



Imersão em GenAI para Desenvolvedores

Projeto 4: *Prompt Engineering* Avançando

Preâmbulo: Neste projeto, você vai explorar técnicas avançadas de engenharia de prompt para aprimorar o uso de modelos linguísticos em diversas aplicações.

Versão: 1.0

Sumário

I	Preâmbulo	2
II	Instruções gerais	4
III	Ex00 – Guie pelo exemplo (<i>multishot prompting</i>) para orientar o comportamento	6
III.1	Exemplo 1: Analisar o feedback do cliente	7
III.2	Exercício	7
IV	Ex01 – Pré-preenchimento da resposta para uma saída mais precisa	10
IV.1	Exemplo 1: Controlando a formatação da saída	11
IV.2	Exercício	12
V	Ex02 – Encadeie prompts complexos para aumentar o desempenho	14
V.1	Exemplo 1: Analisar e resumir informações	16
V.2	Exercício	17
VI	Entrega e Avaliação entre pares	19
VI.1	Processo de Entrega	19
VI.2	Avaliação entre pares	19
VI.3	Dicas para uma avaliação bem-sucedida	19

Capítulo I

Preâmbulo

*Em matemática, uma **cadeia de Markov** é um processo estocástico que descreve uma sequência de eventos possíveis, em que a probabilidade de cada evento depende apenas do estado alcançado no evento anterior.*

Informalmente, isso pode ser pensado como:

"O que acontece a seguir depende apenas do estado atual das coisas"

Princípios fundamentais:

1. Definição:

- Um processo de Markov é um processo estocástico que satisfaz a propriedade de Markov
- Frequentemente caracterizado como "sem memória"

2. Tipos de cadeias de Markov:

- Requerem especificação do espaço de estados do sistema
- Necessitam definição do índice de parâmetro de tempo

3. Transições:

- Mudanças de estado do sistema
- Elemento central da modelagem Markoviana

Em essência, uma cadeia de Markov modela:

Um sistema que se move de um estado para outro ao longo do tempo, com a probabilidade de cada transição dependendo apenas do estado atual, e não do histórico completo do sistema.

Esta propriedade de “memória limitada” confere às cadeias de Markov um poder de modelagem extraordinário, permitindo sua aplicação em uma ampla gama de fenômenos:

- Comportamento de sistemas físicos
- Evolução da linguagem em modelos de IA
- Previsão de padrões climáticos
- Análise de mercados financeiros
- E muito mais...

— *Adaptado de Hayes, 2013*¹

¹Hayes, Brian. "First links in the Markov chain". American Scientist 101.2 (2013): 92.

Capítulo II


Instruções gerais

- Esta página é sua única referência oficial. Não confie em informações não verificadas.
- Os exercícios estão organizados em ordem crescente de complexidade. É essencial dominar cada exercício antes de prosseguir para o próximo.
- Preste atenção às permissões de seus arquivos e pastas.
- Siga rigorosamente o procedimento de entrega para todos os exercícios.
- Seus exercícios serão avaliados por colegas da Imersão.
- Para exercícios em Shell, utilize `/bin/zsh`.
- Mantenha em sua pasta apenas os arquivos explicitamente solicitados nos enunciados.
- Em caso de dúvidas, consulte seus colegas à direita ou à esquerda.
- Utilize recursos como Google, manuais online e a Internet como referência.
- Leia os exemplos com atenção. Eles podem conter requisitos não explicitamente mencionados no enunciado.
- Para exercícios em Python:
 - Use a versão do Python especificada no exercício de configuração do ambiente.
 - Utilize os modelos e provedores sugeridos para garantir tempos de resposta adequados e consistência nos testes.
- Esteja atento a erros em todos os exercícios. Eles raramente são tolerados durante a avaliação.
- **Aviso sobre o uso de ferramentas de AI (como ChatGPT):**
 - O uso de ferramentas como o ChatGPT não deve ser encarado como um substituto para seu próprio esforço e entendimento.
 - O aprendizado efetivo ocorre quando você interage ativamente com o conteúdo: pesquisando, refletindo e aplicando o que aprendeu.
 - Nas avaliações, serão feitas perguntas para avaliar sua compreensão real sobre o assunto.

- E durante as avaliações, seus colegas também avaliarão seu nível de conhecimento.

Capítulo III

Ex00 – Guie pelo exemplo (*multishot prompting*) para orientar o comportamento

	Exercício : 00
Aprender a utilizar exemplos bem elaborados para guiar o comportamento dos modelos linguísticos, melhorando a precisão e qualidade das respostas criando um sistema de classificação de sentimentos para comentários do GitHub	
Pasta de entrega : <i>ex00/</i>	
Arquivos para entregar : <code>sentiment_analyzer.py</code>	

Por que usar exemplos?

- **Precisão:** Exemplos diminuem as chances de as instruções serem mal interpretadas.
- **Consistência:** Exemplos reforçam uma estrutura e um estilo uniformes.
- **Desempenho:** Exemplos bem escolhidos aumentam a capacidade do modelo de lidar com tarefas complexas.

Uma lista de exemplos bem escolhida, junto com instruções claras, faz toda a diferença no desempenho do modelo.

Regras chave para bons exemplos:

- Explique que os exemplos são um guia de como o modelo deve se comportar.
- Mostre o que foi explicado nas instruções na prática.
- Inclua exemplos de como agir em situações fora do comum.

- Mostre quando o modelo deve agir diferente do que faria normalmente.
- Use exemplos relacionados ao que você quer ensinar, mas diferentes dos dados de treinamento (se possível).
- Evite exemplos repetitivos para que o modelo não aprenda coisas erradas, seja variado, mas consistente.

Conclusão: Exemplos são essenciais.

Criando bons exemplos

Para que o modelo funcione da melhor forma possível, seus exemplos devem ser:

- **Relevantes:** Os exemplos devem ser parecidos com as situações que você vai usar de verdade.
- **Variados:** Inclua exemplos de situações incomuns, problemas que podem aparecer e coisas diferentes para que o modelo não aprenda coisas erradas sem querer.
- **Organização:** Coloque seus exemplos entre as tags <exemplo> (se tiver vários, coloque todos dentro de <exemplos>).

III.1 Exemplo 1: Analisar o feedback do cliente

Sem exemplos:

```
Analise este feedback do cliente e categorize os problemas. Use estas categorias: UI/UX, Desempenho, Solicitação de Recurso, Integração, Preços e Outros. Avalie também o sentimento (Positivo/Neutro/Negativo) e a prioridade (Alta/Média/Baixa).
```

```
Aqui está o feedback: {{FEEDBACK}}
```

Com exemplos:

```
Nossa equipe de atendimento ao cliente está sobrecarregada com feedback não estruturado. Sua tarefa é analisar o feedback e categorizar os problemas para nossas equipes de produto e engenharia. Use estas categorias: UI/UX, Desempenho, Solicitação de Recurso, Integração, Preços e Outros. Avalie também o sentimento (Positivo/Neutro/Negativo) e a prioridade (Alta/Média/Baixa). Aqui está um exemplo:
```

```
<exemplo>
```

```
Texto: O novo painel está uma bagunça! Demora uma eternidade para carregar e não consigo encontrar o botão de exportação. Corrijam isso o mais rápido possível!
```

```
Categoria: UI/UX, Desempenho
```

```
Sentimento: Negativo
```

```
Prioridade: Alta
```

```
</exemplo>
```

III.2 Exercício

Você descobriu que a inclusão de exemplos nos prompts pode aprimorar significativamente a precisão, consistência e qualidade das respostas geradas por modelos de linguagem. Essa técnica, chamada de few-shot ou multishot prompting, é especialmente útil em tarefas que demandam saídas estruturadas ou aderência a formatos específicos.

Instruções

1. Implementar uma função `call_llm` que envia um prompt com exemplos para o modelo Gemini 1.5 Flash.
2. Criar uma função `parse_llm_response` para extrair o sentimento da resposta do modelo.
3. Classificar os sentimentos dos comentários do GitHub.

Como o seu programa será chamado:

```
github_comments = [
    {
        "text": "Ótimo trabalho na implementação desta feature! O código está limpo e bem documentado. Isso vai ajudar muito nossa produtividade.",
        "sentiment": ""
    },
    # ... (restante em anexo na página do projeto na Intra)
]

def call_llm(text):
    # Implementar chamada ao modelo com exemplos (multishot)

def parse_llm_response(response):
    # Extrair o sentimento da resposta do modelo

def analyze_sentiments(comments):
    for comment in comments:
        llm_response = call_llm(comment["text"])
        comment["sentiment"] = parse_llm_response(llm_response)

analyze_sentiments(github_comments)

# Imprimir resultados
for comment in github_comments:
    print(f"Texto: {comment['text']}")
    print(f"Sentimento: {comment['sentiment']}")
    print("-" * 50)
```

Saída esperada:

```
Texto: Ótimo trabalho na implementação desta feature! O código está limpo e bem documentado. Isso vai
    ajudar muito nossa produtividade.
Sentimento: Positivo

Texto: Esta mudança quebrou a funcionalidade X. Por favor, reverta o commit imediatamente.
Sentimento: Negativo


...
```



Não deixe a sua chave de API no código submetido. Durante a avaliação, você precisa configurá-la adequadamente.

Capítulo IV

Ex01 – Pré-preenchimento da resposta para uma saída mais precisa

	Exercício : 01
Aprender a utilizar a técnica de pré-preenchimento (<i>prefilling</i>) para guiar as respostas do modelo, melhorando a precisão e consistência do output, especialmente para formatos estruturados como JSON	
Pasta de entrega : <i>ex01/</i>	
Arquivos para entregar : <i>movie_info.py</i>	

Ao usar um modelo, você pode guiar as respostas preenchendo previamente a mensagem. Essa técnica permite direcionar as ações do modelo, impor formatos específicos como JSON ou XML e até ajudar o modelo a manter a consistência do personagem em situações de *role-play*.

Em alguns casos, quando o modelo não está funcionando como esperado, algumas frases pré-preenchidas podem melhorar o desempenho geral. Um pequeno pré-preenchimento pode fazer uma grande diferença!

Para pré-preencher, inclua o texto inicial desejado na mensagem enviada ao modelo (a resposta do modelo continuará de onde a mensagem enviada parou):

```
# Histórico de mensagens com pré-preenchimento
messages = [
    {"role": "user", "content": "What is your favorite color?"},
    {"role": "assistant", "content": "As an AI language model, I don't have personal preferences or favorites. However, if I were to choose a color, I might say blue because"}
]

# Simulação de chamada ao modelo
resposta_do_modelo = modelo(messages)

# Resultado hipotético
print(resposta_do_modelo)
# Saída: As an AI language model, I don't have personal preferences or favorites. However, if I were to choose a color, I might say blue because it is often associated with calmness, depth, and stability.
```

O pré-preenchimento guia o modelo a começar a resposta de uma maneira específica, influenciando o tom e o conteúdo da continuação.



Nas interações com modelos via API, o histórico de mensagens tipicamente inclui três papéis distintos:

- **system:** Define o comportamento geral do modelo.
- **user:** Representa as mensagens do usuário.
- **assistant:** Contém as respostas geradas pelo modelo.

Cada mensagem no histórico deve especificar um desses papéis, permitindo uma clara distinção entre instruções do sistema, entradas do usuário e respostas do modelo.

IV.1 Exemplo 1: Controlando a formatação da saída

Sem pré-preenchimento:

Quem: usuário

Conteúdo:

```
Extraia o nome, data, local e palestrante principal deste anúncio de evento de tecnologia como um objeto JSON:
<anúncio>
Tech Summit 2024
Junte-se a nós para o principal evento de tecnologia do ano!
Data: 15-17 de novembro de 2024
Local: Expo Center Norte, São Paulo
Palestrante principal: Dr. Ada Lovelace, pioneira em computação
</anúncio>
```

Quem: modelo

Conteúdo:

```
Aqui estão as informações extraídas em formato JSON:
{
  "name": "Tech Summit",
  "date": ["15-11-2024", "16-11-2024", "17-11-2024"],
  "place": "Expo Center Norte, São Paulo",
  "main_speaker": "Ada Lovelace"
}
Extraí os seguintes detalhes do anúncio do evento:
Nome: Tech Summit 2024
Data: 15-17 de novembro de 2024
Local: Expo Center Norte, São Paulo
Palestrante principal: Dr. Ada Lovelace
O objeto JSON fornece um formato estruturado que é fácil para os programas analisarem e usarem. Informe-me se precisar de alguma modificação ou tiver outras perguntas!
```

Com pré-preenchimento:

Quem: usuário

Conteúdo:

```
Extraia o nome, data, local e palestrante principal deste anúncio de evento de tecnologia como um objeto JSON:
<anúncio>
Tech Summit 2024
Junte-se a nós para o principal evento de tecnologia do ano!
```

```
Data: 15-17 de novembro de 2024
Local: Expo Center Norte, São Paulo
Palestrante principal: Dr. Ada Lovelace, pioneira em computação
</anúncio>
```

Quem: modelo (pré-preenchido)

Conteúdo:

```
{
```

Quem: modelo

Conteúdo:

```
"name": "Tech Summit",
"date": ["15-11-2024", "16-11-2024", "17-11-2024"],
"place": "Expo Center Norte, São Paulo",
"main_speaker": "Ada Lovelace"
}
```

IV.2 Exercício

Você descobriu que pré-preencher parte da resposta do assistente pode melhor direcionar as ações do modelo, impor formatos específicos e ajudar o modelo a manter consistência em cenários de role-play. Você deve criar um sistema que utiliza pré-preenchimento para obter informações estruturadas sobre filmes em formato JSON.

Instruções

1. Implementar uma função `get_movie_info` que usa `prefilling` para obter informações de filmes em formato JSON.
2. Utilizar o prompt fornecido, incorporando o título do filme.
3. Garantir que a resposta siga o formato JSON especificado.
4. Lidar com possíveis erros na geração do JSON.

Prompt pré-preenchido para ser utilizado

```
user_prompt = f"""
Provide information about the movie "{movie_title}" in JSON format.
Start your response with:
{{
  "title": "{movie_title}",
  ""
```

JSON esperado do modelo

```
{
  "title": "Movie Title",
  "year": 0000,
  "director": "Director's Name",
  "genre": ["Genre1", "Genre2"],
  "plot_summary": "Brief plot summary"
}
```

Como o seu programa será chamado:

```
movie_titles = ["The Matrix", "Inception", "Pulp Fiction", "The Shawshank Redemption", "The Godfather"]

for title in movie_titles:
    print(f"\nAnalyzing: {title}")
    result = get_movie_info(title)
    if result:
        for key, value in result.items():
            print(f"{key}: {value}")
    else:
        // Tratamento de erro adequado
    print("-" * 50)
```

Saída esperada:

```
Analyzing: The Matrix

title: The Matrix
year: 1999
director: The Wachowskis
genre: ['Science Fiction', 'Action']
plot_summary: A computer programmer discovers a hidden world where machines control humanity and joins a
               rebellion to overthrow them.
-----

Analyzing: Inception

Error: Failed to generate valid JSON
-----
```




Considere adicionar validações adicionais para garantir que todos os campos necessários estejam presentes no JSON retornado.



Não deixe a sua chave de API no código submetido. Durante a avaliação, você precisa configurá-la adequadamente.

Capítulo V

Ex02 – Encadeie prompts complexos para aumentar o desempenho

	Exercício : 02
Aprender a utilizar a técnica de encadeamento de prompts para lidar com tarefas complexas e multifacetadas, melhorando a precisão e a capacidade do modelo em realizar análises em múltiplas etapas	
Pasta de entrega : <i>ex02/</i>	
Arquivos para entregar : <code>shannon_analysis.py</code>	

A necessidade de dividir problemas complexos em etapas menores provavelmente já é familiar para você, afinal, isso também é útil para nós, humanos. No contexto dos LLMs, usamos frases como "Vamos pensar passo a passo" ou "Vamos analisar isso por partes".

Um problema bastante comum é o enigma sobre secar camisetas no sol¹. Alguns modelos presumem incorretamente que o número de camisetas afeta o tempo necessário para concluir a tarefa. A resposta correta é 4 ou "o mesmo", enquanto que, de acordo com outros modelos, a resposta pode ser 16.

Ao trabalhar com tarefas complexas, o modelo pode, às vezes, falhar se você tentar lidar com tudo em um único prompt. Usar *chain-of-thought*² (às vezes abreviado como *CoT*) pode ajudar a lidar com isso, mas e se sua tarefa ainda tiver várias etapas distintas que exigem um pensamento mais aprofundado?

É aí que entra o encadeamento de prompts (*prompt chaining*): dividir tarefas complexas em etapas menores, dividindo a complexidade.

¹"If we lay 5 shirts out in the sun and it takes 4 hours to dry, how long would 20 shirts take to dry? Answer immediately giving the number and nothing else."

²<https://www.ibm.com/topics/chain-of-thoughts>

Por que encadear prompts?

- **Precisão:** Cada etapa recebe toda a atenção do modelo, o que diminui as chances de erros.
- **Clareza:** Etapas mais simples resultam em instruções e saídas mais fáceis de entender.
- **Facilidade de correção:** Identifique e corrija problemas na sua cadeia de prompts com mais facilidade.

Quando encadear prompts

Use o encadeamento de prompts para tarefas que envolvem várias etapas, como síntese de pesquisa, análise de documentos ou criação de conteúdo de forma iterativa. Quando uma tarefa exige várias transformações, citações ou instruções, o encadeamento evita que o modelo perca ou interprete incorretamente as etapas.

Lembre-se: Cada etapa da sequência de ação vai receber a atenção total do modelo!



Quando o modelo falhar em completar uma etapa específica ou tiver um desempenho insatisfatório, você pode isolar essa etapa e refazer todas as etapas subsequentes.

Como encadear prompts

1. **Identifique as etapas:** Divida sua tarefa em etapas distintas e que sigam uma ordem lógica.
2. **Estruture para transportar informações:** Use separadores (como XML) para passar as saídas de um prompt para o próximo.
3. **Tenha um objetivo único para cada tarefa:** Cada etapa deve ter um objetivo claro e específico.
4. **Itere:** Aprimore as etapas com base no desempenho do modelo.

Exemplo de fluxos de trabalho encadeados

- Pipeline de criação de conteúdo: Pesquisa, esboço, rascunho, editar, formatar.
- Processamento de dados: Extrair, transformar, analisar, visualizar.

- Tomada de decisão: Coletar informações, listar opções, analisar cada uma, recomendar.
- Loops de verificação: Gerar conteúdo, revisar, refinar, revisar novamente.

V.1 Exemplo 1: Analisar e resumir informações

Sem encadeamento:

Quem: usuário

Conteúdo:

Analise a vida e obra de Gottfried Wilhelm Leibniz, extraia seus principais feitos e contribuições, e então forneça um resumo conciso em alemão:

Gottfried Wilhelm Leibniz (1646-1716) foi um polímata alemão reconhecido como um dos maiores pensadores dos séculos XVII e XVIII. Ele fez contribuições significativas em várias áreas, incluindo filosofia, matemática, lógica e ciências políticas. Leibniz é conhecido por desenvolver o cálculo infinitesimal independentemente de Newton, criar a notação moderna para o cálculo integral e diferencial, e inventar o sistema binário usado em computadores modernos. Na filosofia, ele propôs a teoria das mônadas e o conceito de "o melhor dos mundos possíveis". Leibniz também trabalhou em projetos diplomáticos e foi bibliotecário na corte de Hanôver.

Quem: modelo

Conteúdo:

Hier ist eine Analyse und Zusammenfassung über Gottfried Wilhelm Leibniz:

Hauptbeiträge:

1. Entwicklung der Infinitesimalrechnung unabhängig von Newton
2. Schaffung der modernen Notation für Integral- und Differentialrechnung
3. Erfindung des binären Systems, das in modernen Computern verwendet wird
4. Philosophische Theorie der Monaden
5. Konzept der "besten aller möglichen Welten"

Zusammenfassung:

Gottfried Wilhelm Leibniz (1646-1716) war ein deutscher Universalgelehrter und einer der bedeutendsten Denker des 17. und 18. Jahrhunderts. Er leistete bahnbrechende Beiträge in verschiedenen Bereichen, insbesondere in Mathematik und Philosophie. In der Mathematik entwickelte er die Infinitesimalrechnung und schuf wichtige Notationen. Seine Erfindung des Binärsystems legte den Grundstein für die moderne Computertechnologie. In der Philosophie prägte er mit seiner Monadenlehre und dem Konzept der "besten aller möglichen Welten" das Denken seiner Zeit. Leibniz' vielseitiges Wirken als Mathematiker, Philosoph, Logiker und Diplomat macht ihn zu einer Schlüsselfigur der europäischen Geistesgeschichte.

Com encadeamento:

Prompt 1: Extração de informações

Quem: usuário

Conteúdo:

Extraia os principais feitos e contribuições de Gottfried Wilhelm Leibniz:

Gottfried Wilhelm Leibniz (1646-1716) foi um polímata alemão reconhecido como um dos maiores pensadores dos séculos XVII e XVIII. Ele fez contribuições significativas em várias áreas, incluindo filosofia, matemática, lógica e ciências políticas. Leibniz é conhecido por desenvolver o cálculo infinitesimal independentemente de Newton, criar a notação moderna para o cálculo integral e diferencial, e inventar o sistema binário usado em computadores modernos. Na filosofia, ele propôs a teoria das mônadas e o conceito de "o melhor dos mundos possíveis". Leibniz também trabalhou em projetos diplomáticos e foi bibliotecário na corte de Hanôver.

Quem: modelo

Conteúdo:

Principais feitos e contribuições de Gottfried Wilhelm Leibniz:

1. Desenvolvimento do cálculo infinitesimal independentemente de Newton
2. Criação da notação moderna para o cálculo integral e diferencial
3. Invenção do sistema binário, base para computadores modernos
4. Proposição da teoria das mônadas na filosofia
5. Desenvolvimento do conceito de "o melhor dos mundos possíveis"
6. Contribuições significativas em filosofia, matemática, lógica e ciências políticas
7. Trabalho em projetos diplomáticos
8. Atuação como bibliotecário na corte de Hanôver

Prompt 2: Resumo em alemão

Quem: usuário

Conteúdo:

```
Usando os principais feitos e contribuições extraídos, escreva um resumo conciso sobre Gottfried Wilhelm
Leibniz em alemão:
<feitos_e_contribuicoes>
[Saída do Prompt 1]
</feitos_e_contribuicoes>
```

Quem: modelo

Conteúdo:

Zusammenfassung über Gottfried Wilhelm Leibniz:

Gottfried Wilhelm Leibniz (1646-1716) war ein herausragender deutscher Universalgelehrter des 17. und 18. Jahrhunderts. Seine bahnbrechenden Leistungen umfassten die unabhängige Entwicklung der Infinitesimalrechnung und die Schaffung moderner mathematischer Notationen. Leibniz' Erfindung des Binärsystems legte den Grundstein für die heutige Computertechnologie. In der Philosophie prägte er mit seiner Monadenlehre und dem Konzept der "besten aller möglichen Welten" das Denken seiner Zeit. Neben seinen wissenschaftlichen Beiträgen war Leibniz auch als Diplomat tätig und arbeitete als Bibliothekar am Hof von Hannover. Seine vielseitigen Beiträge in Mathematik, Philosophie, Logik und Politikwissenschaft machen ihn zu einer der einflussreichsten Persönlichkeiten seiner Epoche.

V.2 Exercício

Você descobriu que dividir tarefas complexas em etapas menores pode melhorar significativamente a performance do modelo. Esta técnica, conhecida como encadeamento de prompt, permite criar fluxos de trabalho sofisticados para análises multi-etapas e criação de conteúdo complexo. Você deve criar um sistema que utiliza prompt chaining para obter e analisar informações sobre Claude Elwood Shannon³, o pai da teoria da informação.

Instruções

1. Implementar completamente a função `run_prompt_chain` no arquivo `shannon_analysis.py`, incluindo a lógica para interagir com o modelo de linguagem de sua escolha.
2. A função `run_prompt_chain` deve incluir as seguintes etapas de análise:
 - Etapa 1: Obter uma visão geral da vida e carreira de Claude Shannon.

³https://en.wikipedia.org/wiki/Claude_Shannon

- Etapa 2: Analisar suas principais contribuições para a teoria da informação.
 - Etapa 3: Explorar o impacto de seu trabalho na computação moderna e nas tecnologias de comunicação.
 - Etapa 4: Sintetizar as informações das etapas anteriores em uma análise abrangente.
3. Utilizar tags XML para estruturar as respostas e passar informações entre as etapas.
 4. Garantir que cada etapa utilize informações das etapas anteriores para construir uma análise progressivamente mais profunda e coesa.

Como o seu programa será chamado:

```
if __name__ == "__main__":  
    run_prompt_chain()
```

Saída esperada

```
[resultado final do encadeamento, apresentando uma análise abrangente sobre Claude Shannon]
```

Função auxiliar fornecida

```
def extract_content(text, tag):  
    import re  
    pattern = f"<{tag}>(.*?)</{tag}>"  
    match = re.search(pattern, text, re.DOTALL)  
    return match.group(1).strip() if match else ""
```



Não deixe a sua chave de API no código submetido. Durante a avaliação, você precisa configurá-la adequadamente.

Capítulo VI

Entrega e Avaliação entre pares

VI.1 Processo de Entrega

- Submeta seu trabalho no repositório Git gerado na página principal do projeto.
- Certifique-se de que todos os arquivos necessários estejam incluídos e organizados conforme as instruções do projeto.
- Respeite o prazo de entrega estabelecido.

VI.2 Avaliação entre pares

- Seu projeto será avaliado por um dos seus colegas.
- A avaliação focará na qualidade do seu código e na aderência aos requisitos do projeto.
- Critérios de avaliação podem incluir:
 1. Funcionalidade: O código atende a todos os requisitos especificados?
 2. Legibilidade: O código é claro e bem estruturado?
 3. Eficiência: As soluções implementadas são otimizadas e seguem boas práticas?
 4. Organização: Os arquivos e estrutura do projeto estão bem organizados?
- Feedback detalhado é esperado, mas pode variar em extensão e detalhamento.

VI.3 Dicas para uma avaliação bem-sucedida

- Revise seu código antes da submissão final.

- Teste exaustivamente todas as funcionalidades implementadas.
- Se entender necessário para clareza, documente claramente qualquer decisão ou suposição feita.
- Esteja preparado para explicar suas escolhas de implementação.



A avaliação entre pares é uma oportunidade para aprendizado e crescimento pessoal e profissional. Esteja aberto ao feedback recebido e use-o para aprimorar suas habilidades.