RubyStyleGuide (Ruby)

- Please add yourself as a contributor if you create a page or make significant changes to a page.
- Please create new pages using a style similar to existing ones.

Naming Conventions

- **CONSTANTS_USE_UPPER_CASE** -- for example: `Math::PI`, `Curses::KEY_DOWN`, `OptionParser::REQUIRED_ARGUMENT`
- **ClassesUsePascalCase** -- join words of class names by capitalizing the first letter of each word. The same goes for ModuleNames as well. (See also CamelCase.)
- **methods use underscores, as do local_variables, @instance_variables, and @@class_variables** -- join words with underscores (aka "snake_case")
- Keep acronyms in class names capitalized. **MyXMLClass?**, not MyXmlClass?. Variables should use all lower-case.

There is no place in Ruby where `camelCase` is ever used.

Formatting Conventions

Method Parameters

- ParenParams -- put parentheses around non-trivial parameter lists
- ButtParams -- don't put spaces between a method name and its parameter list

Whitespace Conventions

- Use two spaces for indentation
  - *Tabs versus spaces is a matter of personal preference when editing, but the community generally agrees that two-spaces is how you should release all your code. Space indentation outnumbers tab indentation more than 10:1 in the source code.*

`{ ... }` blocks versus `do ... end` blocks

There are two prevailing conventions on when to use curly braces for blocks versus the `do ... end` keywords:

1. Use curly braces for one-line blocks, use `do...end` for multi-line blocks
2. Use curly braces when the return value is desired (or chained), use `do...end` when the side-effect is desired

The latter convention seems to be gaining in popularity, but there does not seem to be consensus on this issue at the time of this writing. (Additionally, some people always use `{ ... }`, while others always

use `do ... end`.)

Note that the two block notations have different precedence, so they are not always interchangeable unless parentheses are used.

Programming Conventions

Need categorizing

Jim Weirich captured these patterns from responses to a post about some Ruby code samples in an article being written for IBM's developerworks site.

- /RefactorIntoMethods -- Factor redundant code into a method
- /CatchAndThrowVersusExceptions - Use catch and throw for control, rather than exceptions
- /UseIterators -- Take advantage of iterators, rather than using explicit loops
- /UseBlocksForResourceManagement
- /AvoidGlobals
- /BooleansAreTrueAndFalse
- /UseLoop - to code infinite loops
- /UseMostMeaningfulClass
- /MakeObjectsDoTheWork - push operations down into the objects, rather than manipulating them globally
- /UseEnumerable -- Enumerable has a number of useful methods that work on most collections (including files)

Some things that I (JohnCarter) think are particularly rubyish....

- /ClassesAreCheap -- Creating another class in ruby is very cheap and easy. Make many small classes rather than a few large classes.
- /MixinsAddFlavour? -- Having made a data structure, pause for a moment and think what can you mix in to make it a lot more powerful.
- /ModulesAreEasy? -- Having made a few classes, it is easy to package'em up into modules.
- /ExtendRatherThanWrap -- Extend existing classes rather than create new ones.
- /InjectBlocks -- Unleash your clients, let them inject blocks into your works.

(OK, So I need to hunt up some good examples as I go along, I won't complain if someone finds them for me...)

NatPryce suggested

- /DomainSpecificIterators -- refactor code that uses generic loops and end-of-loop logic into domain specific iterators.

StephenWhite suggests:

- /UseThrowCatchWithinMethodsOnly

There's also:

- /ChopInputAsYouRead
- /InjectComplexTestsIntoObjects

LyleJohnson suggests:

- /RespondToGoesWithMethodMissing
- /DontPrefix

Miscellaneous comments

GavinKistner massively reorganized this page around revision 55 - blame him if stuff is confusing.

A copy of the original code needs to be on the front page somewhere so people will understand where the example re-writes are coming from. *(Does anyone know what this comment means? Can we lose it? --GavinKistner)*

I think, that the consensus should be given by matz implementation of the xemacs ruby.el ruby-mode. It has very good indentation and xemacs is AllYouNeed? for development. -- Brian

```
This page has a mix of advice for source code formatting (e.g., "use
CamelCase") and development techniques (e.g., "Chop input.").  Is there
a better way to partition this information to prevent this mix?  For
example, should "Chop input" be removed from this page and put on the
RubyIdioms page?  Are there better page titles for the these different
topics?  For me, 'Style Guide' means stuff like when to use CamelCase
and the like, while suggestions such as using throw-catch only in methods
sound like progamming techniques.  The two are orthogonal.
  -- JamesBritt
```

Related Pages

- RubyIdioms
