CENTRO UNIVERSITÁRIO UNIFECAF
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GABRIEL MASCARENHAS DE LIMA

# **PROJETO :** SISTEMA DE GERENCIAMENTO DE EVENTOS UNIVERSITÁRIOS

GABRIEL MASCARENHAS DE LIMA

**TÍTULO**: PORTFÓLIO DE DESENVOLVIMENTO: SISTEMA DE GERENCIAMENTO DE EVENTOS UNIVERSITÁRIOS

Subtítulo: Sistema de gerenciamento de eventos em python.

Trabalho apresentado como requisito parcial de avaliação da disciplina Computational Logic Using Python do Curso de Graduação em Análise e Desenvolvimento de Sistemas do Centro Universitário UniFECAF. Tutor(a): Osvaldo Silva

TABOÃO DA SERRA, SP
2024

## SUMÁRIO



TABOÃO DA SERRA, SP
2024

## 1 INTRODUÇÃO

O programa é um sistema de gerenciamento de eventos, escrito em Python, que permite aos usuários cadastrar eventos, visualizar detalhes, inscrever participantes, e gerenciar as inscrições. Ele funciona por meio de um menu interativo, onde os usuários podem escolher diferentes opções para realizar as operações.

## 2 ESTRUTURA DO CÓDIGO

### 2.1 Dicionários de Dados

Os dicionários “eventos” e “inscricoes” são usados para armazenar informações sobre os eventos e as inscrições feitas pelos participantes.

- eventos: Armazena os detalhes de cada evento, como nome, data, descrição, número de vagas e número de inscritos.
- inscricoes: Relaciona o nome do evento a uma lista de participantes inscritos.

### 2.2 Funções

O programa utiliza funções para organizar as operações, facilitando a manutenção e expansão do código.

- **Exibir Menu**

Exibe as opções disponíveis para o usuário assim que iniciamos o sistema no terminal.

**Código:**

```
def exibir_menu():
    print("\n--- Gerenciamento de Eventos ---")
    print("1. Cadastrar novo evento.")
    print("2. Atualizar eventos.")
    print("3. Visualizar eventos cadastrados.")
    print("4. Se inscrever em um evento.")
    print("5. Visualizar inscritos.")
    print("6. Excluir eventos.")
    print("7. Sair.")
```

Essa função apresenta ao usuário as opções disponíveis no sistema. Cada opção está associada a uma funcionalidade. Cada uma será explicada abaixo:

1. **Cadastrar novo evento**

Permite cadastrar um novo evento no sistema.

**Código:**

```
def cadastrar_evento():
    print("\n--- Cadastrar Evento ---")
    nome = input("Nome do evento: ")
    if nome in eventos:
        print(f"Evento '{nome}' já cadastrado.")
    else:
        try:
            data = input("Data do evento (dd/mm/aaaa): ")
            descricao = input("Insira uma descrição: ")
            vagas = int(input("Quantidade de vagas disponíveis: "))
            eventos[nome] = {"data": data, "descrição": descricao, "vagas":
vagas, "inscritos": 0}
            print(f"Evento '{nome}' cadastrado com sucesso!")
        except ValueError:
            print("Formato inválido, utilize somente números.")
```

Essa funcionalidade permite que o usuário cadastre o evento em si. Primeiro é solicitado por meio do comando *input* que o usuário digite o nome do evento. Em seguida, por meio da condição *if nome in eventos*, verifica se já existe um evento cadastrado com este nome. Caso haja, uma mensagem informando que já existe um evento cadastrado com o mesmo nome é mostrada.

Caso contrário (*else),* o sistema pede ao usuário uma data e uma descrição no formato *string* e a quantidade de vagas disponível no formato *int*. Assim que os dados são digitados, o sistema os armazenará no dicionário vinculando as informações ao nome do evento e enviará uma mensagem avisando sobre o cadastro ter sido realizado.

Se no campo da quantidade de vagas o usuário tentar digitar letras ou qualquer coisa que não seja um número inteiro, por meio do comando *except ValueError,* o sistema retornará uma mensagem de formato inválido.

## 2. Atualizar evento

Permite alterar as informações de um evento existente.

**Código:**

```
def atualizar_eventos():
    print("\n--- Atualizar Evento ---")
    nome = input("Digite o nome do evento: ")
    if nome in eventos:
        try:
            data = input(f"Digite a nova data do evento '{nome}' (dd/mm/aaaa):
")
            descricao = input(f"Insira uma nova descrição para o evento
'{nome}': ")
            vagas = int(input(f"Digite a nova quantidade de vagas disponíveis
para o evento '{nome}': "))
            eventos[nome].update({"data": data, "descrição": descricao,
"vagas": vagas})
            print(f"Evento '{nome}' atualizado com sucesso!")
        except ValueError:
            print("Formato inválido, utilize somente números.")
    else:
        print(f"Evento '{nome}' não encontrado.")
```

Essa funcionalidade permite que eventos já cadastrados sejam modificados. Ao selecionar esta opção, o usuário digitará o nome do evento e caso exista um evento com o nome digitado, ele pedirá que sejam inseridas as novas informações (data, descrição e vagas). Em seguida os dados serão armazenados pelo sistema ligados ao nome do evento e uma mensagem avisando que o evento foi atualizado é exibida.

Caso não haja um evento cadastrado com o nome digitado, o sistema mostrará uma mensagem de evento não encontrado.

A mesma mensagem de erro da funcionalidade 1 aparecerá caso o usuário digite algo que não seja um número inteiro no campo de vagas disponíveis.

### 3. Visualizar eventos cadastrados

Exibe a lista de eventos cadastrados e seus detalhes.

**Código:**

```
def visualizar_eventos():
    print("\n--- Eventos Disponíveis ---")
    if not eventos:
        print("Nenhum evento cadastrado.")
        return

    for nome, detalhes in eventos.items():
        vagas_restantes = detalhes["vagas"] - detalhes["inscritos"]
        print(f"Nome: {nome}")
        print(f"Data: {detalhes['data']}")
        print(f"Descrição: {detalhes['descrição']}")
        print(f"Vagas restantes: {vagas_restantes}")
        print("-" * 40)
```

Essa funcionalidade busca todos os eventos já cadastrados. Caso não exista nenhum evento, o sistema retornará uma mensagem avisando que não há eventos e interrompe a execução da função, pois não há eventos para exibir. Sem o *return*, o programa continuaria executando as linhas seguintes, o que não faz sentido se não há eventos cadastrados.

Caso existam eventos adicionados, o sistema iniciará um loop que percorre todos os itens do dicionário “eventos” associando o nome como chave e as outras informações como detalhes. O sistema também fará uma contagem das vagas restantes, subtraindo a quantidade de inscrições da quantidade de vagas cadastradas.

Assim que as informações são coletadas, o sistema gerará uma lista com todos eventos e seus dados.

4. **Inscrever Participante**

Permite inscrever um participante em um evento.

**Código:**

```
def inscrever_evento():
    nome = input("Digite o nome do evento que deseja se inscrever: ")
    if nome not in eventos:
        print("Evento não existente.")
        return

    evento = eventos[nome]
    if evento["inscritos"] >= evento["vagas"]:
        print("Não há vagas neste evento.")
        return

    participante = input("Nome do participante: ")

    if nome not in inscricoes:
        inscricoes[nome] = []

    inscricoes[nome].append(participante)
    evento["inscritos"] += 1
    print(f"Inscrição realizada com sucesso para {participante} no evento
'{nome}'!")
```

Essa funcionalidade serve para que usuários se inscrevam nos eventos cadastrados.Primeiro, é solicitado o nome do evento ao usuário, caso não exista nenhum evento com o nome digitado, o sistema retornará uma mensagem avisando que o evento não existe e interrompe a execução da função com o *return*, pois não há evento para se inscrever.

Caso o nome digitado esteja no dicionário de eventos, mas não houver vagas disponíveis, o sistema mostrará uma mensagem informando que não há vagas para o evento.

Se houver vagas, o sistema solicitará o nome do participante e verificará a lista de inscritos para que um mesmo participante não se inscreva duas vezes. Se o nome

digitado não estiver na lista de inscritos, o sistema adicionará o nome a lista e informa que a inscrição foi realizada com sucesso.

5. **Visualizar Inscritos**

Exibe os participantes inscritos em um evento específico.

**Código:**

```
def visualizar_inscritos():
    print("\n--- Lista de Inscritos ---")
    nome = input("Digite o nome do evento: ")
    if nome not in eventos:
        print("Evento não cadastrado.")
        return
    if nome not in inscricoes or not inscricoes[nome]:
        print("Nenhum participante inscrito neste evento.")
        return

    print(f"Participantes inscritos no evento '{nome}':")
    for participante in inscricoes[nome]:
        print(f"- {participante}")
```

Para que seja possível visualizar a lista de inscritos em um evento, o sistema pedirá que você digite o nome do evento em questão. Se o nome digitado não existir no dicionário, ele mostrará uma mensagem informando que um evento com esse nome não está cadastrado. Se não houver nenhum inscrito, o sistema mostrará uma mensagem informando que não há inscritos.
Caso o nome digitado esteja na lista e existam participantes, o sistema fará uma lista de inscritos e mostrará na tela do terminal.

### 6. Remover Evento

Remove um evento do sistema.

**Código:**

```
def remover_evento():
    print("\n--- Remover Evento ---")
    nome = input("Nome do evento a ser removido: ")
    if nome not in eventos:
        print("Erro: Evento não encontrado.")
        return

    del eventos[nome]
    inscricoes.pop(nome, None)
    print(f"Evento '{nome}' removido com sucesso!")
```

Caso o usuário queira remover um evento da lista, ele selecionará opção 6 e o sistema pedirá o nome desse evento. Se o nome não estiver no dicionário de eventos, o sistema mostrará uma mensagem de erro informando que o evento com esse nome não foi encontrado. Caso o nome seja encontrado na lista de eventos, o sistema irá deletar o evento e os inscritos associados a esse evento e mostrará uma mensagem de sucesso na remoção do registro.

**Loop Principal**

O loop principal mantém o programa em execução até que o usuário escolha sair.

**Código:**

```
while True:
    exibir_menu()
    opcao = input("Escolha uma opção: ")

    if opcao == "1":
        cadastrar_evento()
    elif opcao == "2":
        atualizar_eventos()
    elif opcao == "3":
        visualizar_eventos()
    elif opcao == "4":
        inscrever_evento()
    elif opcao == "5":
        visualizar_inscritos()
    elif opcao == "6":
        remover_evento()
    elif opcao == "7":
        print("Encerrando o sistema. Até logo!")
        break
    else:
        print("Opção inválida. Tente novamente.")
```

O loop acima é o responsável por gerar o menu e fazer com que a escolha das opções resulte em uma resposta. Para cada funcionalidade citada anteriormente, foi atribuído um número, esse número foi utilizado no loop para indicar a função ligada a funcionalidade. Para sair do sistema, o usuário pode digitar o número 7 e o sistema mostrará uma mensagem de despedida e encerra o loop. Caso seja selecionado qualquer número que não uma das opções listadas, será mostrada uma mensagem de opção inválida.

## 3 CONCLUSÃO

Este projeto de gerenciamento de eventos mostrou como a programação pode ajudar a organizar tarefas do dia a dia de forma simples e eficiente. Usando recursos como dicionários, funções e loops, criei um sistema que permite cadastrar, atualizar, visualizar e excluir eventos, além de gerenciar as inscrições dos participantes.

Cada função foi pensada para fazer uma tarefa específica, deixando o código organizado e fácil de entender. Além disso, o cuidado com mensagens claras e a validação de entradas ajudaram a tornar o sistema mais prático e confiável para quem usa.

O sistema também pode crescer, caso seja necessário. Por exemplo, podemos incluir novas funções, como enviar avisos para os participantes, gerar relatórios mais completos ou até mesmo colocar uma senha para diferenciar um usuário comum de um administrador. Isso mostra como a linguagem Python é versátil e útil para criar programas que atendem a várias necessidades.

Por fim, trabalhar neste projeto foi uma ótima forma de aprender na prática. Resolver problemas reais e encontrar soluções ajudou a fixar os conceitos de programação e a preparar o terreno para desafios maiores no futuro.

TABOÃO DA SERRA, SP
2024