TRATAMENTO DE ERROS E EXCEÇÕES

IFBA – Campus Ilhéus

Curso: Back End com Python

Objetivo da Aula

- Ensinar como identificar, capturar e tratar erros para tornar programas robustos.

O que são exceções?

- São erros ocorridos durante a execução do programa (runtime), interrompendo seu fluxo normal.
- Exemplo comuns:
  - Divisão por zero, conversão inválida de texto para número ou acesso a uma posição inexistente em uma lista.
  - ZeroDivisionError, ValueError, IndexError, FileNotFoundError.

Exceções

- Nesse momento o interpretador "lança" uma exceção. Se essa exceção não for tratada, o programa termina com um erro (traceback).
- O tratamento de exceções permite que o programa reaja de forma controlada a esses problemas.
- Essencial para criar programas mais seguros, estáveis e fáceis de depurar.

Bloco try / except

- A construção básica para tratar exceções:

```
try:
    # código que pode gerar exceção
except TipoDaExcecao:
    # tratamento específico
except (Tipo1, Tipo2) as e:
    # tratar múltiplos tipos
except Exception as e:
    # tratamento genérico (preferir evitar quando possível)
else:
    # executa se não houve exceção
finally:
    # executa sempre (ocorra exceção ou não)
```

Bloco try / except

- Exemplo prático:

```
try:
    x = int(input("Digite um inteiro: "))
    y = int(input("Digite outro inteiro: "))
    resultado = x // y
except ValueError:
    print("Por favor, digite apenas números inteiros.")
except ZeroDivisionError:
    print("Não é possível dividir por zero.")
else:
    print("Resultado:", resultado)
finally:
    print("Fim da operação.")
```

Bloco try / except

- Boas práticas:
  - Trate exceções específicas sempre que possível (não use except: sem tipo).
  - Mantenha o bloco try pequeno — só o código que pode lançar a exceção que você quer tratar.

Tratamento de exceções comuns

- ValueError — conversões inválidas

```
s = "abc"
try:
    n = int(s)
except ValueError:
    print("Não foi possível converter para inteiro.")
```

Tratamento de exceções comuns

- ZeroDivisionError — divisão por zero

```
try:
    r = 10 / 0
except ZeroDivisionError:
    print("Divisão por zero detectada.")
```

Tratamento de exceções comuns

- IndexError — índice fora do intervalo

```
lista = [1, 2, 3]
try:
    print(lista[10])
except IndexError:
    print("Índice inválido na lista.")
```

Tratamento de exceções comuns

- KeyError — chave inexistente em dicionário

```
d = {"a": 1}
try:
    print(d["b"])
except KeyError:
    print("Chave não encontrada.")
```

Tratamento de exceções comuns

- FileNotFoundError — arquivo não encontrado

```
try:
    with open("arquivo.txt", "r") as f:
        conteudo = f.read()
except FileNotFoundError:
    print("Arquivo não encontrado.")
```

Uso de finally

- O bloco finally é útil para liberar recursos (fechar arquivos, conexões, liberar locks) que devem ser executados sempre, mesmo se ocorrer exceção ou return.
- Exemplo:

Uso de finally

```
f = None
try:
    f = open("dados.txt", "r")
    primeiro = f.readline()
    print(primeiro)
except FileNotFoundError:
    print("Arquivo ausente.")
finally:
    if f:
        f.close()    # garante fechamento mesmo após erro
```

Uso de finally

- Observações:
  - Quando possível, prefira o gerenciador de contexto with (que cuida automaticamente do fechamento), mas finally continua útil para outros recursos.
  - Exemplo:

```
def exemplo():
    try:
        return "valor do try"
    finally:
        print("finally executa mesmo após return")
```

Lançando exceções com raise

- É possível (e devemos) lançar nossas próprias exceções quando detectarmos situações inválidas dentro do código, para sinalizar erro para quem chamou a função.
- Exemplo:

Lançando exceções com raise

```
def verificar_idade(idade):
    if idade < 18:
        raise ValueError("A idade mínima é 18 anos.")
    print("Acesso permitido.")

# Chamada da função
idade_usuario = int(input("Digite sua idade: "))
verificar_idade(idade_usuario)

print("Esta mensagem só aparece se não ocorrer exceção.")
```

Resumo de boas práticas

- Prefira capturar exceções específicas.
- Não sufoque exceções (capturar e ignorar).
- Mantenha try pequeno e localizado ao código que pode falhar.
- Use with para recursos (arquivos, conexões).
- Lance (raise) exceções quando sua função detectar uso incorreto.

Exercícios

1. Escreva um programa que peça ao usuário dois números inteiros e exiba o resultado da divisão inteira. Trate ValueError (entrada inválida) e ZeroDivisionError (divisão por zero). Mostre mensagens adequadas ao usuário.

Objetivo: usar try/except.

Exercícios

2. Implemente uma função def validar_idade(idade): que:
   - Recebe um número.Se idade for negativa, lance ValueError com a mensagem "Idade não pode ser negativa".
   - Caso contrário, retorne True.

Crie um pequeno programa que leia a idade, chame validar_idade() e trate a exceção.

Objetivo: usar raise e tratamento.

Exercícios

3. Crie uma lista com 5 elementos. Peça ao usuário um índice e mostre o elemento correspondente. Trate ValueError (entrada não numérica) e IndexError (índice fora do intervalo).

Objetivo: capturar várias exceções.

Exercícios

4. Faça um programa que:
   - Peça um número inteiro.
   - Dentro de try, converta a entrada para int.
   - Use else para calcular e imprimir o dobro do número (apenas se a conversão tiver sucesso).
   - Use except para informar erro de conversão.

   Objetivo: demonstrar else em blocos de exceção.