Homework 01

[https://github.com/Henrique-Givisiez/Engenharia-Software-Estudo-Dirigido-1](https://github.com/Henrique-Givisiez/Engenharia-Software-Estudo-Dirigido-1https://github.com/Henrique-Givisiez/Engenharia-Software-Estudo-Dirigido-1)

Henrique Givisiez dos Santos - 211027563

Departamento de Ciência da Computação

Universidade de Brasília

Brasília, Brasil

# Introdução

O estudo dirigido realizado tem como objetivo praticar os conhecimentos e fundamentos da linguagem Ruby ministrados em sala de aula na disciplina Engenharia de Software, por meio de exercícios propostos. Dessa forma, buscando solidificar o aprendizado e entender como o desenvolvimento de aplicações em Ruby funciona, será utilizada uma *Large Language Model* (LLM) que irá auxiliar na explicação de funções e elementos básicos de Ruby, bem como a sintaxe correta para a escrita dos códigos dos exercícios. No que diz respeito a LLM utilizada, cabe destacar o GPT-4 criado pela OpenAI como a escolha para este estudo.

# LLMs como aliadas do aprendizado

Para iniciar o estudo, é necessário destacar como a utilização de modelos de linguagem de grande escala (LLMs), pode ser um recurso poderoso no aprendizado da linguagem Ruby, especialmente para quem adota uma abordagem prática e baseada em exercícios. O GPT-4, especificamente, se destaca ao fornecer explicações detalhadas, resolver dúvidas e guiar o usuário na execução e compreensão de diferentes desafios de código em Ruby.

Durante a conversa feita com o GPT-4, foram explorados diversos aspectos da linguagem Ruby com foco em resolver exercícios práticos. O GPT-4 desempenhou um papel importante ao fornecer exemplos personalizados e detalhados, esclarecendo conceitos fundamentais, como:

1. **Estrutura e Sintaxe**: O modelo ajudou a entender a sintaxe básica de Ruby, como a definição de métodos, a utilização de *loops each*, variáveis de instância e variáveis de classe. Esses conceitos foram abordados com exemplos contextualizados, permitindo que cada parte do código fosse explicada de forma detalhada.

1. **Manipulação de Strings e Explicações em Código**: Foram trabalhados métodos específicos de *strings*, como *gsub*, *downcase*, e *reverse*, que são essenciais para manipulação de texto. Esses métodos foram aplicados em exercícios que incluíam verificação de palíndromos e processamento de texto, com o GPT-4 explicando cada linha e o propósito dos métodos usados. Esse tipo de exercício é essencial para solidificar o entendimento de manipulação de strings, um aspecto crucial em Ruby.1

1. **Criação de Métodos Personalizados e Extensões de Classes**: Um dos exercícios explorou a criação de um método personalizado *attr\_accessor\_with\_history* que permitia registrar o histórico de valores de um atributo. Esse exemplo introduziu conceitos mais avançados de Ruby, como manipulação dinâmica de classes e o uso de métodos como *class\_eval*. Com o GPT-4, foi possível entender cada etapa da implementação, incluindo o propósito e o funcionamento das variáveis de instância e de classe.
2. **Tratamento de Erros**: Em outro exercício, foi implementado um jogo de pedra, papel e tesoura, usando classes de erro personalizadas para interceptar chamadas de métodos com parâmetros não definidos.

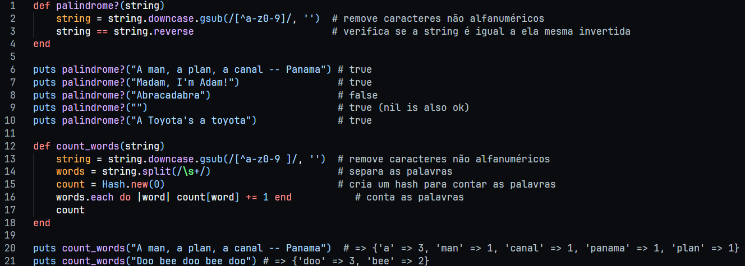
1. **Implementação de Conversores de Moedas com method\_missing**: Foi explorada também a utilização do *method\_missing* para construir um conversor de moedas, onde o GPT-4 guiou na aplicação de expressões regulares para reconhecer e tratar pluralidade, além de demonstrar como manipular dados em variáveis de classe.

A habilidade do modelo de explicar conceitos complexos com uma linguagem simples e contextualizada é extremamente valiosa para quem aprende Ruby. Ele permite explorar tópicos como manipulação de classes, métodos dinâmicos e tratamento de exceções com clareza e praticidade.

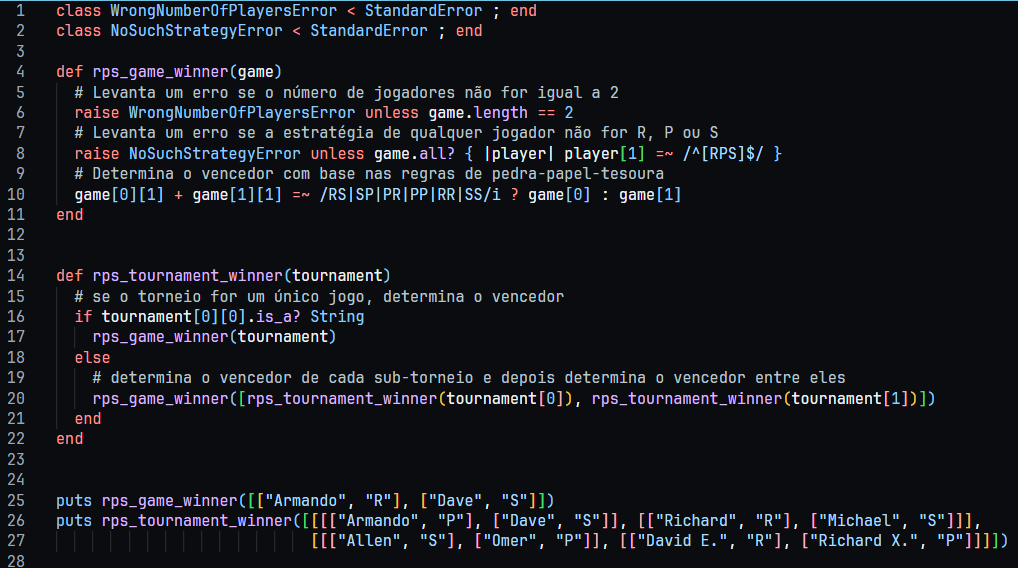
Em resumo, o GPT-4 torna o aprendizado de Ruby mais acessível e interativo, fornecendo explicações passo a passo, exemplos aplicados e respostas imediatas a dúvidas. Essa combinação de assistência técnica e personalização de conteúdo promove uma compreensão mais profunda e prática da linguagem, transformando o processo de aprendizado em uma experiência guiada e contínua.

# Exercícios

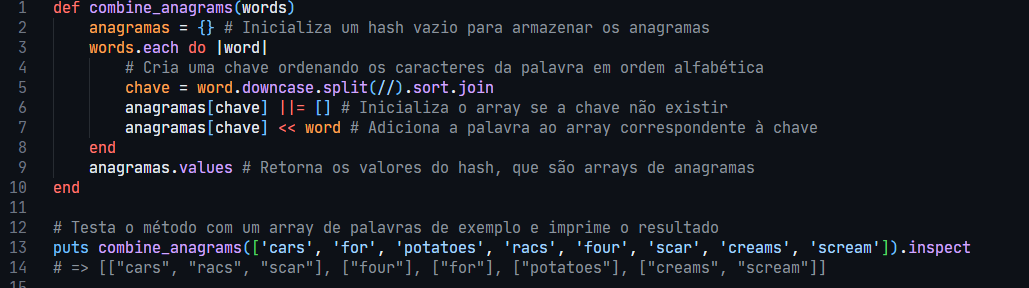
No que diz respeito aos exercícios citados, inicia-se com o primeiro exercício que propõe a elaboração de um método que determina se uma frase ou palavra é um palíndromo. Um palíndromo é uma palavra, frase ou número que permanece igual quando lida de trás para diante. Por extensão, palíndromo é qualquer série de elementos com simetria linear, ou seja, que apresenta a mesma sequência de unidades nos dois sentidos. Além disso, também foi solicitado que retornasse um *hash* em que as chaves fossem as palavras encontradas na *string* de entrada e a frequência de cada uma dessas palavras associadas como valores das chaves. Foi proposto ainda que este *hash* não fizesse uso de *for-loops,* apenas iteradores *each*. Uma estrutura *hash* trata-se de uma coleção de pares chave-valor.

Figura 1 - Exercício 1 (*Fun with Strings*)

Para ilustrar a importância de tratamento de erros, o segundo exercício trata do desenvolvimento de um código que simulasse um torneio de partida do jogo pedra-papel-tesoura. O código do jogo de pedra-papel-tesoura ilustra o tratamento de erros em Ruby ao definir duas classes de erro personalizadas: ***WrongNumberOfPlayersError*** e ***NoSuchStrategyError***. Essas classes são utilizadas para lançar exceções específicas, garantindo que o jogo tenha o número correto de jogadores e que cada jogador utilize uma estratégia válida (apenas "R", "P" ou "S").

Figura 2 - Exercício 2 (*Rock-Paper-Scissors*)

O terceiro exercício envolve a criação de um método que dado um *array* de *strings* como entrada, agrupa os anagramas de cada *string* e retorna um *array* com os mesmos agrupados. Um anagrama é uma espécie de jogo de palavras criado com a reorganização das letras de uma palavra ou expressão para produzir outras palavras ou expressões, utilizando todas as letras originais exatamente uma vez. A solução utilizada para este exercício foi baseada em um *hash* que tem como chaves as *strings* ordenadas e como valores associadas a cada chave, os anagramas destas *strings*.

Figura 3 – Exercício 3 (*Anagrams)*

O Ruby é considerado uma linguagem puramente orientada a objetos, visto que tudo é considerado um objeto, até mesmo os tipos básicos da linguagem. Dessa forma, com o intuito de consolidar os conhecimentos em programação orientada a objetos, os próximos exercícios (exercícios 4-6) requerem a criação de classes, bem como métodos *getters* e *setters* para lidar com instâncias da classe. No exercício 4, foi demandado a criação de 2 classes: *Dessert* e *JellyBean* que demonstravam o funcionamento dos *getters* e *setters*.

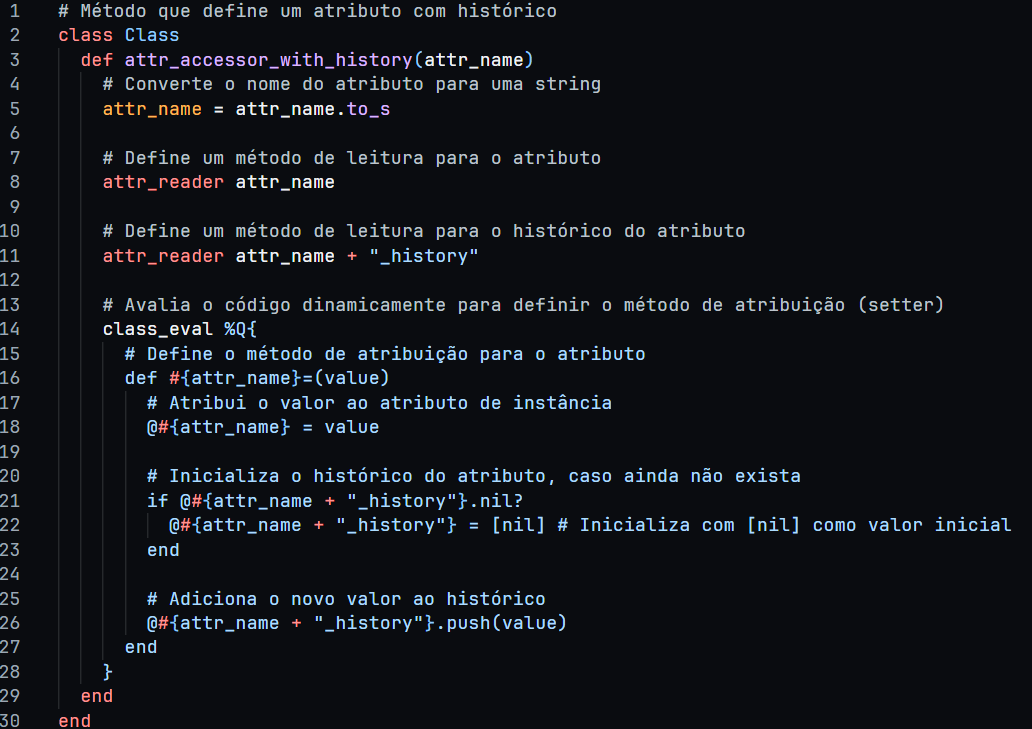
Tela de computador com texto preto sobre fundo branco

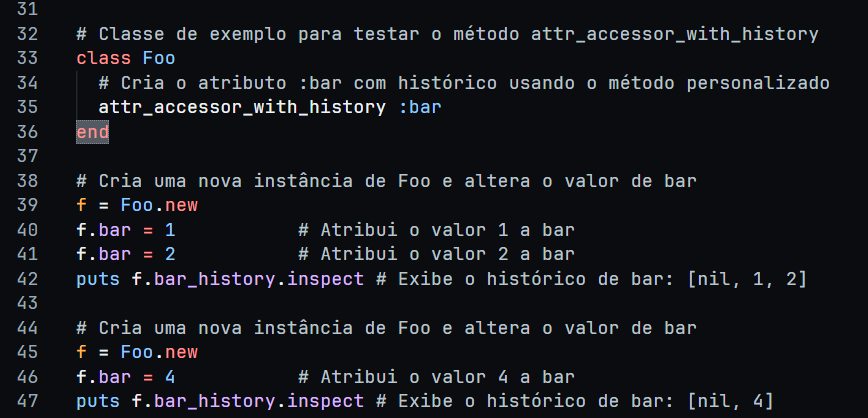
Descrição gerada automaticamenteFigura 4 - Exercício 4 (*Basic OOP*)– parte 1

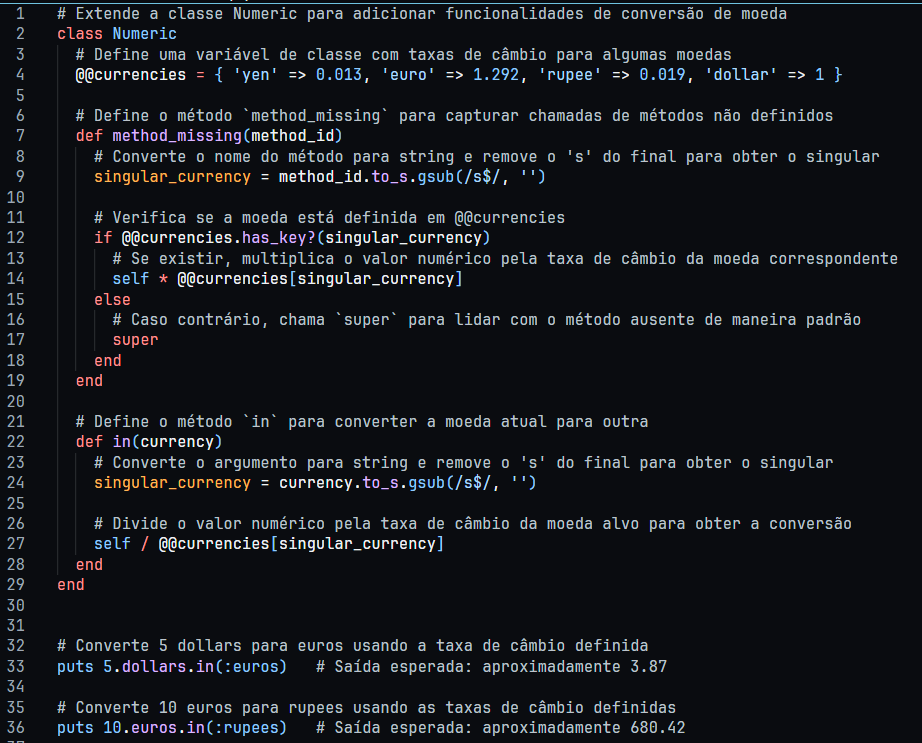
Texto

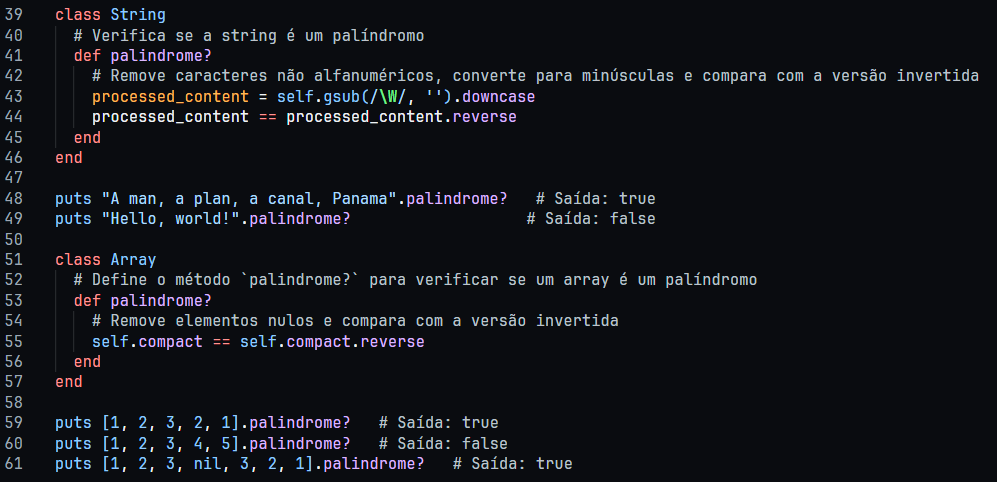
Descrição gerada automaticamenteFigura 5 – Exercício 4 (*Basic OOP*) – parte 2

Nos exercícios 5 e 6, o objetivo foi aprofundar o aprendizado em programação orientada a objetos por meio de métodos personalizados, encapsulamento, metaprogramação e polimorfismo.

Figura 6 - Exercício 5 (*advanced OOP*) - parte 1

Figura 7 - Exercício 5 (*advanced OOP*) - parte 2

Figura 8 - Exercício 6 (*advanced OOP*)- parte 1

Figura 9 - Exercício 6 (*advanced OOP*) - parte 2

É importante destacar que se nota uma repetição desnecessária de código para os exercícios 1 e 6, visto que em ambos os exercícios a função *palindrome?* é utilizada para realizar o mesmo processo. Sendo assim, é evidente que neste caso poderia ser vantajoso usar um *mixin* para a função *palindrome?*. Em Ruby, *mixins* são implementados usando módulos, que permitem compartilhar métodos entre classes diferentes sem duplicação de código.

# Conclusão

A experiência de aprendizado com Ruby, fundamentada nos exercícios propostos, foi enriquecedora e estimulante. Com um esforço e tempo moderados, foi possível não apenas aprender a sintaxe e as particularidades da linguagem, mas também compreender seus conceitos orientados a objetos, que são essenciais para desenvolver aplicações bem estruturadas e reutilizáveis.

Os exercícios abordaram aspectos fundamentais da POO, como encapsulamento, polimorfismo, metaprogramação e *mixins*. Por exemplo, o exercício que desafiou a criar o método *attr\_accessor\_with\_history* foi excelente para entender o poder das variáveis de instância e o controle de histórico em atributos, demonstrando a flexibilidade e as possibilidades de personalização em Ruby. Esse exercício ajudou a explorar como é possível manipular atributos de forma dinâmica, utilizando metaprogramação para definir métodos personalizados.

Outro ponto alto foi a implementação de um conversor de moedas, que aplicou o método *method\_missing*. Esse exercício foi particularmente valioso para compreender o polimorfismo e a criação de métodos dinâmicos em Ruby. O uso de *method\_missing* para interceptar chamadas de métodos desconhecidos reforçou a compreensão sobre como Ruby permite lidar com comportamentos específicos de maneira eficiente, explorando o potencial de métodos que não precisam ser declarados diretamente.

A criação de um método *palindrome?*, estendido tanto para a classe *String* quanto para *Array*, destacou o uso de *mixins* e módulos. Esse exercício proporcionou uma introdução prática aos módulos em Ruby, mostrando como métodos podem ser compartilhados entre classes diferentes sem duplicação de código, promovendo a reutilização e modularidade — princípios essenciais da programação orientada a objetos.

Em resumo, o aprendizado com Ruby foi extremamente positivo. A combinação de teoria e prática nos exercícios permitiu uma compreensão sólida e aplicável da linguagem. Ruby se mostrou intuitiva e poderosa, com uma expressividade que facilita o desenvolvimento de soluções elegantes e eficientes. Esses exercícios foram fundamentais para consolidar os conhecimentos e proporcionaram uma experiência de aprendizado prática e motivadora.