Erros, logs e debug

Fatec 2025

Erros

- Quando o código para de funcionar inesperadamente, antes de concluir todas as tarefas que deveria fazer
- Pode interromper outros processos
- Tratamento de erro garante a robustez do código

Interpretando mensagens de erro

- SEMPRE LEIA a mensagem de erro!
- Erros de sintaxe
  - Quando você escreve um código que não está completamente correto na linguagem
  - Ex: esquecer de fechar parênteses ou omitir `def` na definição de uma função
- Exceções
  - Incluem todos os outros erros, como uma entrada ausente para uma função ou tentativa de buscar uma chave inexistente em um dicionário

Traceback

- Mostram todas as funções chamadas que levam ao erro
- As vezes contém detalhes internos
- Estratégia: começar lendo pelo final
- Acima, uma seta aponta para a linha que causou o erro
  - Muitas vezes o erro está na linha anterior

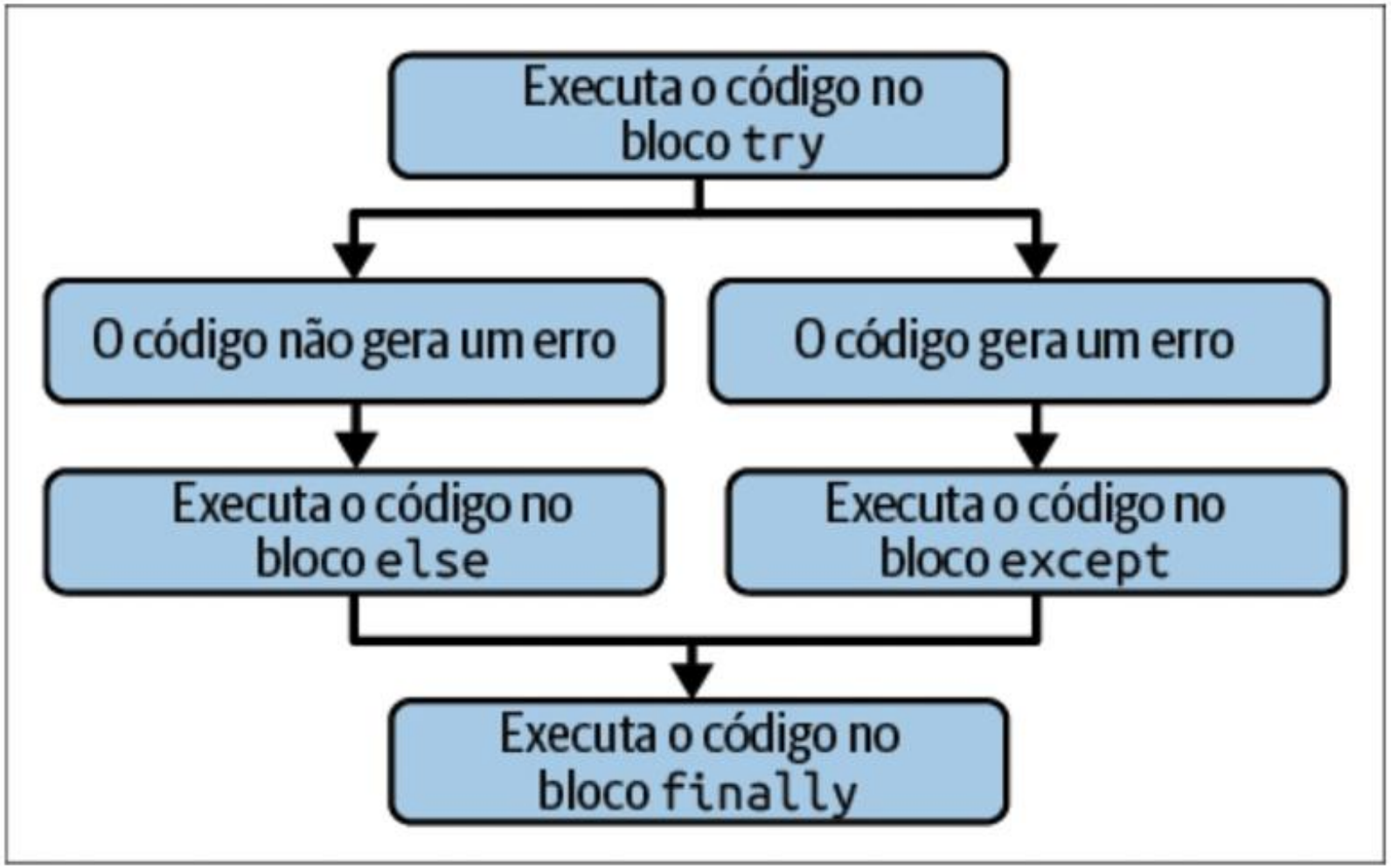
Tratamento de erro

- Muitas vezes não desejamos que o código pare ao encontrar um erro
- Palavras chave : try e except na forma de 2 blocos:

```
try:
   # algum código que você deseja executar
except KeyError:
   #o que você quer que aconteça se um erro desse tipo
   for lançado
```

https://docs.python.org/3/tutorial/errors.html

Tratamento de erro



Erro: pandas.errors.EmptyDataError

Código:

```
import pandas as pd
df = pd.read_csv('arquivo_vazio.csv')
```

Explicação:
O erro ocorre quando o arquivo CSV está vazio. O pandas espera ao menos uma linha de dados.

Erro: pandas.errors.ParserError

Código:

```
import pandas as pd
df = pd.read_csv('arquivo_mal_formatado.csv')
```

Explicação:
Erro ao tentar ler arquivos CSV mal formatados, com delimitadores inconsistentes ou campos ausentes.

Erro: Erro em groupby com NaN

Código:

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2, None]})
df.groupby('A').sum()
```

Explicação:
Problemas podem ocorrer se a coluna de agrupamento tiver valores nulos. Pode gerar grupos inesperados.

Erro: ValueError no train_test_split

Código:

```
from sklearn.model_selection import train_test_split
X = [1, 2, 3]
y = [0, 1]
train_test_split(X, y)
```

Explicação:
Ocorre quando os arrays de entrada (X) e rótulo (y) têm tamanhos diferentes.

Erro: KeyError com loc

Código:

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2]})
df.loc[3]
```

Explicação:
Erro gerado ao tentar acessar uma linha com índice que não existe no DataFrame.

Erro: MemoryError com NumPy

Código:

```
import numpy as np
grande_array = np.ones((10**8, 10**8))
```

Explicação:
O erro ocorre ao tentar criar arrays muito grandes, exigindo mais memória do que disponível.

Erro: SettingWithCopyWarning

```
Código:
df = df[df['A'] > 1]
df['A'] = 10
```

Explicação:
Ocorre ao modificar subconjuntos de DataFrame sem `.copy()`. Pode levar a alterações inesperadas.

Erro: numpy.linalg.LinAlgError

```
Código:
from numpy.linalg import inv
inv([[1, 2], [2, 4]])
```

Explicação:
O erro ocorre quando tentamos inverter uma matriz que não é invertível (determinante = 0).

Erro: ValueError no matplotlib

Código:

```
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [1, 2]
plt.plot(x, y)
```

Explicação:
O erro acontece quando os arrays `x` e `y` possuem tamanhos diferentes.

Erro: KeyError em coluna inexistente

Código:

```
df['nova'] = df['coluna_inexistente'] + 1
```

Explicação:
O erro aparece ao tentar acessar uma coluna que não existe no DataFrame.

Gerando erros com Raise

- Palavra-chave que gera um erro, mesmo que ele não seja lançado normalmente
- Permite personalizar o erro

```
from scipy.stats import linregress

def fit_trendline(year_timestamp, data):
  if not year_timestamp or not data:
    raise ValueError("Timestamps e data não podem ser listas vazias.")

  result = linregress(year_timestamp, data)
  slope = round(result.slope, 3)
  r_squared = round(result.rvalue**2, 3)
  return slope, r_squared
```

```
timestamp = []
data = [18.36,18.36,17.91]
fit_trendline(timestamp,data)
```

- É possível fornecer uma mensagem de erro personalizada
- É possível executar essa função com uma lista vazia

erros personalizados

- A mensagem de erro deve ser informativa para a pessoa que lê
- Contribuir para a legibilidade do código
- Para projetos grandes, nomear erros de forma personalizada facilita a identificação da origem do erro pelos usuários do código

•Criação de **uma classe de erro genérica** (NewException)
•**Herança** para criar um erro mais específico (InputInvalidoError)
•Uso do **construtor com super()** para manter a hierarquia da exceção
•Como **lançar (raise) e capturar (except)** a nova exceção

```python
# Classe base de exceções
class NewException(Exception):
    def __init__(self, message="Erro personalizado na aplicação."):
        self.message = message
        super().__init__(self.message)
# Exceção específica que herda de NewException
class InputInvalidoError(NewException):
    def __init__(self, entrada, message="Entrada inválida fornecida."):
        self.entrada = entrada
        super().__init__(f"{message} Valor recebido: {entrada}")
# Uso em uma função
def processar_entrada(valor):
    if not isinstance(valor, int):
        raise InputInvalidoError(valor)
    print(f"Valor processado com sucesso: {valor}")
# Testando com entrada incorreta
try:
    processar_entrada("abc")
except InputInvalidoError as e:
    print("Erro capturado:", e)
```

Registro em logs

- Método que registra o que seu código fez em um arquivo separado enquanto estava sendo executado
- o código pode comunicar a outras pessoas o que está acontecendo
- Ajuda na compreensão
- Torna o código mais legível e robusto, oferencendo um registro de falhas
- Preserva o histórico da ações do código
- Em produção, não desejamos interromper a execução do código para idenficar o que está acontecendo
- Evidencia que o código fez o que deveria fazer

O que registrar em um log

- Mensagens informando o início ou término de uma tarefa
- Mensagem de erro para identificar problemas no sistema
- Quais funções chamaram outras funções
- Entradas e saídas de uma função
- O caminho do arquivo em que alguns dados foram salvos
- Cudaido: uma mensagem “salvando arquivo” não serve para nada!

Módulo Loggin

- A biblioteca se encarrega de salvar todas as mensagens que registramos em logs em arquivo separado
- Níveis de gravidade:

padrão

Nível	Quando é usado
DEBUG	Informações detalhadas, normalmente de interesse somente no diagnóstico de problemas.
INFO	Confirmação de que tudo está funcionando conforme o esperado.
WARNING	Uma indicação de que algo inesperado ocorreu ou um indicativo de algum problema em um futuro próximo (por exemplo, "disk space low [pouco espaço em disco]"). O software ainda está funcionando conforme o esperado.
ERROR	Devido a um problema mais grave, o software não conseguiu executar alguma função.
CRITICAL	Um erro grave, indicando que o próprio programa pode não conseguir continuar em execução.

https://docs.python.org/3/howto/logging.html

Implementando log

```python
import logging
from scipy.stats import linregress
import numpy as np
import os

# Definir caminho absoluto para o log
log_path = "/content/log_trendline.log"
# Apagar arquivo de log anterior (opcional para testes)
if os.path.exists(log_path):
    os.remove(log_path)
    print(f"Arquivo de log apagado: {log_path}")
# Criar logger totalmente novo e isolado
logger = logging.getLogger("trendline_logger")
logger.setLevel(logging.INFO)
# Limpar handlers anteriores para evitar conflito no Colab
if logger.hasHandlers():
    logger.handlers.clear()
# Criar handler de arquivo
file_handler = logging.FileHandler(log_path, mode='w')  # 'w' sobrescreve
file_handler.setLevel(logging.INFO)
# Definir formatação
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)
# Adicionar handler ao logger
logger.addHandler(file_handler)
```

Implementando log

```python
# Função de regressão
def fit_trendline(year_timestamp, data):
    logger.info("Executando a função fit_trendline")
    result = linregress(year_timestamp, data)
    slope = round(result.slope, 3)
    r_squared = round(result.rvalue**2, 3)
    logger.info(f"Análise completa. Slope da função trendline é {slope}")
    return slope, r_squared

# Dados de exemplo
anos = np.array([2015, 2016, 2017, 2018, 2019, 2020])
valores = np.array([100, 120, 130, 150, 170, 200])

# Executar função
slope, r2 = fit_trendline(anos, valores)
print("Coeficiente angular:", slope)
print("Coeficiente de determinação R²:", r2)

# Forçar gravação no disco
for handler in logger.handlers:
    handler.flush()
    print(f"Arquivo de log gravado: {log_path}")

# Ler o log
print("\nConteúdo do log:")
with open(log_path, "r") as f:
    print(f.read())
```

Log viewers – visualizadores de log

KIBANA -
HTTPS://WWW.ELASTIC.CO/KIBANA

PROMETHEUS -
HTTPS://PROMETHEUS.IO

Depuração

- Identificação e remoção de bugs
- Bug = código gera erros inusitados ou resultados não esperados
- Dica: fazer experimentos!
- Testes indicarão se a mudança feita gera o resultado esperado
- Dica: salvar uma cópia do código que causou o erro
- Compartilhe e discuta o erro e tome um café!

Ferramentas de depuração

- Vs Code
  - https://learn.microsoft.com/en-us/visualstudio/python/debugging-python-in-visual-studio?view=vs-2022
- PyCharm
  - https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#where-is-the-problem
- Python Debugger
  - https://docs.python.org/3/library/pdb.html

Depuração

```python
def calcular_media(notas):
    soma = 0
    for nota in notas:
        soma += nota
        print(f"DEBUG: Soma atual = {soma} depois de adicionar {nota}") # Ponto de depuração
    media = soma / len(notas)
    print(f"DEBUG: Média final = {media}") # Ponto de depuração
    return media

notas_alunos = [7, 8, 5, 9, 6]
media_final = calcular_media(notas_alunos)
print(f"A média das notas é: {media_final}")

# Exemplo com um erro sutil: uma nota que não deveria ser adicionada
notas_com_erro = [7, 8, 5, "dez", 6] # "dez" causará um TypeError
try:
    media_com_erro = calcular_media(notas_com_erro)
    print(f"A média das notas com erro é: {media_com_erro}")
except TypeError as e:
    print(f"Erro capturado: {e}. Isso aconteceu porque tentamos somar um tipo inválido.")
```

Depuração

```python
#!pip install ipdb # Instale o pacote ipdb se ainda não estiver instalado
import ipdb

def calcular_desconto(valor_original, percentual_desconto):
    #ipdb.set_trace() # Descomente para ativar o depurador aqui
    valor_com_desconto = valor_original * (1 - percentual_desconto / 100)
    return valor_com_desconto

preco_produto = 100
desconto_aplicado = 10

# Ativando o depurador antes de uma linha específica
# Para ativar o depurador em qualquer ponto, use breakpoint() (Python 3.7+) ou ipdb.set_trace()
# No Colab, você também pode usar %debug para entrar no depurador após um erro.
# Ou %pdb on para ativar o depurador automaticamente em caso de erro.

# Vamos forçar um erro para demonstrar %debug
def dividir(a, b):
    #ipdb.set_trace()
    return a / b

try:
    resultado = dividir(10, 0) # Isso causará um ZeroDivisionError
    print(resultado)
except ZeroDivisionError:
    print("Ocorreu um erro de divisão por zero. Entrando no depurador para investigar.")
    # Digite %debug na célula após o erro para ativar o depurador neste ponto
    # Ou rode a célula com o erro, e depois rode uma nova célula com %debug
    # Outra opção é usar o "magic command" %pdb on antes de rodar o código.
```

Depurador Colab

- **Como usar o depurador no Colab (após um erro):**
- **Execute a célula** que contém o erro (ex: dividir(10, 0)).
- **Após o erro aparecer**, crie uma **nova célula** e digite %debug e execute-a.
- Você será levado para o prompt do depurador (ipdb>).
  - **Comandos úteis:**
    - n (next): Executa a próxima linha.
    - s (step): Entra em chamadas de função.
    - c (continue): Continua a execução até o próximo breakpoint ou fim do programa.
    - q (quit): Sai do depurador.
    - p <variavel>: Imprime o valor de uma variável (ex: p a).
    - l (list): Mostra as linhas de código ao redor.
    - w (where): Mostra o stack trace (onde você está na pilha de chamadas).
- **Para usar ipdb.set_trace() (breakpoint programático):**
  - Descomente a linha ipdb.set_trace() onde você quer que a execução pare.
  - Execute a célula.
  - Quando a execução atingir essa linha, o prompt ipdb> aparecerá na saída da célula. Você pode então usar os comandos do depurador.

Visualização do Depurador com Python Tutor

- Python Tutor é uma ferramenta excelente para visualizar a execução do código passo a passo
- útil para entender o fluxo e o estado das variáveis
- Vá para http://pythontutor.com/
  - Cole seu código Python
  - Clique em "Visualize Execution"

Exemplo para o Python tutor

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
numero = 4
resultado_fatorial = factorial(numero)
print(f"O fatorial de {numero} é {resultado_fatorial}")
```