Tratamento de Erros e Exceções

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins

Objetivos de aprendizado

- Compreender o conceito de erros e exceções.
- Aprender a identificar diferentes tipos de erros.
- Tratar erros e exceções utilizando blocos `try...except`.

O que são Erros e Exceções?

- **Erro (Error):** Um erro é um problema que ocorre durante a execução do programa. Pode ser detectado em tempo de compilação (erros de sintaxe) ou durante a execução (erros em tempo de execução).
- **Exceção (Exception):** Uma exceção é um tipo específico de erro que interrompe o fluxo normal da execução do programa. Quando uma exceção ocorre, o interpretador Python tenta encontrar um manipulador para ela. Se não encontrar, o programa termina abruptamente com uma mensagem de erro.

Tipos Comuns de Erros em Python:

- `SyntaxError`: Erro de sintaxe no código (ex: parênteses desbalanceados, dois pontos ausentes).
- `TypeError`: Operação ou função aplicada a um tipo de dado inadequado (ex: somar uma string com um inteiro).
- `NameError`: Variável não definida.
- `IndexError`: Tentativa de acessar um índice inválido em uma lista ou tupla.
- `KeyError`: Tentativa de acessar uma chave inexistente em um dicionário.
- `ValueError`: Função recebe um argumento do tipo correto, mas com valor inadequado (ex: converter uma string não numérica para inteiro).
- `FileNotFoundError`: Arquivo que se espera encontrar não existe.
- `ZeroDivisionError`: Tentativa de dividir por zero.

Tratamento de Exceções com `try...except`

A estrutura `try...except` permite que você "capture" e trate exceções, evitando que o programa termine abruptamente.

```
try:
    # Código que pode gerar uma exceção
    resultado = 10 / 0  # Isso causará um ZeroDivisionError
    print("Resultado:", resultado) # Esta linha não será executada se a divisão por zero ocorrer
except ZeroDivisionError as e:
    # Código para lidar com a exceção ZeroDivisionError
    print(f"Erro: Divisão por zero! Detalhes: {e}")
except Exception as e:  # Captura qualquer outra exceção (boa prática)
    print(f"Ocorreu um erro inesperado: {e}")
else:
    # Código executado se NENHUMA exceção ocorrer no bloco try
    print("Operação realizada com sucesso!")
finally:
    # Código que SEMPRE será executado, independentemente de ter ocorrido ou não uma exceção.
    print("Finalizando a operação.")
```

Explicação

- `try:` : O código dentro do bloco `try` é monitorado para possíveis exceções.
- `except NomeDaExcecao as variavel:` : Se uma exceção do tipo `NomeDaExcecao` ocorrer no bloco `try`, o código dentro deste bloco `except` será executado. A variável `e` (opcional) armazena a instância da exceção, permitindo acessar informações sobre ela.
- `else:` : O código dentro do bloco `else` é executado apenas se *nenhuma* exceção ocorrer no bloco `try`.
- `finally:` : O código dentro do bloco `finally` é sempre executado, independentemente de ter ocorrido ou não uma exceção. É útil para liberar recursos (ex: fechar arquivos).

Exemplo 1: Lendo um Arquivo

```
try:
    with open("arquivo_inexistente.txt", "r") as f:
        conteudo = f.read()
except FileNotFoundError:
    print("Erro: O arquivo não foi encontrado.")
except Exception as e:
    print(f"Ocorreu um erro ao ler o arquivo: {e}")
```

Exemplo 2: Convertendo uma String para Inteiro

```
entrada = input("Digite um número inteiro: ")
try:
    numero = int(entrada)
    print("Número convertido:", numero)
except ValueError:
    print("Erro: A entrada não é um número inteiro válido.")
```

Exemplo 3: Usando o comando `finally`

```
try:
    # Código que pode gerar uma exceção
    resultado = 10 / 0  # Isso causará um ZeroDivisionError
    print("Resultado:", resultado) # Esta linha não será executada se a divisão por zero ocorrer
except ZeroDivisionError as e:
    # Código para lidar com a exceção ZeroDivisionError
    print(f"Erro: Divisão por zero! Detalhes: {e}")
except Exception as e:  # Captura qualquer outra exceção (boa prática)
    print(f"Ocorreu um erro inesperado: {e}")
else:
    # Código executado se NENHUMA exceção ocorrer no bloco try
    print("Operação realizada com sucesso!")
finally:
    # Código que SEMPRE será executado, independentemente de ter ocorrido ou não uma exceção.
    print("Finalizando a operação.")
```

6. Boas Práticas

- **Seja Específico:** Capture apenas as exceções que você espera e pode lidar adequadamente. Evite usar `except Exception:` como a única forma de tratamento, pois isso pode mascarar erros inesperados.
- **Registre os Erros:** Use logging para registrar informações sobre as exceções que ocorrem, o que facilita a depuração do programa.
- **Levante Exceções (Raise):** Em algumas situações, você pode querer levantar uma exceção novamente após tratá-la ou adicionar informações adicionais à exceção existente. Use `raise` para isso.
- **Documente:** Comente seu código explicando como as exceções são tratadas e por que.

Exercícios Práticos

1. **Divisão Segura:** Escreva uma função que receba dois números como entrada e retorne o resultado da divisão, tratando a exceção `ZeroDivisionError`.
2. **Leitura de Arquivo com Tratamento de Erros:** Crie um programa que leia um arquivo especificado pelo usuário. Trate as exceções `FileNotFoundError` e `IOError`.
3. **Conversão de Lista para Inteiros:** Escreva uma função que receba uma lista de strings como entrada e tente converter cada string em um inteiro. Use `try...except` para lidar com os casos em que a conversão falha.
4. **Simulação de Erro:** Crie um código que intencionalmente cause um `TypeError`. Utilize o bloco `try...except` para capturar e tratar essa exceção.