Ruby style guide

Classes & Modules

Use a consistent structure in your class definitions.

```ruby
class Person
  # extend and include go first
  extend SomeModule
  include AnotherModule
  # inner classes
  CustomErrorKlass = Class.new(StandardError)

  # constants are next
  SOME_CONSTANT = 20
```

```ruby
# afterwards we have attribute macros
attr_reader :name

# followed by other macros (if any)
validates :name

# public class methods are next in line
def self.some_method
end
```

```ruby
# initialization goes between class methods
# and other instance methods
def initialize
end

# followed by other public instance methods
def some_method
end
```

```
  # protected and private methods are grouped
  # near the end
  protected

  def some_protected_method
  end

  private

  def some_private_method
  end
end
```

Always supply a proper `to_s` method!

```ruby
class Person
  attr_reader :first_name, :last_name

  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end

  def to_s
    "#{@first_name} #{@last_name}"
  end
end
```

Use the `attr` family of functions to define trivial accessors or mutators.

```ruby
# bad
class Person
  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end

  def first_name
    @first_name
  end

  def last_name
    @last_name
  end
end
```

```ruby
# good
class Person
  attr_reader :first_name, :last_name

  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end
end
```

Use attr_reader and attr_accessor.

```
# good
attr_accessor :something
attr_reader :one, :two, :three
```

Consider using `Struct.new`, which defines the trivial accessors, constructor and comparison operators for you.

```ruby
# good
class Person
  attr_accessor :first_name, :last_name

  def initialize(first_name, last_name)
    @first_name = first_name
    @last_name = last_name
  end
end
```

```ruby
# better
Person = Struct.new(:first_name, :last_name)
```

Prefer duck-typing over inheritance.

```ruby
# good
class Duck
  def speak
    puts 'Quack! Quack'
  end
end

class Dog
  def speak
    puts 'Bau! Bau!'
  end
end
```

Avoid the usage of class (`@@`) variables due to their “nasty” behavior in inheritance.

```ruby
class Parent
  @@class_var = 'parent'

  def self.print_class_var
    puts @@class_var
  end
end

class Child < Parent
  @@class_var = 'child'
end

Parent.print_class_var # => will print 'child'
```

As you can see all the classes in a class hierarchy actually share one class variable. Class instance variables should usually be preferred over class variables.

Assign proper visibility levels to methods (`private`, `protected`) in accordance with their intended usage. Don't go off leaving everything `public` (which is the default). After all we're coding in *Ruby* now, not in *Python*.

Indent the `public`, `protected`, and `private` methods as much as the method definitions they apply to. Leave one blank line above the visibility modifier and one blank line below in order to emphasize that it applies to all methods below it.

```ruby
class SomeClass
  def public_method
    # ...
  end

  private

  def private_method
    # ...
  end

  def another_private_method
    # ...
  end
end
```

Use `def self.method` to define class methods. This makes the code easier to refactor since the class name is not repeated.

```ruby
class TestClass
  # bad
  def TestClass.some_method
    # body omitted
  end

  # good
  def self.some_other_method
    # body omitted
  end
```

License