Como LLM pode promover o aprendizado e o desenvolvimento de aplicações Ruby?

- No arquivo homework01 – Resolution.rb estão os códigos e suas devidas explicações.
- Nesse relatório iremos abordar alternativas e oferecer um *feedback* sobre o trabalho.
- Todas as implementações alternativas estão no final do documento.

Propor implementações alternativas e suas vantagens/desvantagens?

- PART 01
  As alternativas propostas nesse exercício consistem em:
  Utilizar os métodos *char* e *select:*
  Em *'def palindrome?' t*orna-se vantajoso essa alternativa quando buscamos maior performance em *strings* grandes, contudo o código é mais complexo de ler e entender além da ocorrência de *strings* pequenas tornar o código menos eficiente.

  Utilizar método *group_by:*
  Mais lento para grandes listas, torna-se vantajoso para evitar criação de novos hash além de mais conciso.

- PART 02
  A utilização de um hash para armazenar as regras de vitória em *rps_game_winner* , torna o código facilmente expansível e modificável, porém em sistemas muito limitados, o uso de hash ocupa maior quantidade de memória.

  Como opção em *rps_tournament_winner* podemos optar por **um loop iterativo** o qual é mais complexo de entender, contudo, evita stack overflow.

- PART 03
  Tornando o código curto e simples, **o método *group_by*** elimina a necessidade de manipular um hash manualmente ao preço de possuir um desempenho levemente inferior para listas pequenas.

- PART 04
  Ao utilizar hash para armazenar atributos, o código possui facilidade de adicionar novos atributos futuros, seu grande ponto fraco é não aproveitar o *"att_acessor"*, o que a longo prazo confunde para manter o código.

- PART 05
  Ao **utilizar um módulo para adicionar o histórico**, tornamos o código mais fácil de compreender e isolar, essa ação atribui maior complexidade e mesmo facilitando o entendimento, torna menos direto a compreensão.

- PART 06

Ao **utilizar uma biblioteca de conversão de moedas *"money",*** torna mais precisa a conversão, porém agora o código possui dependência externa e é mais complexo em casos simples de conversão

Uma segunda ideia é **utilizar um hash para *'@@curencies',*** o que reduz a necessidade de variáveis de classe contudo torna as taxas globais, a qual pode ser um problema em códigos mais extensos e complexos

Feedback final

Esse feedback vem de alguém que está tendo o primeiro contato com a linguagem Ruby. Posso afirmar que a utilização de um modelo de linguagem (LLM) facilitou o entendimento e a compreensão da linguagem. Ler as solicitações realizadas e acompanhar a criação do código em tempo real fez com que a compreensão de Ruby se tornasse mais natural. O tempo de dedicação foi moderado, entre 3 e 4 horas para entender o básico. A análise das respostas foi realizada por meio de testes próprios.

De modo geral, o uso de uma LLM foi proveitoso para promover o aprendizado e o desenvolvimento em Ruby. Sem dúvidas, essa tecnologia facilitou a jornada de aprendizado.

Códigos alternativos

---------- PART 01 ----------

```
def palindrome?(string)
  cleaned_string = string.downcase.chars.select { |char| char =~ /\w/ }.join
  cleaned_string == cleaned_string.reverse
End


def count_words(string)
  words = string.downcase.scan(/\b\w+\b/)
  words.group_by(&:itself).transform_values(&:count)
end
```

---------- PART 02 ----------

```
def rps_game_winner(game)
  raise WrongNumberOfPlayersError unless game.length == 2
```

```
  rules = { "R" => "S", "P" => "R", "S" => "P" }
  player1_name, player1_strategy = game[0]
  player2_name, player2_strategy = game[1]

  player1_strategy = player1_strategy.upcase
  player2_strategy = player2_strategy.upcase
  valid_strategies = rules.keys

  raise NoSuchStrategyError unless valid_strategies.include?(player1_strategy) &&
valid_strategies.include?(player2_strategy)

  if player1_strategy == player2_strategy || rules[player1_strategy] == player2_strategy
    [player1_name, player1_strategy]
  else
    [player2_name, player2_strategy]
  end
end

def rps_tournament_winner(tournament)
  while tournament.length > 1
    tournament = tournament.each_slice(2).map { |game| rps_game_winner(game.flatten(1)) }
  end
  tournament.first
end
```

---------- PART 03 ----------

```
def combine_anagrams(words)
  words.group_by { |word| word.downcase.chars.sort.join }.values
end
```

---------- PART 04 ----------

```
class Dessert
  def initialize(name, calories)
    @attributes = { name: name, calories: calories }
  end

  def name
    @attributes[:name]
  end

  def calories
    @attributes[:calories]
  end

  def healthy?
    @attributes[:calories] < 200
  end

  def delicious?
    true
  end
end
```

---------- PART 05 ----------

```
module AccessorWithHistory
  def attr_accessor_with_history(attr_name)
    attr_name = attr_name.to_s
    ivar_name = "@#{attr_name}"
```

```
    define_method(attr_name) { instance_variable_get(ivar_name) }
    define_method("#{attr_name}_history") { instance_variable_get("@#{attr_name}_history") }

    define_method("#{attr_name}=") do |value|
      history = instance_variable_get("@#{attr_name}_history") || [nil]
      instance_variable_set("@#{attr_name}_history", history << value)
      instance_variable_set(ivar_name, value)
    end
  end
end

class Class
  include AccessorWithHistory
end
---------------------------------------------------------- PART 06 ----------------------------------------------------------
CURRENCIES = { yen: 0.013, euro: 1.292, rupee: 0.019, dollar: 1 }
```