Aula 5 – Tratamento de Exceções e Aplicações com Arquivos

Objetivo da Aula

Conhecer e aplicar a técnica de tratamento de exceções, o uso de arquivos de texto e o uso de arquivos binários.

Apresentação

Em programação, o tratamento de exceções é uma técnica importante que ajuda a melhorar a confiabilidade, estabilidade e qualidade do software. Ele permite que o programa lide com exceções e erros de forma eficaz, identifique problemas no software e melhore a experiência do usuário.

Outro recurso em programação é o uso de arquivos para armazenar e gerenciar grandes volumes de dados, no formato texto ou no formato binário, para armazenar dados em outros formatos, como áudio e vídeo.

Em Python, é possível trabalhar com o tratamento de exceções e com o armazenamento de arquivos nos formatos de texto e binário.

1. Tratamento de Exceções

O tratamento de exceções é uma técnica importante para lidar com erros e exceções em programas. Quando uma exceção ocorre, o programa pode ser interrompido, mas também pode ser configurado para lidar com a exceção, de maneira adequada, exibindo uma mensagem de erro, tomando uma ação alternativa ou retornando um valor apropriado. Isso permite que o programa continue executando, mesmo em caso de erros, aumentando a confiabilidade e a robustez do software.

O tratamento de exceções no Python é feito com o uso dos comandos try, except, else e finally. O bloco try contém o código que pode gerar uma exceção, enquanto o bloco except contém o código que será executado se ocorrer uma exceção. O bloco else é opcional e é executado se o bloco try for executado com sucesso, sem gerar nenhuma exceção. O bloco

finally também é opcional e é sempre executado, independentemente de ocorrer ou não uma exceção.

Veja um exemplo de código que usa o tratamento de exceções:

```python
try:
    # Código que pode gerar exceção
    x = int(input("Digite um número: "))
    y = int(input("Digite outro número: "))
    resultado = x / y
    print(f"O resultado da divisão é: {resultado}")

except ValueError:
    # Código para tratar exceção de valor inválido
    print("Você deve digitar apenas números inteiros.")

except ZeroDivisionError:
    # Código para tratar exceção de divisão por zero
    print("Não é possível dividir por zero.")

else:
    # Código que será executado se não houver exceções
    print("Cálculo realizado com sucesso!")

finally:
    # Código que será executado sempre, com ou sem exceção
    print("Programa encerrado.")
```

Nesse exemplo, o programa solicita dois números ao usuário e tenta realizar a divisão. Se um dos valores não for um número inteiro ou se ocorrer uma divisão por zero, o programa trata a exceção e exibe uma mensagem apropriada. Se a divisão for bem-sucedida, o programa exibe o resultado e uma mensagem de sucesso. O bloco finally exibe uma mensagem de encerramento do programa, independentemente de ocorrer ou não uma exceção.

Você pode definir quantos blocos except forem necessários para tratar diferentes tipos de exceções que possam ocorrer em seu código. É uma boa prática tratar exceções específicas e evitar exceções genéricas que podem mascarar erros ou tornar seu código mais difícil de depurar.

2. Arquivos de Texto

Arquivos de texto são uma maneira fácil e conveniente de armazenar dados em um formato legível por humanos e computadores. Eles podem armazenar uma grande quantidade de informações, incluindo textos simples, números e caracteres especiais. Para trabalhar com arquivos de texto no Python, você pode seguir os seguintes passos:

- Abrir o arquivo: você pode abrir um arquivo usando a função open(). Essa função recebe dois argumentos – o nome do arquivo e o modo em que você deseja abrir o arquivo. Existem vários modos disponíveis para abrir um arquivo, mas os principais são:
  - 'r': modo leitura (read). Abre o arquivo para leitura. Esse é o modo padrão, caso nenhum seja especificado;
  - 'w': modo escrita (write). Abre o arquivo para escrita. Se o arquivo não existir, ele será criado. Se existir, o arquivo será truncado;
  - 'a': modo adição (append). Abre o arquivo para adicionar conteúdo ao final. Se o arquivo não existir, ele será criado.

Por exemplo, para abrir um arquivo chamado meuarquivo.txt em modo de leitura, você pode usar o seguinte código:

```
arquivo = open('meuarquivo.txt', 'r')
```

- Ler o arquivo: existem várias maneiras de ler o conteúdo de um arquivo de texto no Python. A forma mais simples é usar o método read() do objeto arquivo, que lê todo o conteúdo do arquivo de uma só vez e retorna uma string.

```
conteudo = arquivo.read()
print(conteudo)
```

- Escrever no arquivo: para escrever em um arquivo de texto, você pode usar o método write() do objeto arquivo. Esse método recebe uma string como argumento e escreve essa string no arquivo.

```
arquivo = open('meuarquivo.txt', 'w')
arquivo.write('Olá, mundo!\n')
arquivo.write('Essa é uma nova linha.')
arquivo.close()
```

Nesse exemplo, abrimos o arquivo no modo de escrita ('w') e escrevemos duas linhas no arquivo. Observe que adicionamos o caractere de nova linha (\n), no final da primeira linha, para que a segunda linha seja escrita em uma nova linha.

3. Arquivos Binários

Arquivos binários são usados em Python para armazenar e ler dados em formato binário, em oposição aos arquivos de texto que armazenam dados em formato legível por humanos.

Ao usar arquivos binários, é possível armazenar dados complexos, como imagens, áudio e vídeo, além de dados numéricos e de texto.

Para trabalhar com arquivos binários em Python, você precisa seguir os seguintes passos:

- Abra um arquivo binário, usando a função open() e especificando o modo de abertura como “rb” para leitura binária ou “wb” para gravação binária:

```
1   file = open("filename.bin", "rb")
```

- Leia ou grave dados no arquivo binário, usando as funções read() e write(). A função read() retorna os dados lidos do arquivo, enquanto a função write() grava dados no arquivo. Feche o arquivo binário usando a função close():

```
2   # Lendo dados do arquivo binário
3   data = file.read()
4   # Gravando dados no arquivo binário
5   file.write(data)
6   file.close()
```

É importante lembrar que, ao usar arquivos binários, você precisa estar ciente dos tipos de dados que está lendo ou gravando no arquivo. Você precisa garantir que os dados estejam em um formato adequado e compatível com o arquivo binário. Por exemplo, se você estiver lendo um arquivo binário que contém dados numéricos, deve garantir que esteja lendo os dados no tipo correto, como inteiros ou floats, e que estejam em um formato adequado.

Além disso, para trabalhar com arquivos binários, você pode usar algumas bibliotecas Python úteis, que permitem usar algumas funcionalidades importantes.

4. Exemplo Prático

Vamos agora analisar um código completo em Python. Esse código implementa um sistema simples de gerenciamento de produtos, em que os dados são armazenados em um arquivo de texto chamado produtos.txt. O usuário pode executar as seguintes ações:

- Incluir um novo produto na lista;
- Consultar um produto pelo código;
- Alterar um produto pelo código;
- Excluir um produto pelo código;

- Listar todos os produtos.

```
# Inicializa a lista de produtos vazia
produtos = []

# Verifica se existe um arquivo de produtos e o carrega
try:
    with open('produtos.txt', 'r') as arquivo:
        for linha in arquivo:
            codigo, nome, quantidade = linha.strip().split(',')
            produtos.append({'codigo': codigo, 'nome': nome, 'quantidade': int
            (quantidade)})
except FileNotFoundError:
    pass
```

A primeira linha define uma lista vazia chamada de "produtos", que será usada para armazenar os dados dos produtos.

A seguir, há um bloco de código try/except, que tenta abrir o arquivo produtos.txt em modo de leitura ('r'). Se o arquivo existir, o código lê cada linha do arquivo, divide a linha em três valores separados por vírgulas (o código do produto, o nome do produto e a quantidade em estoque), converte a quantidade para um inteiro e adiciona um dicionário representando o produto à lista produtos.

A linha é formatada como uma string com três campos separados por vírgulas: código, nome e quantidade. A função strip() é usada para remover qualquer espaço em branco no início e no final da string.

A seguir, a função split(',') é usada para dividir a string em três substrings, usando a vírgula como separador. Isso produz uma lista de três elementos, contendo o código, o nome e a quantidade do produto. Se o arquivo não existir, o bloco except passa para a próxima instrução.

Finalmente, a linha atribui cada um dos elementos da lista a uma variável separada, na mesma ordem em que aparecem na lista. Assim, a variável codigo recebe o primeiro elemento da lista, a variável nome recebe o segundo elemento, e a variável quantidade recebe o terceiro elemento. O resultado é que as informações do produto

são armazenadas em três variáveis separadas, tornando mais fácil as manipular posteriormente no programa.

```
12
13  while True:
14      # Exibe o menu de opções
15      print('\n\nEscolha uma opção:')
16      print('1 - Incluir produto')
17      print('2 - Consultar produto')
18      print('3 - Alterar produto')
19      print('4 - Excluir produto')
20      print('5 - Listar produtos')
21      print('0 - Sair')
22
23      # Lê a opção escolhida pelo usuário
24      opcao = input('Digite a opção desejada:')
```

O código entra em um loop while True que exibe um menu com as opções disponíveis para o usuário e lê a opção escolhida pelo usuário. Dependendo da opção escolhida, o código executa uma ação diferente.

```
25
26  if opcao == '1':
27      # Inclui um novo produto na lista
28      codigo = input('Código: ')
29      nome = input('Nome: ')
30      quantidade = input('Quantidade: ')
31      produtos.append({'codigo': codigo, 'nome': nome, 'quantidade': int(quantidade)})
32
33      # Grava a lista de produtos no arquivo
34      with open('produtos.txt', 'w') as arquivo:
35          for produto in produtos:
36              arquivo.write(f"{produto['codigo']},{produto['nome']},{produto['quantidade']}\n")
37
38      print('Produto incluído com sucesso!')
39
```

Se o usuário escolher a opção 1, o código pede para o usuário digitar o código, o nome e a quantidade do produto e adiciona um novo dicionário de produto à lista produtos. Em seguida, grava a lista atualizada de produtos no arquivo produtos.txt.

```
40  elif opcao == '2':
41      # Consulta um produto na lista
42      codigo = input('Código: ')
43      for produto in produtos:
44          if produto['codigo'] == codigo:
45              print(f"Nome: {produto['nome']}")
46              print(f"Quantidade: {produto['quantidade']}")
47              break
48      else:
49          print('Produto não encontrado!')
50
```

Se o usuário escolher a opção 2, o código pede para o usuário digitar o código do produto e, em seguida, percorre a lista de produtos, procurando por um produto com o código correspondente. Se encontrar, imprime o nome e a quantidade do produto. Caso contrário, informa que o produto não foi encontrado.

```
50
51      elif opcao == '3':
52          # Altera um produto na lista
53          codigo = input('Código: ')
54          for produto in produtos:
55              if produto['codigo'] == codigo:
56                  nome = input('Novo nome: ')
57                  quantidade = input('Nova quantidade: ')
58                  produto['nome'] = nome
59                  produto['quantidade'] = int(quantidade)
60
61                  # Grava a lista de produtos no arquivo
62                  with open('produtos.txt', 'w') as arquivo:
63                      for produto in produtos:
64                          arquivo.write(f"{produto['codigo']},{produto['nome']},{produto['quantidade']}\n")
65
66                  print('Produto alterado com sucesso!')
67                  break
68          else:
69              print('Produto não encontrado!')
```

Se o usuário escolher a opção 3, o código pede para o usuário digitar o código do produto a ser alterado. Se encontrar, o código pede ao usuário que digite o novo nome e a quantidade para o produto e atualiza o dicionário de produto correspondente na lista produtos. Em seguida, grava a lista atualizada de produtos no arquivo produtos.txt. Caso contrário, informa que o produto não foi encontrado.

```
70
71      elif opcao == '4':
72          # Exclui um produto na lista
73          codigo = input('Código: ')
74          for i, produto in enumerate(produtos):
75              if produto['codigo'] == codigo:
76                  del produtos[i]
77
78                  # Grava a lista de produtos no arquivo
79                  with open('produtos.txt', 'w') as arquivo:
80                      for produto in produtos:
81                          arquivo.write(f"{produto['codigo']},{produto['nome']},{produto['quantidade']}\n")
82
83                  print('Produto excluído com sucesso!')
84                  break
85          else:
86              print('Produto não encontrado!')
87
```

Se o usuário escolher a opção 4, o código pede para o usuário digitar o código do produto a ser excluído. Se encontrar, o código remove o dicionário de produto correspondente da

lista produtos. Em seguida, grava a lista atualizada de produtos no arquivo produtos.txt. Caso contrário, informa que o produto não foi encontrado.

```
87
88      elif opcao == '5':
89          # Lista todos os produtos da lista
90          print('Lista de produtos:')
91          for produto in produtos:
92              print(f"Código: {produto['codigo']}")
93              print(f"Nome: {produto['nome']}")
94              print(f"Quantidade: {produto['quantidade']}")
95      elif opcao == '0':
96          break
```

Se o usuário escolher a opção 5, o código lista todos os produtos na lista produtos. Se o usuário escolher a opção 0, o código interrompe o loop e sai do programa.

Considerações Finais da Aula

O tratamento de exceções em programação é uma técnica importante que ajuda a melhorar a robustez e a confiabilidade do software. O tratamento de exceções é usado para lidar com situações imprevistas ou erros que podem ocorrer durante a execução do programa. Algumas das razões pelas quais o tratamento de exceções é importante são: manutenção de estabilidade, identificação de erros, melhoria da experiência do usuário e possibilidade gerenciar erros em tempo de execução.

Os arquivos de texto são uma maneira fácil e conveniente de armazenar dados em um formato legível e são uma forma eficaz e versátil de compartilhar dados em programação, devido à sua facilidade de uso, portabilidade e compatibilidade com versões mais antigas de programas.

Os arquivos binários podem armazenar informações de forma mais rápida e com menos espaço de armazenamento do que arquivos de texto. Isso os torna uma escolha popular para armazenar dados de alta intensidade, como imagens, vídeo, áudio e outros arquivos multimídia.

Materiais Complementares

Livro:

Python e Django
2020, Francisco Marcelo de B. Maciel. Editora Alta Books.

Esse livro apresenta mais informações sobre os conceitos básicos apresentados nesta aula sobre as características da linguagem Python, a sua sintaxe e os seus comandos básicos.

Site

AlexandreLouzada/Pyquest: Pyquest/envExemplo/Lista05 e Pyquest/envExemplo/Lista06
2023, Alexandre N. Louzada. Github.
Os links indicados permitem o acesso a um repositório com várias listas de exemplo de programas em Python utilizando arquivos.
Links para acesso: https://github.com/AlexandreLouzada/Pyquest/tree/master/envExemplo/Lista05 e https://github.com/AlexandreLouzada/Pyquest/tree/master/envExemplo/Lista06 (acessos em 24 maio 2023.)

Referências

ALVES, William P. ***Programação Python***: aprenda de forma rápida. [s.l.]: Editora Saraiva, 2021.

PYTHON SOFTWARE FOUNDATION. ***Python Language Site***. Documentation, 2023. Página de documentação. Disponível em: https://docs.python.org/pt-br/3/tutorial/index.html. Acesso em: 8 mar. 2023.