RubyMine Inspections

Syntax Inspections

- ✓ Parentheses around arguments in def
  Inspection checks using def with parentheses when there are arguments.
- ✓ For loop check
  For loop is better to be replaced with foreach.
- ✓ Then in multi-line if/unless
  This inspection warns you about unnecessary then identifier in multiline if/unless block
- ✓ Ternary operator instead of if/then/else/end constructs
  Ternaries are more common and obviously more concise.
- ✓ Ternary operator inspection
  Checks for one expression per branch in a ternary operator. Ternary operators must not be nested. Checks for using ternary operator except if/then/else/end constructs.
- ✓ Deprecated syntax
  - if x: ...
  - when x: ...
- ✓ Prefer if/unless in case of single-line body. Should be 'do_someting if some_condition'
- ✓ Unless instead of if for negative conditions (or control flow or)
- ✓ Usage unless with else
- ✓ +do block instead of {...} +
  Prefer {...} over do...end for single-line blocks. Avoid using {...} for multi-line blocks (multiline chaining is always ugly). Always use do...end for "control flow" and "method definitions" (e.g. in Rakefiles and certain DSLs). Avoid do...end when chaining.
- ✓ Return where not required
- Spaces around the = operator when assigning default values to method parameters
- Line continuation () where not required
  In practice, avoid using line continuations at all.
- ✓ Space before arguments parentheses
  Highlights method calls with space before arguments parentheses.
- Comment inspection
  If multiple lines are required to describe the problem, subsequent lines should be indented two spaces after the #.
- ✓ Case block without else
  This inspection highlights case blocks without else statement.
  The else statement should specify the default result to be returned if no match is found.
- ✓ Simplify boolean expression
  This inspection warns about redundant parts inside boolean function.
- ✓ super() call inspection
  This inspection warns about super() call with no superclasses actually defined.
- ✓ Class variable usage
  Check if class variable is defined and warns about possible unpredictable behavior
- ✓ Convert control flow
  This inspection warns that block with positive condition is preferable to block with negative condition
- Cyclomatic complexity block
  Check that the cyclomatic complexity of all methods/blocks is below the threshold.
- ✓ Empty rescue block
  This inspection reports empty rescue blocks. While occasionally intended, such
  empty rescue blocks can make debugging difficult.
- ✓ Assignment expression in conditional
  This inspection warns about using '=' instead of '==' in conditionals.
- ✓ Large Class
  A class or module that has a large number of instance variables, methods or lines of code
- Method/module line count check
  Check that the number of lines in a method/module is below the threshold.
- Parameters number check
  Check that the number of parameters on a method is below the threshold.
- ✓ Nested ternary operators
  This inspection highlights nested ternary operators.
- ✓ Parentheses around condition in conditionals
  This inspection warns about parentheses around the condition of an if/unless/while.
- Nested iterators
  This inspection highlights nested iterators.
- Duplication inspection
  Warns about two fragments of code look nearly identical, or two fragments of code have nearly identical effects at some

conceptual level.

- ? Simulated Polymorphism
- ✓ Assignment in conditional
  This inspection warns about using '=' instead of '==' in conditionals.
- ✓ Incorrect call argument count
  Highlights method calls where the number of arguments passed to the method does not match the number of method parameters.
- ✓ Jump error
  Highlights wrong and restricted usages of return, break, continue and other jump calls.
- ✓ Scope inspection
  Reports about dangerous usages of local variables or parameters.
- ✓ Unnecessary return statement
- ✓ Unnecessary retutn value
- ✓ Unnecessary semicolon
- ✓ Unreacheable code
  Highlights code statements which will never be executed in any control flow.
- ✓ Unresolved Ruby reference
  Warns about the references in Ruby code which can't be resolved to any valid target.
- ✓ Unused local variable inspection
- ✓ Wrong hash inspection
  Checks the hashes and highlights the ones with missing key or value parts.
- ✓ Yard tags insection
  Highlights any wrong usages of YARD tags.
- ✓ Inspection converts a relative path in a 'require' statement into an absolute one

✓ Naming convensions
These inspections report any elements whose names are either too short, too long, or do not follow the specified regular expression pattern.

- Parameter naming convention
- Local variable naming convention
- Instance variable namingconvention
- Global variable naming convention
- Class method naming convention
- Class module naming convention
- Class variable naming convention

Collections inspections

- ✓ Literal array syntax
  Prefer %w to the literal array syntax when you need an array of strings.
- ✓ Hash syntax inspections
  - Use symbols instead of strings as hash keys
  - Check 'Hash[]' for bugsafety syntax
  - Duplicated keys in hash
    By default all duplicated keys except last one are silently ignored and it is a hard bug to find.
    This inspection warns about duplicated keys in hash.
- Avoid the use of mutable object as hash keys
- Never modify a collection while traversing it

Strings inspections

- ✓ Inspection converts concatenations of strings to a single one with substitutions #{}
- ✓ Single-quoted strings instead of double-quoted
  Prefer single-quoted strings when you don't need string interpolation or special symbols such as \t, \n, ', etc.
- ✓ {} around instance variable string
  Don't use {} around instance variables being interpolated into a string.

Percent literals

- Regular expressions
  Use %r only for regular expressions matching more than one '/' character
- Avoid %q, %Q, %x, %s, and %W
- Prefer () as delimiters for all % literals

Source reference:

- https://github.com/bbatsov/ruby-style-guide
- https://github.com/martinjandrews/roodi#readme

- https://github.com/troessner/reek/wiki/code-smells