Tratamento de Exceções

Prof. Gabriel Rodrigues Caldas de Aquino

gabrielaquino@ic.ufrj.br
Instituto de Computação - Universidade Federal do Rio de Janeiro

Compilado em:
October 5, 2025

Afinal qual o motivo de tratarmos exceções?

Antes de começarmos o nosso assunto, uma reflexão...

Qual o efeito de um código que não funciona corretamente?

Veja o caso to THERAC-25

Erro de Sintaxe Vs. Exceção

Pontos de observação

- **Erros de Sintaxe**:
  - Detectados antes da execução
  - Exemplo: esquecer dois-pontos (`:`) em um `if`
  - Mensagem mostra o local aproximado do erro antes de executar o código
- **Exceções**:
  - Ocorrem durante a execução do código
  - Representam erros lógicos ou condições inesperadas
    - Exemplos: divisão por zero, tipo incorreto
  - Mostram tipo da exceção e (ex: `ZeroDivisionError`) encerram a execução do código

Erro de Sintaxe Vs. Exceção

Principais Diferenças

- **Erro de Sintaxe**:
  - Impede a execução do código
  - Ou seja o código não roda
  - Precisa ser corrigido antes de executar
- **Exceção**:
  - Ocorre **durante** a execução do código
  - Pode ser tratada com `try-except`
  - Caso seja tratada, o funcionamento do código pode ser recuperado

Exemplos de Erros de Sintaxe

Exemplos de Erros de Sintaxe

- **Esquecer os dois-pontos**:

```
if x > 5   # ERRO: falta ':'
    print("Maior que 5")
```

- **Parênteses não fechados**:

```
print("Olá, mundo"   # ERRO: falta ')'
```

Exemplos de Exceções em Python

Exemplos Práticos

- **ZeroDivisionError**:

```
10 / 0   # Tenta dividir por zero
```

- **NameError**:

```
print(var_inexistente)  # Variável não definida
```

- **TypeError**:

```
"2" + 2  # Concatenação de tipos incompatíveis
```

- **IndexError**:

```
lista = [1, 2]
lista[3]  # Acesso a índice inexistente
```

Mas o que é uma exceção?

Definição

Uma exceção é um acontecimento inesperado ou incomum no fluxo normal do código.

Características principais

- Situações que fogem do comportamento esperado do código
  - Podemos prever ou não
- Códigos podem lançar exceções intencionalmente
- `ZeroDivisionError` e `ValueError` são exemplos de exceções

Então, exceção é isso!

Pontos principais

- **Interrompem** no fluxo normal do programa
- Ocorrem quando algo **inesperado** acontece
- Exemplos:
  - Tentar abrir um arquivo que não existe
  - Dividir um número por zero
  - Acessar uma posição inválida em uma lista

Por que tratar exceção é importante?

- Permitem **recuperar** o programa de erros
- Evitam que o programa **trave** completamente
- Oferecem **feedback** útil para depuração

Entendendo as Mensagens de Erro

Mensagem de Erro

- **Traceback**:
  - Histórico que levou ao erro
- **Localização**:
  - `path/to/file.py`
  - Linha 3: `c=a/b`
- **Tipo da Exceção**:
  - Classe do erro
  - ex: `ZeroDivisionError`
- **Descrição**:
  - Explicação legível do problema
  - ex: "division by zero"

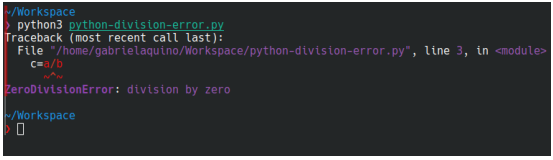


Figure: Exemplo de mensagem

Try/Except

Código Original

```
a = 10
b = 0
c = a / b  # Problema aqui
print(c)
```

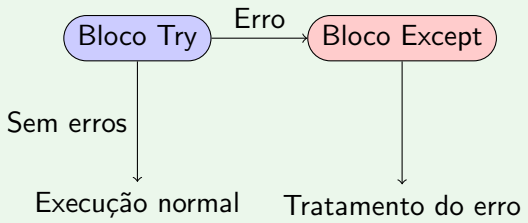
Versão Corrigida com Try/Except

```
a = 10
b = 0

try:
    c = a / b
except ZeroDivisionError:
    print("Erro: Divisão por zero")
    c = float('inf') # Valor padrão
```

Fluxo de Execução

Fluxo de Execução



Como podemos fazer o tratamento?

- Especificar o tipo de exceção
- Registrar que ocorreu um problema
- Definir valores padrão caso tenha um problema

Exemplo de tratamento

Exemplo de tratamento

```
try:
    print("Início do bloco try")
    x = 10 / 0  # O problema ocorre aqui
    print("Esta linha NUNCA será executada")
except ZeroDivisionError:
    print("Erro capturado - divisão por zero")
```

Qual problema temos aqui?

Código Vulnerável

```
idade = int(input("Digite sua idade: "))
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

O que pode dar errado?

Porque esse código é vulnerável ?

Problema com Entrada do Usuário

Código Vulnerável

```
idade = int(input("Digite sua idade: "))
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

O que pode dar errado?

- ValueError: Se usuário digitar "dez" em vez de 10

Solução com Tratamento de Exceções

Código com o tratamento de exceção

```
try:
    idade = int(input("Digite sua idade: "))
    if idade >= 18:
        print("Maior de idade")
    else:
        print("Menor de idade")
except ValueError:
    print("Por favor, digite apenas números!")
```

O que pode dar errado neste codigo?

Discutam o código abaixo

```
num = int(input("Número: "))
print(f"Resultado: {100 / num}")
```

Riscos levantados

Pontos de risco

```
num = int(input("Número: "))          # Risco 1: Valor não inteiro
print(f"Resultado: {100 / num}")      # Risco 2: Divisão por zero
```

- ValueError:
  - Entradas como "vinte", "a"...
- ZeroDivisionError:
  - Divisão por zero caso num=0

Solução de Tratamento

Código com Tratamento de Erros

```
try:
    num = int(input("Número: "))
    print(f"Resultado: {100 / num}")
except ValueError:
    print("Digite um número inteiro válido!")
except ZeroDivisionError:
    print("Não pode ser zero!")
```

Exemplo de Execuções

- Entrada "10" → Resultado: 10.0
- Entrada "0" → "Não pode ser zero!"
- Entrada "abc" → "Digite um número inteiro válido!"

O Bloco `else` no Tratamento de Exceções

Quando usar?

O bloco `else` executa **somente se**:

- O bloco `try` for concluído **sem erros**
- Nenhuma exceção foi levantada

Diferença entre fluxos

```
try:
    # Código que pode falhar
except MinhaExcecao:
    # Executa SE ocorrer erro
else:
    # Executa SE NÃO ocorrer erro
finally:
    # Executa SEMPRE
```

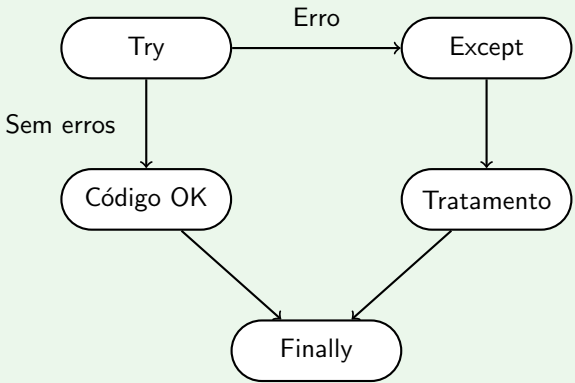
O bloco `finally`

Características do `finally`

- Sempre executa, independentemente:
  - Se ocorrer erro **ou não**
  - Se o erro foi tratado **ou não**
  - Se houve `return` no bloco
- Uso típico para:
  - Liberar recursos (arquivos, conexões)
  - Fazer limpeza
  - Registrar/logging de operações

Fluxo Try-Except-Finally

Fluxo de Execução com Tratamento de Exceções com o finally



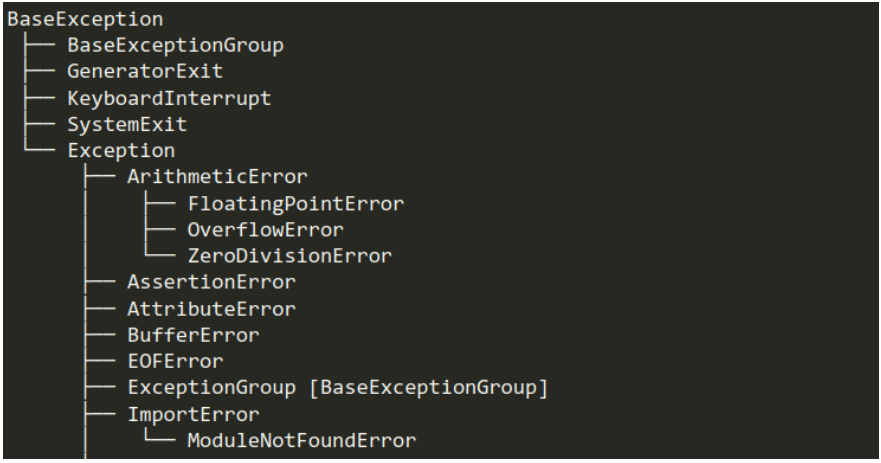
O Bloco `finally` em Python

Funcionamento Básico

```
arquivo = None
try:
    arquivo = open("dados2.txt", "r")
    # Operações com o arquivo
except FileNotFoundError:
    print("Arquivo não encontrado!")
finally:
    print("Sempre executa")
    if arquivo != None:
        arquivo.close()  # Garante o fechamento
    else:
        print(arquivo)
```

Hierarquia de Exceções em Python

Todas as exceções herdam de `BaseException`. Veja "Exception hierarchy" em `https://docs.python.org/3/library/exceptions.html`



Capturando Exceções Genéricas

Exemplo Prático

```
try:
    num = int(input("Digite um número: "))
    resultado = 100 / num
    print(f"Resultado: {resultado}")

except Exception as err:
    print(f"Ocorreu um erro: {type(err).__name__}")
    print(f"Mensagem: {str(err)}")
    print(f"Detalhes completo: {err}")
```

Tratando Diferentes Tipos de Exceções

Exemplo Completo

```
try:
    num = int(input("Digite um número (não zero): "))
    resultado = 100 / num
    print(f"Resultado: {resultado:.2f}")

except ZeroDivisionError:
    print("Erro: Não é possível dividir por zero!")
except ValueError:
    print("Erro: Digite apenas números inteiros!")

except BaseException as err:
    print()
    print(f"Ops! {type(err).__name__}")
    print(f"Fim!")
```

Exceções Personalizadas

Como criar uma exceção básica

```
class MeuErroCustomizado(Exception):
    pass

raise MeuErroCustomizado("Mensagem de erro especial")
```

Exemplo Prático

```
try:
    raise MeuErroCustomizado("Algo deu errado!")
except MeuErroCustomizado as erro:
    print(f"Erro capturado: {erro}")
```

Saída:

```
Erro capturado:  Algo deu errado!
```

Verificação de Triângulo Equilátero

Código que verifica se os lados formam um triângulo equilátero

```
def verifica_equilatero(triangulo):
    if triangulo[0] == triangulo[1] == triangulo[2]:
        return True
    else:
        raise ValueError("Não é um triângulo equilátero!")

try:
    lados = [5, 5, 5]
    if verifica_equilatero(lados):
        print("É um triângulo equilátero!")
except ValueError as e:
    print(f"Erro: {e}")
```

Exceções - Criando exceções personalizadas

Quando usar?

Usada quando precisamos definir nossos próprios tipos de erro para tornar o código mais legível e facilitar o tratamento de erros específicos.

Passo a passo para criação

1. Criar uma classe que herda de `Exception`
2. Definir um construtor (`__init__`) para personalizar a exceção
3. Levantar a exceção (`raise`) no código
4. Capturar a exceção (`except`) e tratá-la

Exceções - Criando exceções personalizadas

1. Criar uma classe que herda de Exception

Cada exceção personalizada deve ser uma classe que herda da classe Exception. Isso garante que ela tenha o comportamento de uma exceção normal.

Exemplo Básico

```
class SaldoInsuficienteError(Exception):
    """Exceção para indicar que o saldo da conta é insuficiente."""
    pass
```

`SaldoInsuficienteError` já funciona como uma exceção, mas ainda não tem uma mensagem personalizada.

Exceções - Criando exceções personalizadas

2. Definir um construtor (`__init__`) para personalizar a exceção

Podemos adicionar um construtor para aceitar parâmetros e definir uma mensagem de erro.

Exemplo com construtor personalizado

```
class SaldoInsuficienteError(Exception):
    """Exceção para saldo insuficiente."""
    def __init__(self, saldo, valor,
                 mensagem="Saldo insuficiente para a operação."):
        self.saldo = saldo
        self.valor = valor
        self.mensagem = f"{mensagem} Saldo atual: R${saldo:.2f}, valor
solicitado: R${valor:.2f}."
        super().__init__(self.mensagem)
```

Exceções - Criando exceções personalizadas

3. Levantar a exceção (`raise`) no código

Agora podemos usar `raise` para lançar essa exceção em uma função.

Exemplo de uso com `raise`

```
def sacar(saldo, valor):
    if valor > saldo:
        raise SaldoInsuficienteError(saldo, valor)
    saldo -= valor
    return saldo
```

Exceções - Criando exceções personalizadas

4. Capturar a exceção (except) e tratá-la no código

Como qualquer outra exceção, podemos capturá-la com `try-except`

Exemplo completo de tratamento

```
try:
    saldo_atual = 100.0
    novo_saldo = sacar(saldo_atual, 200.0)
    print(f"Saque realizado! Novo saldo: R${novo_saldo:.2f}")
except SaldoInsuficienteError as e:
    print(f"Erro: {e}")
```

Criando Exceções Personalizadas

```
class SaldoInsuficienteError(Exception):
    def __init__(self, saldo, valor, mensagem="Saldo insuficiente para a
    operação."):
        self.saldo = saldo
        self.valor = valor
        self.mensagem = f"{mensagem} Saldo atual: R${saldo:.2f},
        valor solicitado: R${valor:.2f}."
        super().__init__(self.mensagem)
def sacar(saldo, valor):
    if valor > saldo:
        raise SaldoInsuficienteError(saldo, valor)
    saldo -= valor
    return saldo
try:
    saldo_atual = 100.0
    novo_saldo = sacar(saldo_atual, 20.0)
    print(f"Saque realizado! Novo saldo: R${novo_saldo:.2f}")
except SaldoInsuficienteError as e:
```

Dicas e melhores práticas

- Não use *except Exception* de forma genérica.
- Não use *except* sem nenhuma exceção.
- Para pegar uma exceção e tratar, precisa que o código que você queira testar esteja dentro do *try*
- Não sabe qual exceção pegar? Leia a documentação do python
  - Google é seu amigo, procure por *"python exception list"*
  - ou vá na URL: `https://docs.python.org/3/library/exceptions.html`