` or before `)`, `]`
  - after `!`
- Use empty lines between `def`s and to break up a method into logical paragraphs

Conventions from the field…

- Typically single line comments are used
  - everything that follows #
- Use parentheses in method when there are arguments
  - omit the parentheses when the method does not accept any arguments
  - exceptions: `puts`, and control statements
- `each` is preferred over `for`
- Never use `then` for multi-line `if/unless`
- Never use `unless` with `else`
- Prefer `{…}` over `do…end` for single-line blocks
- Avoid using `{…}` for multi-line blocks
- Always use `do…end` for control flow and method definition

Conventions from the field…

- Avoid `return` when not required
- Never put a space between a method name and the opening parenthesis
- Use `snake_case` for methods and variables
- Use `CamelCase` for classes and modules
  - keep acronyms (like HTTP) in the original case
- Use `SCREAMING_SNAKE_CASE` for other constants
- Avoid the usage of class variables (`@@`)
- Prefer string interpolation instead of string concatenation
- Prefer double-quoted strings

License