2. つまり、Rubyはプログラマの腕がコンスタントに試される言語であるわけだ。
   多様性ゆえに、初学者が愚かで非効率なコードを書くことも容易だし、腕のいいプログラマが書けば、非常に効率的で美しいコードになる。
   だからこそ、わたしはRubyが好きなのだ。
3. そんなことを考えていると、以下の記事を見つけた。
   Refactor to use Ruby standard library
4. ここには、Rubyの初学者が陥りやすい標準ライブラリにまつわるアンチパターンが列挙されている。
   リンク先のコードの引用になるが、以下のようなコードを書く人間は、**驚くほど多い**。
5. [コードをここに]
6. 破壊的メソッドを使っているのに、わざわざ同じインスタンスに代入し直すような実装も、実際に見かけるから恐ろしい。
7. また、Railsの初学者にありがちなのが、以下のようなコントローラのコードである。
8. [コードをここに]
9. ここには2つムダがある。おわかりだろうか？
   まず1つは、@numberをインスタンス変数にしている点。ビューで@numberを使うならこれで良いのだが、ビューで使われていないのにわざわざインスタンス変数にしているコードが散見される。
   どうしてわざわざオブジェクトのスコープを不必要に広げるような実装をするのだろうか？理解に苦しむ。
   2つめは、3行目のto_iメソッド。ActiveRecordのlimitメソッドは、引数の型を吸収してくれるので、to_iメソッドが整数を返すオブジェクトを渡せばよしなにしてくれる。だから3行目のto_iは不要である。
   もっというと、この程度なら3行目と4行目を合わせてしまって、@books = Book.order(:created_at).limit(params[:number])と書いてしまってもよいかもしれない。
   ただし、to_iできないオブジェクトをlimitに渡してしまうと例外を吐くので、事前にチェックするなり例外をキャッチして400を返すなりしなければならない。
10. Railsを構成するライブラリは巨大なので、Rubyの標準ライブラリ以上に使いこなすことが難しい。
    そのため、Railsではリンク先にあるようなアンチパターンの宝庫となりえる。
    我々プログラマはRailsを理解し、普段からスマートなコードを書くことを意識せねばならない。