FACULDADE INFNET
CURSO ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CEDRICK FELICIO

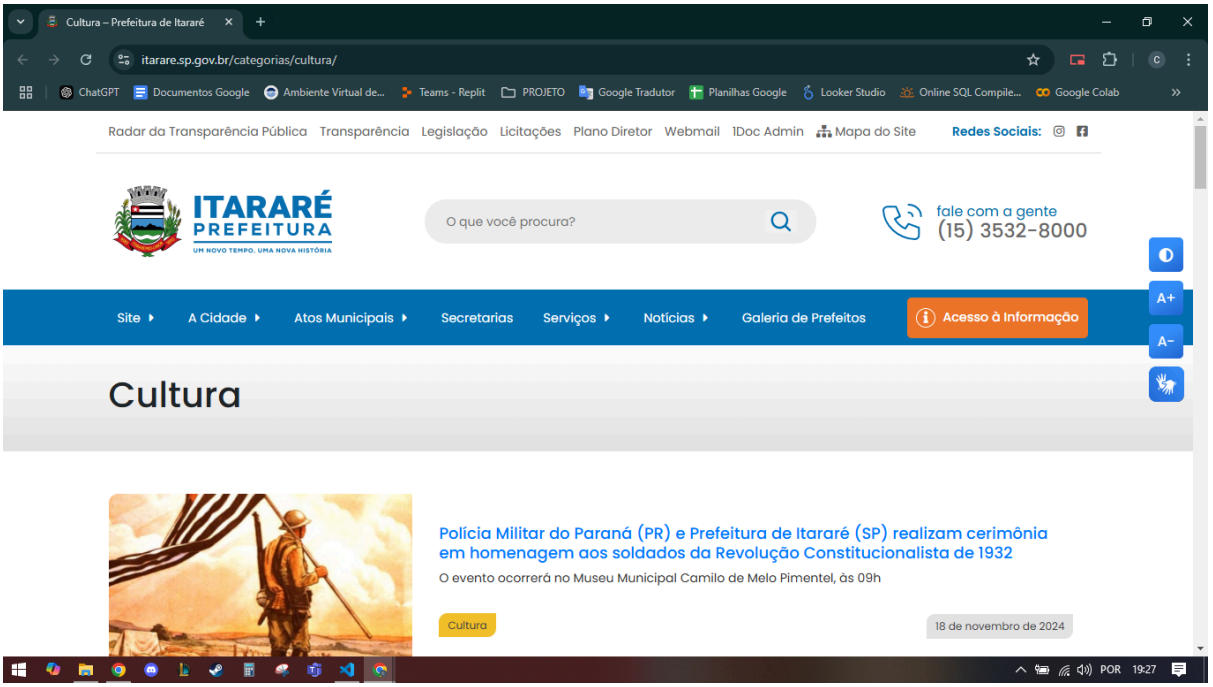
TESTE DE PERFORMANCE - TP5

Projeto de Bloco: Fundamentos de Dados

SÃO PAULO
2024

Site utilizado

Este é o site da cidade de itararé, na qual possui os eventos relacionados à cidade. O TP todo elaborado baseado nas informações contidas nele. Link para o site: https://itarare.sp.gov.br/categorias/cultura.



Estrutura HTML analisada

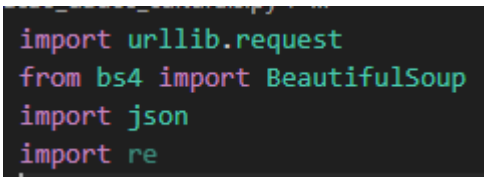
Foi a partir desta estrutura que os dados do site foram extraídos. Os principais elementos utilizados para realizar a busca pelos eventos nas páginas do site foram:

- <article>: Identifica o bloco principal que agrupa os dados de um evento.
- <a> (com href e title): Extrai o link do evento e seu título.
- <p>: Extrai a descrição do evento.
- <time>: Extrai a data do evento.
- <ul class="list-categories">: Extrai a categoria ou tipo do evento.

Arquivo extracao_dados_culturais.py

Este arquivo tem como objetivo extrair dados específicos do site https://itarare.sp.gov.br/categorias/cultura, utilizando bibliotecas como urllib e BeautifulSoup. O processo deste arquivo consiste em extrair os dados do site e inseri-los em um arquivo JSON estruturado. O objetivo de gerar o arquivo JSON é organizar os dados e realizar um futuro tratamento em seus registros, para que assim, os registros possam ser inseridos em um banco de dados SQLite.

Bibliotecas e módulos utilizados



urllib.request

A biblioteca urllib.request é utilizada porque precisamos acessar páginas da web e obter o conteúdo HTML delas. No caso do código, ela é essencial para fazer a requisição HTTP ao site de onde os dados serão coletados. O método urlopen faz exatamente isso: ele envia uma solicitação à URL fornecida e retorna a resposta com o conteúdo da página.

Sem essa biblioteca, não teríamos como acessar o HTML que contém as informações dos eventos. Além disso, ela permite manipular a resposta para decodificá-la em texto legível (no caso, UTF-8), o que é necessário antes de processar os dados.

BeautifulSoup

O HTML retornado pelo urllib.request é apenas um texto bruto. Para extrair informações úteis, como o título, a descrição e a data de eventos específicos, precisamos de uma ferramenta que organize e processe esse conteúdo de maneira estruturada. É aí que entra o BeautifulSoup.

Ele transforma o texto HTML em uma estrutura de dados navegável, como uma árvore de objetos, permitindo localizar e extrair elementos específicos (tags, classes, IDs) com métodos simples como .find_all() e .find(). No código, isso é usado para localizar os elementos <article> que contêm informações sobre eventos e navegar dentro desses elementos para capturar dados relevantes como título, link, descrição e data. Sem o BeautifulSoup, seria muito mais trabalhoso processar o HTML manualmente, pois teríamos que implementar um analisador HTML personalizado.

json

Depois de extrair os dados dos eventos, precisamos armazená-los em um formato estruturado e reutilizável.

No código, o json é usado para converter os dados extraídos (armazenados como listas e dicionários Python) em um arquivo JSON formatado. Isso facilita o compartilhamento dos dados ou sua reutilização em outras aplicações. Além disso, o JSON preserva a estrutura hierárquica dos dados (como IDs, nomes, descrições, links, etc.), o que torna os dados mais organizados e fáceis de manipular futuramente.

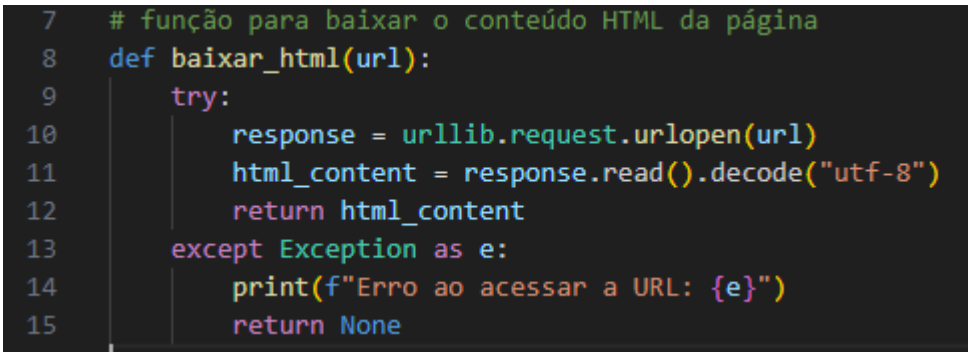
re

Embora o re não tenha sido usado diretamente no código fornecido, ele é frequentemente incluído em scripts de web scraping porque permite encontrar ou validar padrões específicos em texto, algo que BeautifulSoup sozinho não faz. Por exemplo, se precisarmos extrair datas que estão embutidas em um texto não estruturado ou validar links, o re seria a ferramenta ideal.

No caso deste script, se os dados não estivessem em tags HTML específicas ou se fossem encontrados como texto bruto, poderíamos usar re para localizá-los com expressões regulares.

Funções

baixar_html



```python
# função para baixar o conteúdo HTML da página
def baixar_html(url):
    try:
        response = urllib.request.urlopen(url)
        html_content = response.read().decode("utf-8")
        return html_content
    except Exception as e:
        print(f"Erro ao acessar a URL: {e}")
        return None
```

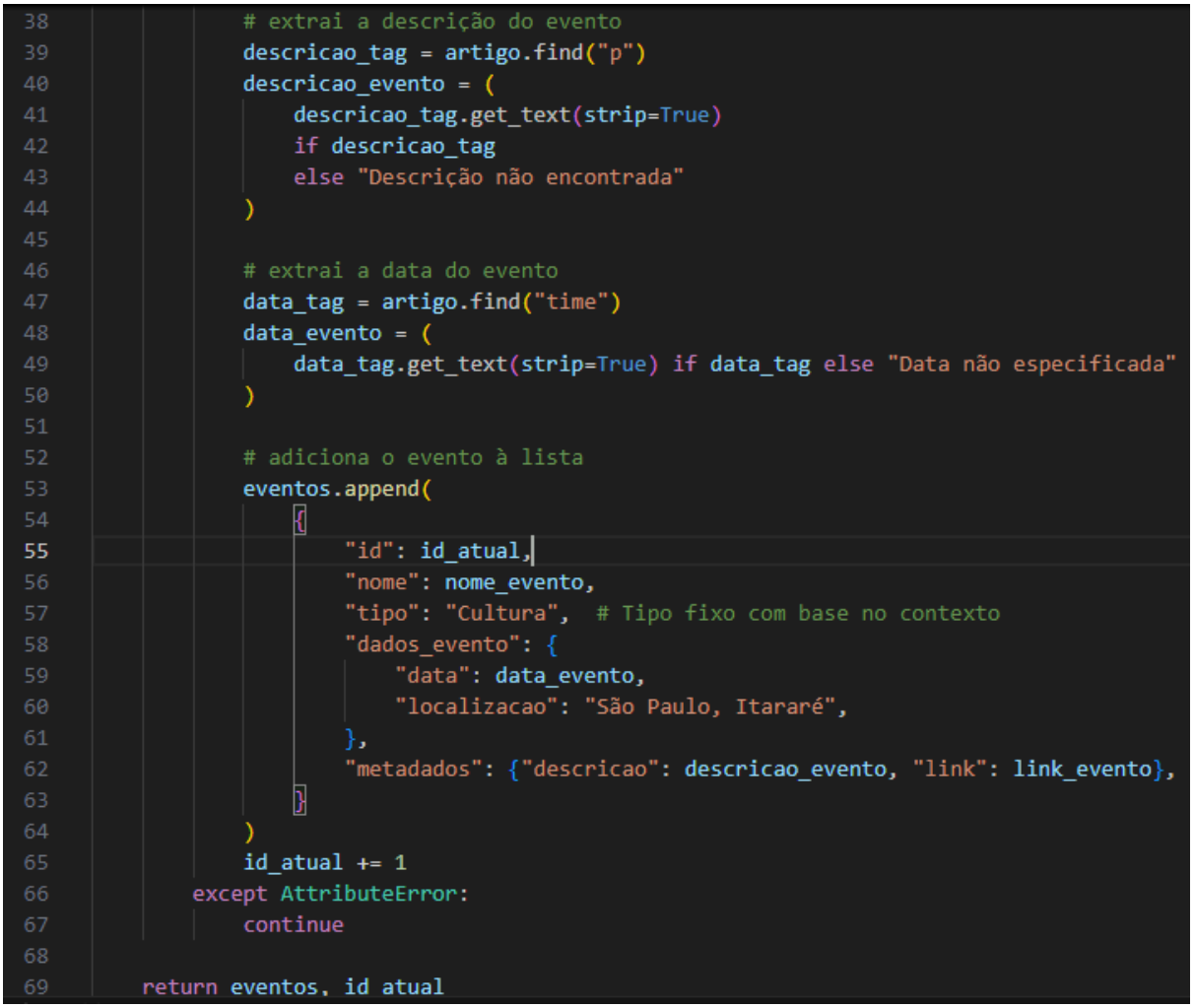
A função baixar_html tem como objetivo acessar uma URL fornecida e retornar o conteúdo HTML dessa página em formato de texto legível (UTF-8). Para isso, ela utiliza a biblioteca urllib.request para fazer a requisição HTTP e obter a resposta da página. Se a operação for bem-sucedida, ela retorna o HTML da página; caso ocorra algum erro (como problemas de conexão ou URL inválida), a função captura a exceção, exibe uma mensagem de erro explicativa e retorna None, garantindo que o programa não seja interrompido inesperadamente.

extrair_dados

Para melhor explicação, vamos explicar esta função por partes.

```python
# função para extrair informações específicas de eventos culturais no site
def extrair_dados(html_content, id_inicial):
    soup = BeautifulSoup(html_content, "html.parser")
    eventos = []
    id_atual = id_inicial

    # procura todos os artigos na página
    artigos = soup.find_all("article", class_="row d-flex align-items-center")
    for artigo in artigos:
        try:
            # extrai o título do evento
            titulo_tag = artigo.find("h6")
            link_tag = titulo_tag.find("a") if titulo_tag else None
            nome_evento = (
                link_tag.get("title", "Título não encontrado")
                if link_tag
                else "Título não encontrado"
            )
            link_evento = link_tag["href"] if link_tag else "Link não encontrado"
```

Na primeira parte da função extrair_dados, o código utiliza o BeautifulSoup para analisar o HTML recebido e transformá-lo em uma estrutura navegável. Ele começa inicializando uma lista chamada eventos para armazenar os dados extraídos e define o identificador inicial com id_inicial. Em seguida, busca todos os elementos <article> da página que possuem a classe específica "row d-flex align-items-center", que aparentemente contém informações dos eventos. Dentro de um loop, para cada artigo encontrado, o código tenta localizar o título do evento em uma tag <h6> e, se disponível, o link associado dentro de uma tag <a>. Ele extrai o texto do título e o link, ou fornece valores padrão caso essas informações não sejam encontradas.



```
            # extrai a descrição do evento
            descricao_tag = artigo.find("p")
            descricao_evento = (
                descricao_tag.get_text(strip=True)
                if descricao_tag
                else "Descrição não encontrada"
            )

            # extrai a data do evento
            data_tag = artigo.find("time")
            data_evento = (
                data_tag.get_text(strip=True) if data_tag else "Data não especificada"
            )

            # adiciona o evento à lista
            eventos.append(
                {
                    "id": id_atual,
                    "nome": nome_evento,
                    "tipo": "Cultura",  # Tipo fixo com base no contexto
                    "dados_evento": {
                        "data": data_evento,
                        "localizacao": "São Paulo, Itararé",
                    },
                    "metadados": {"descricao": descricao_evento, "link": link_evento},
                }
            )
            id_atual += 1
        except AttributeError:
            continue

    return eventos, id_atual
```

Na segunda parte da função extrair_dados, o código continua extraindo informações detalhadas sobre cada evento. Ele busca a descrição do evento em uma tag <p> e a data em uma tag <time>, garantindo valores padrão ("Descrição não encontrada" e "Data não especificada") caso esses elementos não estejam presentes. Depois, organiza todas as informações coletadas (id, nome, tipo, data, localização, descrição e link) em um dicionário estruturado que representa o evento e adiciona esse dicionário à lista eventos. Por fim, incrementa o id_atual para garantir que cada evento tenha um identificador único e retorna a lista de eventos junto com o próximo ID disponível ao final da função.

salvar_json

```python
# função para salvar os dados no formato JSON
def salvar_json(eventos, nome_arquivo):
    try:
        data = {"eventos": eventos}
        with open(nome_arquivo, "w", encoding="utf-8") as f:
            json.dump(data, f, indent=4, ensure_ascii=False)
        print(f"Dados salvos no arquivo JSON: {nome_arquivo}")
    except Exception as e:
        print(f"Erro ao salvar JSON: {e}")
```

A função salvar_json é responsável por armazenar os dados extraídos em um arquivo no formato JSON. Ela recebe a lista de eventos e o nome do arquivo onde os dados serão salvos. Primeiro, organiza os eventos em um dicionário com a chave "eventos" e, em seguida, escreve esse dicionário no arquivo especificado usando o método json.dump, garantindo que o conteúdo seja formatado com indentação e que caracteres especiais sejam preservados com ensure_ascii=False. Caso o processo ocorra com sucesso, exibe uma mensagem confirmando a criação do arquivo; se ocorrer algum erro durante a operação, captura a exceção e exibe uma mensagem de erro explicativa, evitando que o programa seja interrompido.

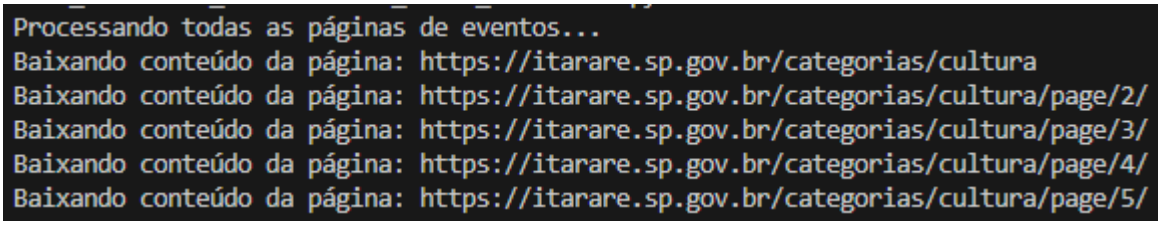
processar_todas_as_paginas

```python
# função principal para processar todas as páginas e extrair os eventos
def processar_todas_as_paginas(base_url):
    print("Processando todas as páginas de eventos...")
    eventos_totais = []
    id_atual = 1
    pagina_atual = 1

    # loop para percorrer todas as páginas que contém os eventos
    while True:
        url = f"{base_url}/page/{pagina_atual}/" if pagina_atual > 1 else base_url
        print(f"Baixando conteúdo da página: {url}")
        html_content = baixar_html(url)

        if not html_content:
            print("Falha ao obter o conteúdo HTML. Encerrando coleta.")
            break

        eventos, id_atual = extrair_dados(html_content, id_atual)
        if not eventos:
            print(
                f"Sem eventos encontrados na página {pagina_atual}. Parando a coleta."
            )
            break

        eventos_totais.extend(eventos)
        pagina_atual += 1

    if eventos_totais:
        print("Salvando dados no formato JSON...")
        salvar_json(eventos_totais, "dados_culturais_html_todas_paginas.json")
    else:
        print("Nenhum dado foi extraído de nenhuma página.")
```

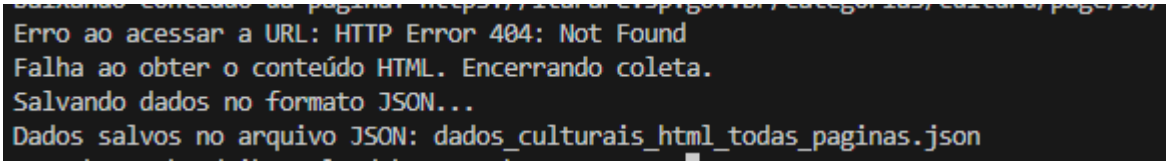
A função processar_todas_as_paginas é responsável por navegar automaticamente pelas páginas de eventos de um site, coletar os dados e salvá-los em um arquivo JSON. Ela inicia com a página base (definida por base_url) e usa um loop para acessar cada página subsequente. Para cada página, a função tenta baixar o HTML com baixar_html e extrair os dados dos eventos usando extrair_dados. Se o HTML não puder ser acessado ou nenhuma informação for encontrada, o processo é interrompido. Caso contrário, os eventos coletados são acumulados em uma lista chamada eventos_totais. Quando todas as páginas são processadas, a lista completa é salva em um arquivo JSON com salvar_json. A função também exibe mensagens para informar o progresso ou qualquer falha durante a execução, garantindo que o processo seja compreensível e rastreável.

Resultado da execução do arquivo

Processo de extração dos dados

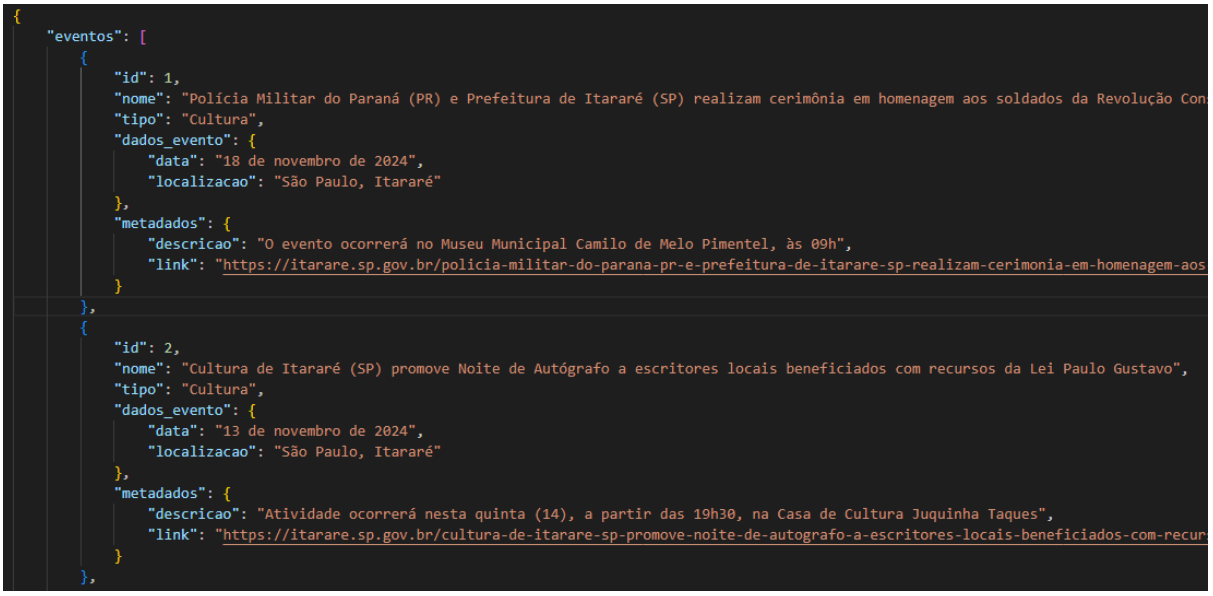


Ao executar o arquivo, são apresentados no terminal estes logs. Eles representam a extração do conteúdo de cada página referente aos eventos.



Após a extração dos dados em todas as páginas, são apresentados logs de erro, pois não são mais encontradas páginas referentes aos eventos. Além disso, é sinalizada o encerramento e que os dados foram salvos no arquivo JSON.

Arquivo JSON gerado



Este é a estrutura do arquivo dados_culturais_html_todas_paginas.json que contém os dados extraidos do site. Este arquivo JSON é gerado na execução do script do arquivo extracao_dados_culturais.py.

Arquivo tratamento_datas_json.py

Este arquivo tem como objetivo realizar o tratamento das datas extraídas do site para o formato ISO, pois por conta do formato original extraído, algumas consultas não são possíveis serem realizadas. Ao final do tratamento, é gerado um novo arquivo JSON chamado dados_culturais_html_atualizado.json. Este arquivo, portanto, está pronto para ser inserido no banco de dados.

Bibliotecas e módulos utilizados

```
1   import json
2   from datetime import datetime
```

json

A biblioteca json neste arquivo é usada para trabalhar com arquivos no formato JSON, que contém os dados dos eventos extraídos anteriormente. Ela é essencial para:

- Ler o arquivo JSON de entrada (input_file): A função json.load carrega o conteúdo do arquivo em um dicionário Python, permitindo a manipulação dos dados.
- Modificar os dados: No caso, atualizar as datas no formato desejado.
- Salvar o arquivo atualizado (output_file): A função json.dump converte o dicionário Python de volta para o formato JSON e o escreve em um novo arquivo.

Sem o json, seria muito mais complexo lidar com a leitura, manipulação e gravação de arquivos JSON, que são o formato ideal para armazenar dados estruturados como esses eventos.

datatime

A biblioteca datetime é utilizada para manipulação de datas e é fundamental para a padronização do formato das datas dos eventos. Embora neste caso específico o código não utilize diretamente as funções do datetime para conversão, ela é carregada para garantir manipulação avançada de datas, se necessário. Por exemplo:

- Converter datas para o formato ISO: O formato "YYYY-MM-DD" é o padrão ISO para datas e é amplamente utilizado em sistemas e bancos de dados.
- Validação de datas: Embora no código atual a validação seja feita manualmente usando um dicionário de meses, a biblioteca datetime poderia validar automaticamente se uma data é válida, evitando erros.

O uso do datetime aqui reflete a necessidade de trabalhar com datas de forma segura e padronizada, mesmo que a conversão seja implementada de forma manual.

Funções

converter_data

```python
# converte as datas extraídas para o formato ISO
def converter_data(data):
    try:
        meses = {
            "janeiro": "01", "fevereiro": "02", "março": "03", "abril": "04", "maio": "05", "junho": "06",
            "julho": "07", "agosto": "08", "setembro": "09", "outubro": "10", "novembro": "11", "dezembro": "12"
        }
        partes = data.lower().split(" de ")
        dia = partes[0]
        mes = meses[partes[1]]
        ano = partes[2]
        return f"{ano}-{mes}-{int(dia):02d}"
    except Exception as e:
        print(f"Erro ao converter data: {data}. Detalhes: {e}")
        return data  # retorna a data original em caso de falha
```

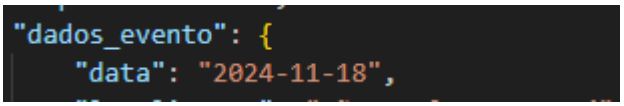
A função converter_data recebe uma data em formato textual, como "12 de junho de 2024", e a converte para o formato padrão ISO, "2024-06-12". Para isso, ela utiliza um dicionário que mapeia os meses do ano em português para seus respectivos números. A função divide a data original em dia, mês e ano usando o texto " de " como separador, e substitui o mês textual pelo seu número correspondente. Se a conversão for bem-sucedida, retorna a data formatada no padrão "YYYY-MM-DD". Caso ocorra algum erro, como uma data mal formatada, ela exibe uma mensagem explicativa e retorna a data original para evitar falhas no restante do código.

atualizar_dados_json

```python
# atualiza as datas no arquivo json e salva em um novo arquivo
def atualizar_dados_json(input_file, output_file):
    try:
        with open(input_file, 'r', encoding='utf-8') as arquivo:
            dados = json.load(arquivo)

        for evento in dados.get("eventos", []):
            if "dados_evento" in evento and "data" in evento["dados_evento"]:
                evento["dados_evento"]["data"] = converter_data(evento["dados_evento"]["data"])

        with open(output_file, 'w', encoding='utf-8') as arquivo_atualizado:
            json.dump(dados, arquivo_atualizado, ensure_ascii=False, indent=4)

        print(f"Conversão concluída. Arquivo salvo como '{output_file}'.")
    except Exception as e:
        print(f"Erro ao processar o arquivo: {e}")
```

A função atualizar_dados_json é responsável por abrir um arquivo JSON contendo informações de eventos, atualizar as datas para o formato ISO (YYYY-MM-DD) e salvar o resultado em um novo arquivo. Primeiro, ela carrega os dados do arquivo de entrada (input_file) e percorre cada evento na lista de eventos. Para os eventos que possuem uma data registrada dentro da chave "dados_evento", a função utiliza converter_data para transformar a data no formato desejado. Após atualizar todos os eventos, ela salva os dados modificados no arquivo de saída (output_file) com uma formatação legível e preservando caracteres especiais. Caso ocorra algum erro durante o processo, uma mensagem explicativa é exibida, garantindo que o código não falhe silenciosamente.

Resultado da execução do arquivo



O resultado da execução deste arquivo, é o tratamento da data original extraída do site, para uma data no formato ISO (YYYY-MM-DD). Este tratamento é realizado e gerado um novo arquivo JSON chamado dados_culturais_html_atualizado.json contendo os mesmos dados extraídos do site, porém, com essa diferença da formatação da data.

Arquivo insercao_sql.py

O objetivo deste arquivo é de criar um banco de dados estruturado para receber os dados do arquivo JSON chamado dados_culturais_html_atualizado.json, que já estão com o tratamento das datas realizadas.

Bibliotecas e módulos utilizados

```
1   import sqlite3
2   import json
```

sqlite3

A biblioteca sqlite3 é usada para criar e gerenciar o banco de dados SQLite, que armazena os dados extraídos de forma organizada e relacional. No código, ela permite:

1. **Criar o banco de dados e tabelas:**
   As tabelas eventos, dados_eventos e metadados são configuradas para armazenar diferentes aspectos dos dados de cada evento, como nome, tipo, data, localização e metadados.
2. **Inserir dados no banco:**
   A biblioteca fornece comandos SQL para adicionar registros, garantindo a consistência e a integridade dos dados com mecanismos como chaves primárias.
3. **Manter persistência e escalabilidade:**
   Ao salvar os dados no banco SQLite, eles podem ser consultados ou manipulados posteriormente de forma eficiente, com suporte a consultas SQL complexas.
4. **Evitar duplicação e conflitos:**
   Com os índices primários e tratamento de exceções como sqlite3.IntegrityError, o código evita problemas ao inserir dados duplicados.

json

A biblioteca json é utilizada para carregar os dados estruturados no arquivo JSON gerado anteriormente. No código, ela:

1. **Lê os dados do arquivo JSON:**

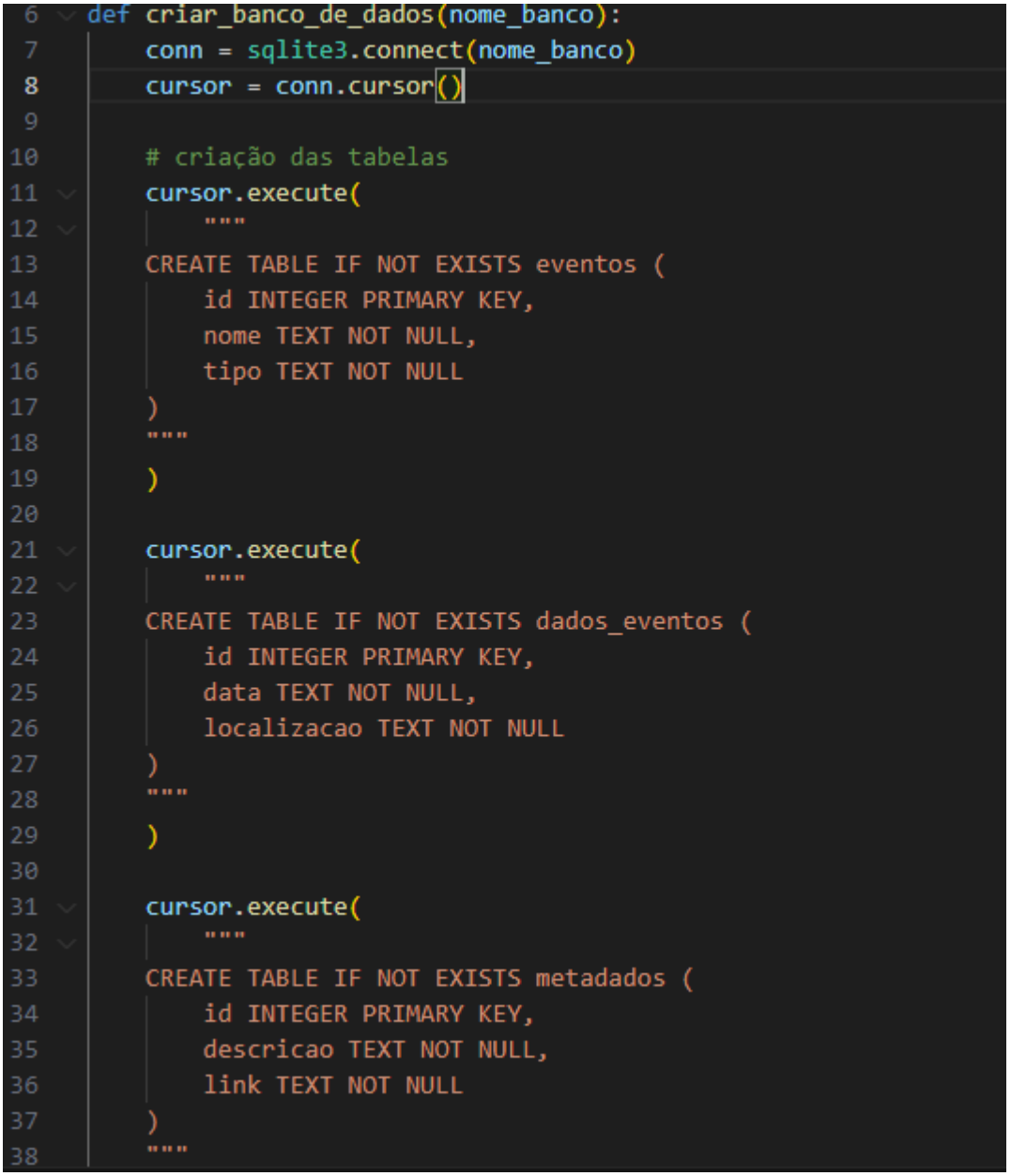
   Com a função json.load, os dados são carregados e transformados em um formato que o Python compreende, como listas e dicionários.
2. **Facilita a transferência de dados:**

   O arquivo JSON serve como um meio de transporte entre a etapa de extração (web scraping) e a etapa de armazenamento no banco de dados.
3. **Garante compatibilidade:**

   O formato JSON é amplamente utilizado e padronizado, o que permite que os dados extraídos possam ser facilmente utilizados por outras aplicações ou processos no futuro.

Funções

criar_banco_de_dados



```
def criar_banco_de_dados(nome_banco):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()

    # criação das tabelas
    cursor.execute(
        """
    CREATE TABLE IF NOT EXISTS eventos (
        id INTEGER PRIMARY KEY,
        nome TEXT NOT NULL,
        tipo TEXT NOT NULL
    )
    """
    )

    cursor.execute(
        """
    CREATE TABLE IF NOT EXISTS dados_eventos (
        id INTEGER PRIMARY KEY,
        data TEXT NOT NULL,
        localizacao TEXT NOT NULL
    )
    """
    )

    cursor.execute(
        """
    CREATE TABLE IF NOT EXISTS metadados (
        id INTEGER PRIMARY KEY,
        descricao TEXT NOT NULL,
        link TEXT NOT NULL
    )
    """
```

Na primeira parte da função criar_banco_de_dados, o código conecta-se a um banco de dados SQLite usando o nome fornecido em nome_banco e cria um cursor para executar comandos SQL. Em seguida, ele cria três tabelas principais no banco de dados: eventos, dados_eventos e metadados. Cada tabela é configurada com uma estrutura específica para armazenar diferentes aspectos dos eventos. A tabela eventos armazena o ID, nome e tipo do evento, enquanto dados_eventos guarda informações como data e localização, e a tabela metadados armazena a descrição e

o link. As tabelas são criadas apenas se ainda não existirem, garantindo que o banco possa ser reutilizado sem sobrescrever os dados.

```
    conn.commit()
    conn.close()
    print(
        f"Banco de dados '{nome_banco}' criado com sucesso, com tabelas configuradas."
    )
```

Na segunda parte da função criar_banco_de_dados, o código garante que todas as alterações feitas no banco de dados, como a criação das tabelas, sejam salvas de forma permanente usando o método conn.commit(). Após isso, a conexão com o banco de dados é encerrada com conn.close(), liberando os recursos utilizados. Por fim, uma mensagem é exibida no console para confirmar que o banco de dados foi criado com sucesso e que as tabelas estão devidamente configuradas, dando um feedback claro sobre o resultado da execução.

carregar_dados_json

```
# função para carregar os dados do arquivo JSON
def carregar_dados_json(nome_arquivo):
    try:
        with open(nome_arquivo, "r", encoding="utf-8") as f:
            dados = json.load(f)
        return dados["eventos"]
    except Exception as e:
        print(f"Erro ao carregar o arquivo JSON: {e}")
        return []
```

A função carregar_dados_json é responsável por abrir e carregar os dados de um arquivo JSON especificado pelo parâmetro nome_arquivo. Ela utiliza a função json.load para transformar o conteúdo do arquivo em um objeto Python, como um dicionário ou uma lista, e retorna a lista de eventos encontrada na chave "eventos". Caso ocorra algum erro durante a leitura ou o carregamento do arquivo, como um arquivo inválido ou inexistente, a função captura a exceção, exibe uma mensagem de erro explicativa e retorna uma lista vazia, garantindo que o programa possa continuar executando sem falhar.

inserir_dados_no_banco

```python
# função para inserir os dados nas tabelas
def inserir_dados_no_banco(nome_banco, eventos):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()
```

Essa primeira parte da função inserir_dados_no_banco, é responsável por realizar a conexão com o banco de dados e criar um cursor para realizar a inserção dos dados.

```python
    for evento in eventos:
        try:
            # insere os dados na tabela eventos
            cursor.execute(
                """
            INSERT INTO eventos (id, nome, tipo)
            VALUES (?, ?, ?)
            """,
                (evento["id"], evento["nome"], evento["tipo"]),
            )
```

Para cada tabela, ele executa a tentativa de inserir os dados na tabela especificada utilizando o cursor.execute.

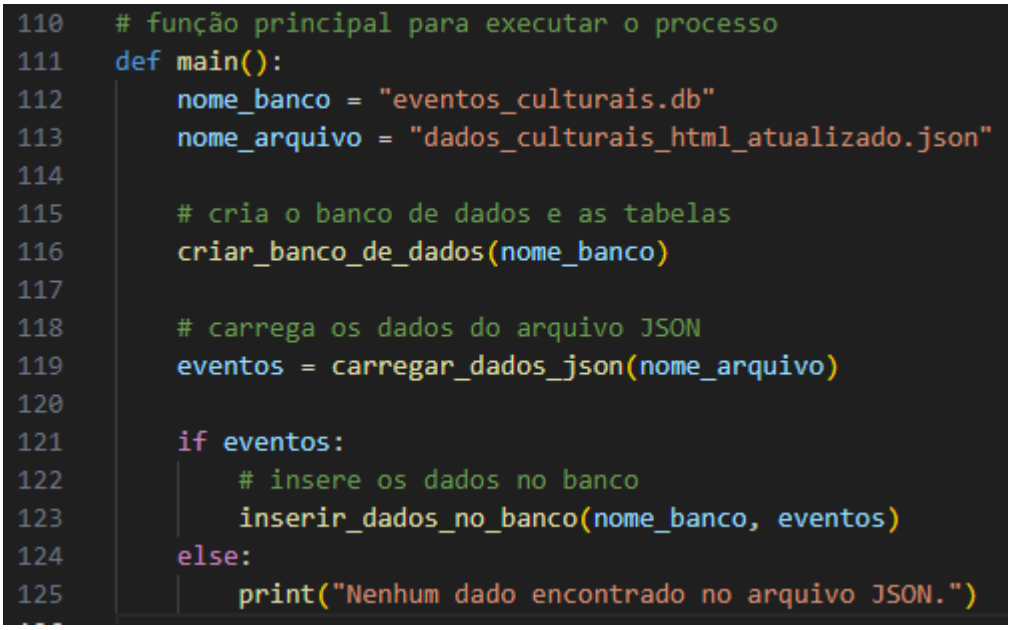
```
            # insere os dados na tabela dados_eventos
            cursor.execute(
                """
            INSERT INTO dados_eventos (id, data, localizacao)
            VALUES (?, ?, ?)
            """,
                (
                    evento["id"],
                    evento["dados_evento"]["data"],
                    evento["dados_evento"]["localizacao"],
                ),
            )

            # insere os dados na tabela metadados
            cursor.execute(
                """
            INSERT INTO metadados (id, descricao, link)
            VALUES (?, ?, ?)
            """,
                (
                    evento["id"],
                    evento["metadados"]["descricao"],
                    evento["metadados"]["link"],
                ),
            )
```

Essa parte do código insere os dados específicos de cada evento nas tabelas dados_eventos e metadados do banco de dados. Na tabela dados_eventos, são armazenadas informações como o ID do evento, a data (extraída do campo "dados_evento") e a localização. Já na tabela metadados, são inseridos o ID do evento, a descrição e o link (extraídos do campo "metadados"). Para isso, o código usa comandos SQL INSERT INTO com placeholders (?) para garantir segurança e evitar ataques de injeção de SQL. Cada conjunto de informações do evento é mapeado corretamente para as colunas correspondentes de cada tabela, permitindo que os dados sejam armazenados de maneira organizada e relacional.

```
        except sqlite3.IntegrityError as e:
            print(f"Erro ao inserir dados: {e}")
            continue

    conn.commit()
    conn.close()
    print(f"Dados inseridos com sucesso no banco de dados '{nome_banco}'.")
```

Por fim, essa parte do código lida com possíveis erros de integridade ao inserir os dados no banco de dados, como tentativas de adicionar registros com IDs duplicados. Se um erro desse tipo ocorrer, ele é capturado pelo bloco except sqlite3.IntegrityError, que exibe uma mensagem no console explicando o problema e usa continue para ignorar o registro problemático sem interromper a inserção dos demais dados. Após processar todos os eventos, o código chama conn.commit() para salvar as alterações realizadas no banco e, em seguida, fecha a conexão com conn.close() para liberar os recursos. Por fim, exibe uma mensagem confirmando que os dados foram inseridos com sucesso no banco, fornecendo um feedback claro sobre a conclusão do processo.

main



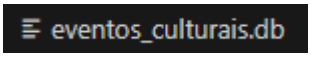
```
110  # função principal para executar o processo
111  def main():
112      nome_banco = "eventos_culturais.db"
113      nome_arquivo = "dados_culturais_html_atualizado.json"
114
115      # cria o banco de dados e as tabelas
116      criar_banco_de_dados(nome_banco)
117
118      # carrega os dados do arquivo JSON
119      eventos = carregar_dados_json(nome_arquivo)
120
121      if eventos:
122          # insere os dados no banco
123          inserir_dados_no_banco(nome_banco, eventos)
124      else:
125          print("Nenhum dado encontrado no arquivo JSON.")
```

A função main organiza todo o fluxo principal do código, desde a configuração do banco de dados até o carregamento e inserção dos dados. Primeiro, ela define o nome do banco de dados e do arquivo JSON que contém os dados dos eventos. Em seguida, chama a função criar_banco_de_dados para criar o banco e suas tabelas, garantindo que o ambiente esteja configurado. Depois disso, utiliza a função carregar_dados_json para carregar os eventos do arquivo JSON especificado. Se houver eventos carregados, a função inserir_dados_no_banco é chamada para inserir essas informações no banco. Caso contrário, exibe uma mensagem informando que nenhum dado foi encontrado. Essa função centraliza o processo completo e garante que as etapas sejam executadas na ordem correta.

Resultado da execução do arquivo

```
Banco de dados 'eventos_culturais.db' criado com sucesso, com tabelas configuradas.
Dados inseridos com sucesso no banco de dados 'eventos_culturais.db'.
```

Estes são os logs apresentados ao finalizar a execução do arquivo. Eles sinalizam a criação do banco de dados e a inserção dos dados. Banco de dados gerado:



Arquivo sistema_consultas.py

Este arquivo executa as consultas solicitadas realizando a leitura direta do banco de dados eventos_culturais.db. Para facilitar a execução das consultas, neste arquivo possui um sistema interativo que possibilita a escolha da consulta desejada. Além disso, caso seja preferível, há uma opção para extrair as consultas em arquivos csv.

Bibliotecas e módulos utilizados

```python
import sqlite3
from datetime import datetime
import csv
```

sqlite3

A biblioteca sqlite3 é utilizada para conectar, consultar e manipular o banco de dados SQLite que armazena as informações dos eventos. No código ela é utilizada para:

1. **Consultas SQL:** Permite realizar consultas para buscar eventos com critérios específicos, como localização, data ou palavras-chave.
2. **Joins entre tabelas:** Facilita a recuperação de dados relacionados entre as tabelas eventos, dados_eventos e metadados, organizando as informações em resultados completos.
3. **Integração com Python:** A biblioteca sqlite3 é integrada ao Python, tornando-a ideal para operações de bancos de dados locais sem necessidade de configurar servidores adicionais.

datatime

A biblioteca datetime é usada para manipulação de datas e cálculos relacionados no código:

1. **Comparação de datas:** Em consultar_eventos_proximos, uma data fixa (2024-01-01) é usada como referência para buscar os eventos mais próximos. A manipulação de datas seria essencial para garantir cálculos precisos caso houvesse mais lógica relacionada a intervalos ou formatos dinâmicos de data.
2. **Padronização de formato ISO:** A biblioteca é relevante para trabalhar com o formato ISO das datas (YYYY-MM-DD), o que facilita consultas e ordenação no banco.

csv

A biblioteca csv é utilizada para exportar os resultados das consultas para arquivos no formato CSV. No código a função salvar_csv grava os resultados das consultas em arquivos organizados com cabeçalhos específicos utilizando a biblioteca csv

Funções

consulta_todos_os_eventos

```
5   # função para mostrar todos os eventos com suas datas, localização e tipo de evento
6   def consultar_todos_eventos(nome_banco):
7       conn = sqlite3.connect(nome_banco)
8       cursor = conn.cursor()
9
10      query = """
11      SELECT eventos.nome, dados_eventos.data, dados_eventos.localizacao, eventos.tipo
12      FROM eventos
13      JOIN dados_eventos ON eventos.id = dados_eventos.id
14      """
15
16      cursor.execute(query)
17      resultados = cursor.fetchall()
18      conn.close()
19      return resultados
```

A função consultar_todos_eventos realiza a consulta de todos os eventos registrados no banco de dados, retornando informações completas como nome,

data, localização e tipo de cada evento. Para isso, ela se conecta ao banco de dados especificado por nome_banco e utiliza um comando SQL que combina as tabelas eventos e dados_eventos através de um JOIN, vinculando os dados de ambas as tabelas com base no ID. Após executar a consulta, ela armazena os resultados em uma lista de tuplas usando cursor.fetchall(), fecha a conexão com o banco e retorna os dados. Essa função é útil para exibir uma visão geral de todos os eventos e suas principais características.

consultar_eventos_proximos

```python
# função para mostrar os 2 eventos mais próximos de iniciar (Considera data atual 01/01/2024)
def consultar_eventos_proximos(nome_banco):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()

    data_atual = "2024-01-01"

    query = """
    SELECT eventos.nome, dados_eventos.data, dados_eventos.localizacao
    FROM eventos
    JOIN dados_eventos ON eventos.id = dados_eventos.id
    WHERE date(dados_eventos.data) >= ?
    ORDER BY date(dados_eventos.data) ASC
    LIMIT 2
    """

    cursor.execute(query, (data_atual,))
    resultados = cursor.fetchall()
    conn.close()
    return resultados
```

A função consultar_eventos_proximos busca os dois eventos mais próximos de acontecer a partir de uma data fixa, definida como "2024-01-01". Ela conecta-se ao banco de dados especificado em nome_banco e utiliza um comando SQL que seleciona o nome, a data e a localização dos eventos, combinando as tabelas eventos e dados_eventos com um JOIN. A consulta filtra apenas os eventos cuja data seja igual ou posterior à data atual e ordena os resultados em ordem crescente pela data do evento. O limite de 2 resultados garante que apenas os dois eventos mais próximos sejam retornados. Após a execução da consulta, os resultados são armazenados, a conexão com o banco é encerrada e a lista de eventos é retornada.

consultar_eventos_em_itarare

```python
# função para mostrar os eventos que acontecem na localização São Paulo, Itararé
def consultar_eventos_em_itarare(nome_banco):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()

    query = """
    SELECT eventos.nome, dados_eventos.data, dados_eventos.localizacao
    FROM eventos
    JOIN dados_eventos ON eventos.id = dados_eventos.id
    WHERE dados_eventos.localizacao = "São Paulo, Itararé"
    """

    cursor.execute(query)
    resultados = cursor.fetchall()
    conn.close()
    return resultados
```

A função consultar_eventos_em_itarare realiza uma consulta no banco de dados para retornar todos os eventos que acontecem na localização específica "São Paulo, Itararé". Para isso, ela se conecta ao banco indicado por nome_banco e utiliza um comando SQL que seleciona o nome, a data e a localização dos eventos, combinando as tabelas eventos e dados_eventos através de um JOIN com base no ID. A consulta inclui um filtro (WHERE) para buscar apenas os eventos cuja localização seja exatamente "São Paulo, Itararé". Após executar a consulta, os resultados são armazenados, a conexão com o banco é encerrada, e a função retorna uma lista contendo os eventos correspondentes.

consultar_eventos_ao_ar_livre

```python
# função para mostrar os eventos ao ar livre (contendo palavras-chave específicas na descrição)
def consultar_eventos_ao_ar_livre(nome_banco):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()

    palavras_chave = ["Parque", "Praça", "Nações"]
    condicoes = " OR ".join(
        [f"metadados.descricao LIKE '%{palavra}%'" for palavra in palavras_chave]
    )

    query = f"""
    SELECT eventos.nome, dados_eventos.data, metadados.descricao
    FROM eventos
    JOIN metadados ON eventos.id = metadados.id
    JOIN dados_eventos ON eventos.id = dados_eventos.id
    WHERE {condicoes}
    """

    cursor.execute(query)
    resultados = cursor.fetchall()
    conn.close()
    return resultados
```

A função consultar_eventos_ao_ar_livre busca no banco de dados todos os eventos que possuem características de ocorrerem ao ar livre, identificados por palavras-chave específicas na descrição. Ela se conecta ao banco especificado por nome_banco e constrói dinamicamente uma condição SQL baseada em palavras como "Parque", "Praça" e "Nações". Essas palavras-chave são combinadas com o operador OR, formando um filtro que verifica se a descrição do evento, armazenada na tabela metadados, contém alguma das palavras. A consulta SQL seleciona o nome do evento, sua data e a descrição, realizando JOINs entre as tabelas eventos, metadados e dados_eventos para combinar as informações relevantes. Após executar a consulta, a função retorna os resultados como uma lista de eventos e fecha a conexão com o banco de dados.

consultar_todos_metadados

```python
# função para mostrar todos os metadados por evento
def consultar_todos_metadados(nome_banco):
    conn = sqlite3.connect(nome_banco)
    cursor = conn.cursor()

    query = """
    SELECT eventos.nome, metadados.descricao, metadados.link
    FROM eventos
    JOIN metadados ON eventos.id = metadados.id
    """

    cursor.execute(query)
    resultados = cursor.fetchall()
    conn.close()
    return resultados
```

A função consultar_todos_metadados realiza uma consulta no banco de dados para retornar os metadados de todos os eventos cadastrados. Ela conecta-se ao banco de dados especificado por nome_banco e utiliza um comando SQL que combina as tabelas eventos e metadados através de um JOIN, vinculado pelo ID do evento. A consulta retorna o nome do evento, sua descrição e o link armazenados na tabela metadados. Após executar a consulta, os resultados são armazenados em uma lista, a conexão com o banco é encerrada e a lista de metadados é retornada para ser utilizada no restante do programa.

salvar_csv

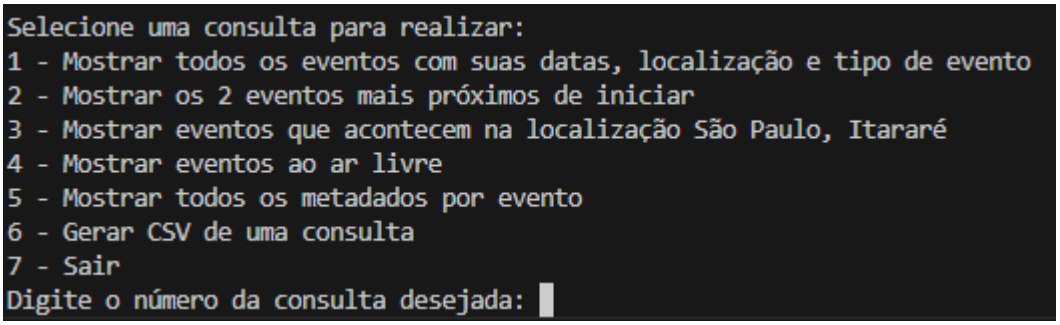
```python
# função para salvar resultados em um arquivo CSV
def salvar_csv(nome_arquivo, cabecalhos, resultados):
    try:
        with open(nome_arquivo, mode="w", encoding="utf-8", newline="") as arquivo:
            escritor = csv.writer(arquivo)
            escritor.writerow(cabecalhos)  # Escreve os cabeçalhos
            escritor.writerows(resultados)  # Escreve os dados
        print(f"Arquivo CSV '{nome_arquivo}' criado com sucesso!")
    except Exception as e:
        print(f"Erro ao salvar o arquivo CSV: {e}")
```

A função salvar_csv é responsável por criar um arquivo CSV com os dados fornecidos. Ela recebe o nome do arquivo (nome_arquivo), uma lista de cabeçalhos (cabecalhos) e os resultados da consulta (resultados). Primeiro, abre o arquivo no modo de escrita ("w") com codificação UTF-8 e utiliza o csv.writer para criar um

objeto que permite escrever no formato CSV. Em seguida, escreve os cabeçalhos no topo do arquivo com writerow e insere os dados da lista de resultados linha por linha usando writerows. Caso o processo seja bem-sucedido, exibe uma mensagem confirmando a criação do arquivo. Se ocorrer algum erro, captura a exceção e exibe uma mensagem explicativa, garantindo que o programa não seja interrompido.

Resultado da execução do arquivo

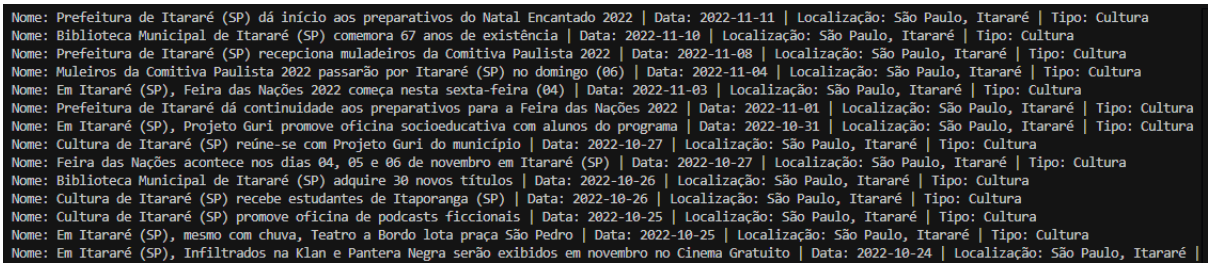
Ao executar o arquivo de consultas, será apresentado no terminal um sistema interativo para que seja selecionada a consulta desejada.



Consultas

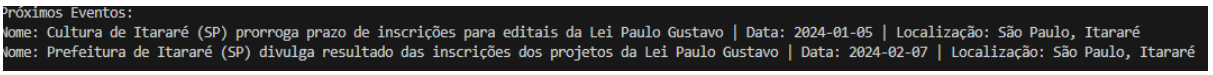
Mostrar todos os eventos com suas datas, localização e tipo de evento

Exemplo:



Mostrar os 2 eventos mais próximos de iniciar

Exemplo:



Ressalva

O site que escolhi é referente à cidade que eu moro e não possuía eventos com datas futuras (por exemplo 2025). Por isso, para que fosse possível realizar a

consulta, coloquei a data inicial de 01/01/2024 para que a query considerasse nas buscas.

Mostrar eventos que acontecem na localização São Paulo, Itararé

Exemplo:

```
Nome: Educação de Itararé (SP) inaugura parques infantis em escolas de Educação Infantil | Data: 2017-08-23 | Localização: São Paulo, Itararé
Nome: Stephanie Pavani é eleita Miss Itararé (SP) 2017 | Data: 2017-08-23 | Localização: São Paulo, Itararé
Nome: Feira da Lua em Itararé traz três atrações nesta quarta-feira (16) | Data: 2017-08-16 | Localização: São Paulo, Itararé
Nome: Quinta edição da ExpoArte de Itararé conta história do município | Data: 2017-08-16 | Localização: São Paulo, Itararé
Nome: Alunos da rede municipal de ensino de Itararé participam de formatura do Proerd | Data: 2017-08-16 | Localização: São Paulo, Itararé
Nome: 1º Festival de Música Infantil agita Feira da Lua em Itararé | Data: 2017-08-11 | Localização: São Paulo, Itararé
Nome: Cinco escolas de Educação Infantil em Itararé recebem parques infantis | Data: 2017-08-09 | Localização: São Paulo, Itararé
Nome: Prefeitura de Itararé promove Tarde Cultural nesta sexta-feira (11) | Data: 2017-08-07 | Localização: São Paulo, Itararé
Nome: Itararé elege mesa diretora para o Conselho Municipal de Política Cultural | Data: 2017-08-01 | Localização: São Paulo, Itararé
Nome: Itararé participa de encontro estadual sobre Cultura | Data: 2017-08-01 | Localização: São Paulo, Itararé
Nome: Educação de Itararé entrega kits aos alunos na volta às aulas | Data: 2017-07-26 | Localização: São Paulo, Itararé
Nome: Entre 190 municípios, Itararé (SP) é um dos três escolhidos por se destacar em ações culturais | Data: 2017-07-25 | Localização: São Paulo, Itararé
```

Mostrar todos os metadados por evento

Exemplo:

```
Nome: Itararé participa de encontro estadual sobre Cultura | Descrição: Um dos objetivos do evento foi debater a promoção da política cultural do Estado Na opo
rtunidade os coordenadores de Cultura Riveli e Turismo Edilson conheceram novo secretário Estadual José Luiz Penna ao centro Na última quinta-feira (27), o Coo
rdenador Municipal de Cultura de Itararé (SP), Alisson Riveli, acompanhado do Coordenador Municipal de Turismo, Edilson Moraes, participou […] | Link: https://
itarare.sp.gov.br/itarare-participa-de-encontro-estadual-sobre-cultura/
Nome: Educação de Itararé entrega kits aos alunos na volta às aulas | Descrição: Mais de 6 mil alunos receberão a benfeitoria Estudantes da Zona Rural foram os
 primeiros a recebê-los A Secretaria Municipal de Educação de Itararé (SP) deu início nesta terça-feira (25) a entrega dos kits escolares ao 2º semestre para o
s alunos da rede municipal de ensino. Estudantes da Zona Rural, bairros da Pedra Branca e […] | Link: https://itarare.sp.gov.br/educacao-de-itarare-entrega-kit
s-aos-alunos-na-volta-as-aulas/
Nome: Entre 190 municípios, Itararé (SP) é um dos três escolhidos por se destacar em ações culturais | Descrição: Seleção foi feita pela Secretaria Estadual da
 Cultura entre cidades que recebem Ponto MIS O coordenador de Cultura de Itararé Alisson Riveli ao centro com o diretor executivo do Ponto Mis Guilherme Pachec
o e a coordenador regional Fabíola Pinote Itararé (SP) foi uma das três cidades escolhidas por se destacar em ações do Ponto MIS. A […] | Link: https://itarare
.sp.gov.br/entre-190-municipios-itarare-sp-e-um-dos-tres-escolhidos-por-se-destacar-em-acoes-culturais/
```