```
print(f"Número de páginas: {num_paginas}")

# Usando o setter como se fosse uma atribuição direta
livro1.paginas = 350
print(f"Páginas atualizadas: {livro1.paginas}")

livro1.paginas = -50 # Tentativa inválida
print(f"Páginas após tentativa inválida: {livro1.paginas}")

# Você não pode alterar o título ou autor diretamente, pois nã
# livro1.titulo = "Novo Título" # Isso causaria um AttributeEr
```

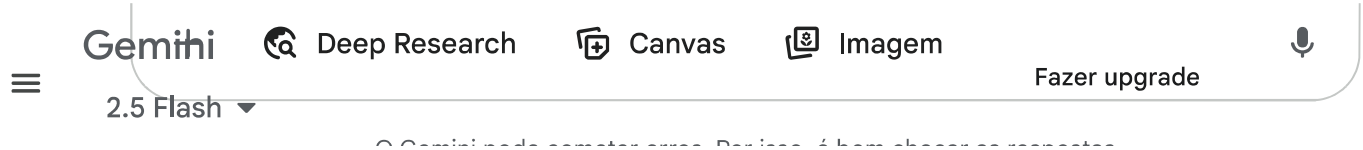
Explicação do `@property`:

- `@property` **acima do método** `paginas` **(o primeiro):** Transforma o método `paginas` em um **getter**. Agora, quando você acessa `livro1.paginas`, este método é invocado e retorna `self._paginas`.
- `@paginas.setter` **acima do método** `paginas` **(o segundo):** Transforma o método `paginas` (o mesmo nome que o getter, mas agora com a anotação `.setter`) em um **setter**. Quando você faz `livro1.paginas = valor`, este método é invocado com `valor` como argumento.

Quando Usar Getters e Setters vs. Acesso Direto?

A "filosofia" Python geralmente favorece o **acesso direto aos atributos** quando não há necessidade de lógica adicional (validação, transformação) ao obter ou definir o valor. Isso é conhecido como o "Princípio do Acesso Uniforme".

- **Acesso Direto (** `obj.atributo = valor` **):** Use quando o atributo não precisa de validação, cálculo ou qualquer lógica especial ao ser


Gemini
Deep Research
Canvas
Imagem
Fazer upgrade
2.5 Flash