UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
MESTRADO PROFISSIONAL
DISCIPLINA: ENGENHARIA DE SOFTWARE (MPI906)
PROFESSOR: VINICIUS CARDOSO GARCIA (vcg)

LISTA DE EXERCÍCIOS 1: Mais Ruby

Programação em Par na Lista de Exercícios 1

Por favor, veja se você pode participar de uma sessão de programação par ao tentar resolver esta lista de exercícios. Siga as instruções do professor para programação em par, você pode, por exemplo, utilizar o hangout (conta do CIn) para promover a interação com seu par.

Lista de Exercícios 1: Ruby Calisthenics

Primeiramente, prepare-se com o esqueleto de código necessário para esta lista de exercícios. Se você estiver na VM você pode clonar o seguinte repositório:

```
git clone https://github.com/saasbook/ruby-calisthenics
```

O objetivo desta tarefa multiparte é fazer com que você se acostume à codificação básica do Ruby e apresentá-lo ao RSpec, a ferramenta de teste de unidade que iremos utilizar pesadamente.

Enquanto é fornecida uma explicação de como o código deve funcionar, você deve se acostumar com a ideia de que a verdadeira especificação está nos arquivos de teste!

Portanto, sugerimos que você trabalhe nesta atribuição usando autotest, que automaticamente re-executa todos os testes RSpec cada vez que você fizer alterações em seu código:

1. Em uma janela de terminal, mude para o diretório raiz dessa lista de exercícios (o que contém subdiretórios `lib/` e `spec/`) e execute o comando `autotest`. RSpec espera encontrar arquivos de código no diretório `lib/` e os arquivos de especificação correspondentes no `spec/`.
2. Inicialmente, toso os testes estão marcados como "pending", conforme indicado pelo argumento `:pending => true` em cada bloco `describe`. Para começar a trabalhar em uma atividade, remova esta opção:

exemplo: em `fun_with_strings_spec.rb`, modifique:
`describe 'palindrome detection', :pending => true do`
para:
`describe 'palindrome detection' do`
e salve o arquivo spec. autotest detecta a alteração e automaticamente re-executa os testes desse grupo, que agora vão falhar (exibidos em vermelho) desde que você não tenha escrito qualquer código ainda.

1. Conforme você preencha o código nos arquivos apropriados no `lib/`, cada vez que você salvar as alterações, os testes serão automaticamente re-executados. Quando um teste passar, ele é exibido em verde. Seu objetivo é fazer com que todos os testes para todas as partes fiquem verde.

(Nota: quando você enviar a sua resolução, também podemos executar casos de teste adicionais além dos fornecidos aqui.)

Preparando o ambiente Cloud9 para a LE1

Caso esteja no **Cloud9** basta criar uma nova área de trabalho (`workspace`) informando a URL do repositório a ser clonado:

```
https://github.com/saasbook/ruby-calisthenics
```

Cloud9 oferece máquinas virtuais Ubuntu que funcionam normalmente como qualquer VM. As VMs no Cloud9 vêm com várias versões diferentes do Ruby, que são controlados pelo sistema chamado `rvm`. Para visualizar suas versões de Ruby, vá para o terminal do Cloud9 e digite:

```
rvm list
```

A versão padrão do Ruby em uma VM do Cloud9 é `ruby-2.1.4`. Você vai precisar mudar para uma versão mais antiga (pelo menos, porque, para mim, a `gem debugger` não funcionou no ambiente padrão do Ruby, acho que requer uma versão mais antiga do que o Ruby 2.0 para funcionar corretamente). Assim, no terminal de Cloud9, digite:

```
rvm use ruby-1.9.3-p547
```

Em seguida, você precisa instalar as gems `rspec`, `debugger` e `autotest`. Vá em frente e digite no terminal do Cloud9:

```
gem install rspec
```

```
gem install debugger
```

```
gem install autotest
```

Mas há um porém! Se você buscar por arquivos ocultos em sua pasta de trabalho:

```
ls -a
```

você vai observar que você tem um arquivo `.rspec`. Ele contém a linha: `--format` aninhada, o que lhe dará todos os tipos de problema, porque a versão do rspec que será instalada com o comando `gem install rspec` é a ***3.1.<alguma coisa>***, e esta versão não entende a linhas `--format` aninhadas. Existem várias maneiras de contornar esse problema, mas eu aqui as que funcionaram para mim:

- instalar o bundler: `gem install bundle`
- No terminal do Cloud9, digite `bundle init`. Você verá na sua área de trabalho um arquivo chamado `Gemfile`
- Abra o arquivo `Gemfile` e adicione as seguintes linhas no final do arquivo:

  ```
  gem 'rspec', '~> 2.14'

  gem 'debugger'

  gem 'autotest'
  ```

- Salve o arquivo `Gemfile`
- No terminal do Cloud9 digite: `bundle`

Agora está tudo pronto! Tente digitar `rspec` no console. Ou, tente digitar `autotest`. Ambos devem funcionar. A propósito, para parar de `autotest`, você precisa pressionar `Ctrl-C` duas vezes.

Caso não funcione, olhar esta dica.

LE 1-1: Fun with Strings

Specs: `spec/fun_with_strings_spec.rb`

Neste problema, você vai implementar três funções que executam processamento básico de string. Você pode começar a partir do modelo `fun_with_strings.rb`

Parte A - Palíndromos

Um palíndromo é uma palavra ou frase em que você lê os mesmos caracteres para a frente como para trás, ignorando caixa, pontuação e pseudopalavra. (Uma "pseudopalavra" é

definida para os nossos propósitos como "um caracter que expressões regulares de Ruby tratariam como um caracter diferente de palavra".)

DICA: Você pode usar a expressão regular `/[^a-z]/` para atestar que a palavra é bem formada.

Você vai escrever um método `palindrome?` que retorna **true** se e somente se o seu receptor é um **palíndromo**.

Como você pode verificar no modelo `fun_with_strings.rb`, demos um jeito de misturar o seu método com a classe String para que ele possa ser chamado assim:

```
"redivider".palindrome?      # =>  deve retornar true
"adam".palindrome?           # =>  deve retornar false ou nil
```

Sua solução não deve usar laços ou iteração de qualquer espécie. Em vez disso, você vai encontrar a sintaxe de expressão regular útil; ela é revisada brevemente no site rubular.com que permite que você experimente expressões regulares em Ruby "ao vivo". Alguns métodos que você pode achar útil (e que você vai ter que olhar na documentação Ruby, ruby-doc.org) incluem: `String#downcase`, `String#gsub`, `String#reverse`. Olhem os exemplos de funcionamento destes métodos!

O arquivo spec contém uma série de casos de teste. No mínimo, todos devem passar antes de você me enviar seu código. Podemos executar casos adicionais também para verificar a conclusão do seu exercício.

Após a conclusão faça o download do arquivo e envie este arquivo para o email do professor.

Parte B - Word Count

Definir uma função `count_words` que, dada uma cadeia de entrada, retornar um hash cujas chaves são palavras na string e cujos valores são o número de vezes que cada palavra aparece:

```
"To be or not to be" # => {"to"=>2, "be"=>2, "or"=>1, "not"=>1}
```

Sua solução não deve usar laços for, mas iteradores como `each` são permitidos. Como antes, "pseudopalavras" e caixa devem ser ignoradas. Uma palavra é definida como uma seqüência de caracteres (string) entre limites de palavra.

DICA: Você pode usar a expressão regular `/[!.,-]/` para verificar se é uma "pseudopalavra".

Após a conclusão faça o download do arquivo e envie este arquivo para o email do professor.

Parte C - Anagramas

Um grupo anagrama é um grupo de palavras de tal modo que qualquer uma pode ser convertido em qualquer outra apenas por rearranjo das letras. Por exemplo, "rats", "tars" e "stars" são um grupo anagrama.

Dada uma lista de palavras separadas por espaço em uma única sequência, escrever um método que agrupe-as em grupos anagrama e retorna um array de grupos. Caixa não importa na classificação da string como anagrama (mas a caixa deve ser preservada na saída), e a ordem dos anagramas nos grupos, não importa.

Após a conclusão faça o download do arquivo e envie este arquivo para o email do professor.

LE 1-2: Programação Orientada a Objetos Básica para a Sobremesa

Specs: `spec/dessert_spec.rb`

1. Crie uma classe `Dessert` com `getters` e `setters` para nome (`name`) e calorias (`calories`). O construtor deve aceitar argumentos para nome e calorias.
2. Defina os métodos de instância `healthy?`, que retorna **true** se e somente se uma sobremesa tem **menos de 200 calorias** e `delicious?`, que retorna **true** para **todas** as sobremesas.
3. Criar uma classe `JellyBean` que herda de `Dessert`. O construtor deve aceitar um único argumento que dá o sabor (`flavor`) da jujuba; uma jujuba recém-criada deve ter 5 calorias e seu nome deve ser o sabor concatenado com "*jelly bean*", por exemplo, "strawberry jelly bean".
4. Adicionar `getter` e `setter` para o sabor.
5. Modificar `delicious?` para retornar **false** se o sabor é `licorice` (alcaçuz), mas **true** para **todos** os outros sabores. O comportamento de `delicious?` para sobremesas que não sejam jujuba (*non-jelly-bean*) deve manter-se inalterado.

Após a conclusão faça o download do arquivo e envie este arquivo para o email do professor.

LE 1-3: Pedra Papel Tesoura

Specs: `spec/rock_paper_scissors_spec.rb`

Em um jogo de pedra-papel-tesoura, cada jogador escolhe jogar pedra (R, *rock*), papel (P, *paper*), ou Tesoura (S, *scissors*). As regras são: Pedra quebra Tesoura, Tesoura corta o Papel, mas o Papel embrulha Pedra.

Em uma partida de pedra-papel-tesoura, o nome e a estratégia (escolha) de cada jogador é codificado como uma matriz de dois elementos, a sabe:

```
[ ["Armando", "P"], ["Dave", "S"] ] # Dave would win since S > P
```

1. Jogo Vencedor:

Crie uma classe `RockPaperScissors` com um método `winner` que tem duas matrizes de 2 elementos como aqueles acima, e retorna um representando o vencedor:

```
RockPaperScissors.winner(['Armando','P'], ['Dave','S'])  # =>
['Dave','S']
```

Se a estratégia do jogador é algo diferente de "R", "P" ou "S" (maiúsculas e/ou minúsculas), o método deve lançar uma exceção do tipo "RockPaperScissors::NoSuchStrategyError" e fornecer a mensagem: "Strategy must be one of R,P,S"

Se ambos os jogadores usam a mesma estratégia, o primeiro jogador é o vencedor.

2. Torneio:

Um torneio de pedra-papel-tesoura é codificado como uma série de jogos - onde, cada elemento pode ser considerado o seu próprio torneio.

```
[
    [
        [ ["Armando", "P"], ["Dave", "S"] ],
        [ ["Richard", "R"],  ["Michael", "S"] ],
    ],
    [
```

```
        [ ["Allen", "S"], ["Omer", "P"] ],
        [ ["David E.", "R"], ["Richard X.", "P"] ]
    ]
]
```

No torneio acima Armando sempre vai jogar P e Dave sempre jogar S. Este torneio se desenrola da seguinte forma:

Sob este cenário, Dave iria bater Armando (S>P) e Richard iria bater Michael (R>S), então Dave e Richard iriam jogar (Richard vence desde que R>S); Da mesma forma, Allen iria bater Omer, Richard X. iria bater David E., e Allen e Richard X. iriam jogar (Allen ganha desde que S>P); e, finalmente, Richard iria bater Allen desde R>S. Isto é, a execução em pares continua até só existir um único vencedor.

Escreva um método `RockPaperScissors.tournament_winner` que recebe um torneio codificado como uma matriz e retorna o vencedor (para o exemplo acima, ele deve retornar `['Richard', 'R']`). Você pode assumir que a matriz está bem formada (ou seja, há 2^n jogadores, e cada um participa em exatamente um jogo por rodada).

DICA: Formular o problema como um método recursivo cujo caso base você resolveu na parte 1.

LE 1-4: RUBY METAPROGRAMMING

Specs: `spec/attr_accessor_with_history_spec.rb`

Na aula, vimos como o `attr_accessor` usa metaprogramação para criar getters e setters para os atributos do objeto em tempo real.

Defina um método `attr_accessor_with_history` que fornece a mesma funcionalidade que `attr_accessor` mas também rastreia cada valor do atributo já teve:

```
class Foo
  attr_accessor_with_history :bar
end

f = Foo.new
f.bar = 3           # => 3
f.bar = :wowzo      # => :wowzo
f.bar = 'boo!'      # => 'boo!'
f.bar_history       # => [nil, 3, :wowzo]
```

(*Chamar o bar_history antes do setter de bar ter sido chamado deve retornar nulo.*)

O histórico das variáveis de instância deve ser mantida separadamente para cada ocorrência do objeto. Ou seja:

```
f = Foo.new
f.bar = 1 ; f.bar = 2
g = Foo.new
g.bar = 3 ; g.bar = 4
g.bar_history
```

assim, a última linha deve retornar apenas `[nil, 3]`, ao invés de `[nil, 1,3]`.

Se você estiver interessado em como o modelo funciona, a primeira coisa a se notar é que se definirmos `attr_accessor_with_history` na classe `Class`, podemos usá-lo como no trecho acima. Isto ocorre porque uma classe de Ruby como `Foo` ou `String` é, na verdade, apenas um objeto da classe `Class`. (Se isso faz com que seu cérebro entre em curto, relaxe e não se preocupe com isso agora.)

A segunda coisa a notar é que o Ruby fornece um método `class_eval` que recebe uma satring e a avalia no contexto da classe atual, ou seja, a classe da qual você está chamando `attr_accessor_with_history`. Esta string terá de conter uma definição de método que implementa um setter-with-history para o atributo `attr_name` desejado.

DICAS

- Não se esqueça que a primeira vez que o atributo recebe um valor, a sua matriz histórica terá de ser inicializada.
- O valor inicial de um atributo é sempre nulo por padrão, por isso, se `foo_history` é referenciado antes de `foo` já ter sido atribuído, a resposta correta é `nil`, mas depois da primeira designação para `foo`, o valor correto para `foo_history` seria `[nil]`.
- Não se esqueça de que variáveis de instância são referenciadas como `@bar` dentro de getters e setters.
- Embora o `attr_accessor` existente possa lidar com vários argumentos (por exemplo `attr_accessor :foo, :bar`), a sua versão apenas precisa lidar com um único argumento.
- A sua implementação deve ser genérica o suficiente para trabalhar no contexto de qualquer classe e para os atributos de qualquer (válida) nome da variável.
- Note que uma característica poderosa da metaprogramação em Ruby é `class_eval` que pode ser chamado na meta-classe `Class`. `class_eval` pode interpretar uma sequência de caracteres em tempo real para criar um código novo. No exemplo

abaixo, foi definido o `add_method()` para a meta-classe (e, por herança, a qualquer classe). Quando chamado, este método define um novo método que deve retornar 42 (observe como `#{name}` é substituído com o parâmetro passado para `add_method`).

```
class Class
  def add_method(name)
    class_eval %Q"
      def #{name}()
        42
      end
      "
    end
  end

class MyClass
  add_method :my_method
end

mc = MyClass.new
puts mc.my_method # => 42
```

Após a conclusão faça o download do arquivo e envie este arquivo para o email do professor.