Jogo Rubynho

Estándar de Implementación

Versión 2.0

Historia de revisiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 06/09/2014 | 1.0 | Versión inicial | Juan Rahi |
| 11/09/2014 | 1.1 | Ajustes | Fernando Banchero |
| 22/10/2014 | 2.0 | Nuevo estándar | Fernando Banchero |
| 26/10/2014 | 2.0 | Revisión | Jorge Artave |

Contenido

[1. Convenciones Generales 3](#_Toc401862524)

[2. Estructura del código fuente 3](#_Toc401862525)

[3. Sintaxis 5](#_Toc401862526)

[4. Nombres 8](#_Toc401862527)

[5. Comentarios 9](#_Toc401862528)

[6. Clases y Módulos 10](#_Toc401862529)

[7. Colecciones 11](#_Toc401862530)

[8. Strings 13](#_Toc401862531)

[9. Otras consideraciones 14](#_Toc401862532)

[10. HTML 14](#_Toc401862533)

[11. CSS 14](#_Toc401862534)

[12. JAVASCRIPT 14](#_Toc401862535)

# Convenciones Generales

* **DRY (*Dont repeat yourself*):** Como dice la sigla, implementar el mínimo de redundancia posible de código. Cada porción de conocimiento debe tener un solo sentido, ser no ambiguo y tener una representación autoritativa en el sistema.
* **Fat Model Skinny Controller** (Modelo gordo, controlador flaco): Haciendo referencia a MVC, mantener la mayoría de la lógica dentro del modelo y dejar lo mínimo posible en los controladores.

# Estructura del código fuente

* Se utilizarán 2 espacios por cada nivel de indentación. (Se puede configurar el ambiente de desarrollo para que cada tabulador sea igual a 2 espacios).
* No uses ; para separar declaraciones y expresiones. Por convención usá una expresión por línea.
* Preferí un formato de única línea por cada definición de clase sin cuerpo.

FooError **=** Class**.**new(StandardError)

* Usá espacios entre operadores, después de las comas, dos puntos y puntos y comas, luego de { y antes de }.  La única excepción, con respecto a los operadores, es el operador exponente.

sum = 1 + 2

a, b = 1, 2

1 > 2 ? true : false; puts 'Hi'

[1, 2, 3].each { |e| puts e }

e **=** M **\*** c**\*\***2

* Sin espacios luego de (, [ o antes de ], ).
* Sin espacios luego de !.
* Usá espacios alrededor del operador = cuando asignes valores predeterminados a los parámetros del método:

# mal

def some\_method(arg1=:default, arg2=nil, arg3=[])

# do something...

end

# bien

def some\_method(arg1 = :default, arg2 = nil, arg3 = [])

# do something...

end

* Al continuar una invocación de método encadenado en otra línea mantener el . en la segunda línea.

*# mal - es necesario leer la primer línea para entender la segunda línea*

one**.**two**.**three**.**

four

*# bien - inmediatamente se ve qué está pasando en la segunda línea*

one**.**two**.**three

**.**four

* Alineá los parámetros de una llamada a un método si ocupan más de una línea.

# bien (indentado normal)

def send\_mail(source)

Mailer.deliver(

to: 'bob@example.com',

from: 'us@example.com',

subject: 'Important message',

body: source.text

)

end

* Alineá los elementos de arrays literales que ocupen múltiples líneas.

# bien

menu\_item = [

"Spam", "Spam", "Spam", "Spam", "Spam", "Spam", "Spam", "Spam",

"Baked beans", "Spam", "Spam", "Spam", "Spam", "Spam"

]

* Agregá guiones bajos para números literales grandes para mejorar su lectura.

num **=** 1\_000\_000

* No dejes una línea en blanco entre el bloque de comentario y el def.
* Limitá las líneas a 100 caracteres.
* Evitá los espacios en blanco.
* No uses los comentarios de bloque.

*# mal*

**==** **begin**

comment line

another comment line

**==** **end**

*# bien*

*# comment line*

*# another comment line*

# Sintaxis

* Usá def con paréntesis cuando tengas argumentos. Omití los paréntesis cuando el método no acepta ningún argumento.
* Nunca uses for, a menos que sepas exactamente para qué lo usás. for se debe implementar en forma de each. for no introduce un nuevo scope (a diferencia de each) y las variables que se definen dentro de su bloque son visibles fuera del mismo.
* Nunca uses then para if/unless con multilíneas.

*# mal*

*if some\_condition then*

*# body omitted*

*end*

*# bien*

*if some\_condition*

*# body omitted*

*end*

* Preferí el operador ternario (?:) en lugar de las construcciones if/then/else/end. Es más común y obviamente más conciso.

*# mal*

result **=** **if** some\_condition **then** something **else** something\_else **end**

*# bien*

result **=** some\_condition ? something : something\_else

* Usá una expresión por fila por operador ternario. Esto también significa que los operadores ternarios no deben anidarse. Es preferible utilizar construcciones if/else en estos casos. Ademas evitá usar ?: (operador ternario) en multilíneas; en su lugar usá if/unless.
* Favorecé unless por sobre if para condiciones negativas (o control de flujo con ||).
* Nunca uses unless con else. Reescribí para que el caso positivo esté primero.
* No uses paréntesis alrededor de la condición de if/unless/while/until.

*# mal*

some\_condition ? (nested\_condition ? nested\_something : nested\_something\_else) : something\_else

*# bien*

**if** some\_condition

nested\_condition ? nested\_something : nested\_something\_else

**else**

something\_else

**end**

* Usá ! en lugar de not.
* No uses las palabras and y or, en su lugar, siempre usá && y ||.
* Nunca uses while/until condition do para un while/until multilínea.

*# mal*

**while** x **>** 5 **do**

*# body omitted*

**end**

**until** x **>** 5 **do**

*# body omitted*

**end**

*# bien*

**while** x **>** 5

*# body omitted*

**end**

**until** x **>** 5

*# body omitted*

**end**

* Favorecé el uso del modificador while/until cuando puedas escribir la comparación en una línea.

*# mal*

**while** some\_condition

do\_something

**end**

*# bien*

do\_something **while** some\_condition

* Favorecé until por sobre while para condiciones negativas.

*# mal*

do\_something **while** **!**some\_condition

*# bien*

do\_something **until** some\_condition

* Usá Kernel#loop con break en lugar de begin/end/until o begin/end/while para validar al final de cada loop.

*# mal*

**begin**

puts val

val **+=** 1

**end** **while** val **<** 0

*# bien*

**loop** **do**

puts val

val **+=** 1

**break** **unless** val **<** 0

**end**

* Elegí {...} por sobre do...end para bloques de una línea. Evitá el uso de {...} para bloques multilíneas (encadenamiento de multilínea siempre es horrible). Siempre usá do...end para "control de flujo" y "definiciones de métodos". Evitá usar do...end cuando estés encadenando métodos.

names **=** **[**'Bozhidar', 'Steve', 'Sarah'**]**

*# mal*

names**.**each **do** **|**name**|**

puts name

**end**

*# bien*

names**.**each { **|**name**|** puts name }

*# mal*

names**.**select **do** **|**name**|**

name**.**start\_with?('S')

**end.**map { **|**name**|** name**.**upcase }

*# bien*

names**.**select { **|**name**|** name**.**start\_with?('S') }**.**map { **|**name**|** name**.**upcase }

* Nunca uses un espacio entre el nombre de un método y la apertura de un paréntesis.

# Nombres

* Nombres de referencia en Inglés.
* Usá snake\_case para los símbolos, métodos y variables.
* Usá CamelCase para clases y módulos. (Mantené en mayúsculas los acrónimos como HTTP, RFC, XML.)
* Usá SCREAMING\_SNAKE\_CASE para las constantes.
* Los nombres de los métodos que afirman (métodos que devuelven un valor booleano) deben terminar con un signo de pregunta y tener como prfefijo “is”. Por ejemplo: isReal?
* Los nombres de métodos potencialmente peligrosos (ejemplo: métodos que modifican self o los argumentos, exit! no ejecuta finalizadores como lo hace exit, etc.) deben terminar con un signo de exclamación solo si existe una versión segura de ese método peligroso.
* Elegí map por sobre collect, find por sobre detecet, select por sobre find\_all, reduce por sobre inject y size por sobre length. No es un requerimiento difícil; si el uso de alias realza la legibilidad, está bien usarlos.

# Comentarios

* Usá un espacio entre el primer caracter # del comentario y el texto propio del comentario.
* Los comentarios que son más largos que una palabra son capitalizados y usan puntuación. Usá [un espacio](http://es.wikipedia.org/wiki/Espacio_entre_oraciones) luego de los puntos.
* Evitá comentarios supérfluos.
* Mantené los comentarios existentes actualizados. Un comentario viejo es peor que no utilizar comentarios.
* Evitá escribir comentarios para explicar código malo. Refactorizá el código para hacerlo más auto-explicativo.

# Clases y Módulos

* Siempre proporcioná un método to\_s para clases que representen objetos de dominio.
* Asigná niveles de visibilidad adecuados para los métodos (private, protected) de acuerdo con su correcto uso. No vayas por ahi dejando todo public (que es el estado predeterminado).
* Indentá las palabas public, protected, y private tanto como los métodos a los que se aplican. Dejá una línea en blanco antes y después del modificador de visibilidad, en orden de enfatizar que eso aplica a todos los métodos que se encuentran debajo.

**class** **SomeClass**

**def** **public\_method**

*# ...*

**end**

**private**

**def** **private\_method**

*# ...*

**end**

**def** **another\_private\_method**

*# ...*

**end**

**end**

# Colecciones

* Preferí el uso de la notación para arrays literales y creación de hashes (excepto que necesites pasar parámetros a sus constructores).

*# mal*

arr **=** Array**.**new

hash **=** Hash**.**new

*# bien*

arr **=** **[]**

hash **=** {}

* Evitá la creación de grandes espacios en arrays.

arr **=** **[]**

arr**[**100**]** **=** 1 *# y así tenés un array con un montón de nils*

* Cuando estés accediendo al primer o último elmento de un array, preferí usar first o last en lugar de [0] o [-1].
* Usá Set en lugar de Array cuando estés tratando con elementos únicos. Set implementa una colección de valores desordenados sin duplicados.
* Preferí símbolos en lugar de strings y hash keys.

*# mal*

hash **=** { 'one' **=>** 1, 'two' **=>** 2, 'three' **=>** 3 }

*# bien*

hash **=** { one: 1, two: 2, three: 3 }

* Evitá el uso de objetos mutables como hash keys.
* Usá la sintaxis de hash literal cuando tus hash keys sean símbolos.

*# mal*

hash **=** { :one **=>** 1, :two **=>** 2, :three **=>** 3 }

*# bien*

hash **=** { one: 1, two: 2, three: 3 }

* Usá Hash#fetch cuando estés tratando con hash keys que deben estar presentes.

heroes **=** { batman: 'Bruce Wayne', superman: 'Clark Kent' }

*# mal - si generamos un error, puede que no lo veamos rápido*

heroes**[**:batman**]** *# => "Bruce Wayne"*

heroes**[**:supermann**]** *# => nil*

*# bien - fetch genera un KeyError, haciendo obvios los problemas*

heroes**.**fetch(:supermann)

* Confiá en el hecho de que desde Ruby 1.9 los hashes están ordenados.
* Nunca modifiques una colección mientras la estés recorriendo.

# Strings

* Preferí interpolación de strings en lugar de concatenación de strings:

*# mal*

email\_with\_name **=** user**.**name **+** ' <' **+** user**.**email **+** '>'

*# bien*

email\_with\_name **=** "#{user**.**name} <#{user**.**email}>"

* Considerá el uso de interpolación de string con espacio. Hace que sea más claro para separar el código del string.

"#{ user**.**last\_name }, #{ user**.**first\_name }"

* Preferí escribir los strings con una sola comilla cuando no tengás la necesidad de realizar interpolación o usar símbolos especiales como \t, \n, ', etc.

*# mal*

name **=** "Bozhidar"

*# bien*

name **=** 'Bozhidar'

* No dejes de usar {} alrededor de las variables de instancia o globales, siendo interpolados dentro de un string.
* Evitá usar String#+ cuando necesites construir un pedazo grande de datos. En su lugar usá String#<<. Concatenación muta la instancia del string en el lugar y siempre es más rápido que String#+, el cual crea un grupo de nuevos objetos de strings.

# Otras consideraciones

* Evitá métodos mayores que 10 LOC (lines of code). Idealmente, la mayoría de los métodos van a ser menores que 5 LOC. Líneas vacías no cuentan como LOC relevantes.
* Evitá listas de parámetros mayores que tres o cuatro parámetros.
* Si realmente necesitás métodos "globales", agregalos a tu Kernel y convertilos en private.
* Usá variables de instancia en módulos en lugar de variables globales.

*# mal*

$foo\_bar **=** 1

*# bien*

**module** Foo

**class** **<<** self

**attr\_accessor** :bar

**end**

**end**

Foo**.**bar **=** 1

* Evitá alias cuando alias\_method hace mejor el trabajo.
* Preferí Time.now por sobre Time.new cuando estés leyendo la hora del sistema.
* Escribí código en forma funcional, evitando mutación cuando eso tenga sentido.
* No mutes argumentos excepto que ese sea el propósito del método.
* Evitá más de tres niveles de anidación de bloques.

# HTML

Se utilizarán los tags de html5, escritos en minúscula.

# CSS

No se incluirá código css inline en el html, se utilizarán archivos css separados.

# JAVASCRIPT

No se incluirán scripts en el html, se utilizarán archivos js separados.