TRATAMENTO DE EXCEÇÕES EM PROGRAMAÇÃO ORIENTADA A OBJETOS

IFBA – Campus Ilhéus

Curso: Back End com Python

Objetivo da Aula

- Melhorar a segurança e robustez do código.

Por que tratar exceções?

- Quando algo inesperado acontece:
  - Entrada inválida
  - Arquivo inexistente
  - Operação proibida (ex: divisão por zero)

- Se não tratarmos a exceção → Programa **para de funcionar**
- Se tratarmos → Programa continua **seguro e robusto**

Revisão rápida — try/except

```
try:
    # código que pode dar erro
except TipoDeErro:
    # o que fazer se der erro
```

Exceções Customizadas (personalizadas)

- Às vezes, o erro **faz sentido no contexto do sistema**.
- Exemplo: Se estamos em um banco, “saldo insuficiente” é um erro do domínio, não do Python. Podemos criar nossa própria exceção:

```
class SaldoInsuficienteError(Exception):
    def __init__(self, mensagem="Saldo insuficiente para realizar a operação!"):
        super().__init__(mensagem)
```

Exemplo prático com classe Conta

```
class ContaBancaria:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

    def sacar(self, valor):
        if valor > self.saldo:
            raise SaldoInsuficienteError("Saldo insuficiente para o saque!")
        self.saldo -= valor
        print(f"Saque de R$ {valor} realizado com sucesso.")
```

Tratamento com try/except dentro de classes

```
class ContaBancaria:
    # mesmo código anterior

    def depositar(self, valor):
        try:
            if valor <= 0:
                raise ValueError("Valor do depósito
deve ser positivo.")
            self.saldo += valor
            print("Depósito realizado!")
        except ValueError as erro:
            print("Erro:", erro)
```

Exemplo completo — uso da classe

```
conta = ContaBancaria("Karine", 100)

try:
    conta.sacar(150)
except SaldoInsuficienteError as e:
    print("Falha na operação:", e)

conta.depositar(-50)
```

Regras importantes

Palavra-chave	Para que serve
`try`	Parte do código onde pode ocorrer erro
`except`	Captura e trata erro
`raise`	Lança exceção manualmente
Exceções customizadas	Sinalizam erros do negócio

Exercício 1

- Crie um programa em Python que valide a idade de uma pessoa usando uma exceção personalizada.
  - Crie uma exceção personalizada.
  - Crie uma classe chamada IdadeInvalidaError, que deve herdar de Exception.
  - Ela deve ser disparada quando alguém tentar informar uma idade menor que 0.
  - A mensagem de erro deve ser algo como: "Idade negativa não é permitida!"

- Crie uma classe Pessoa com:
  - Um atributo nome, definido no momento da criação do objeto.
  - Um atributo idade, inicializado como 0.

Exercício 1 - continuação

- Crie uma função para definir idade:
  - Crie uma função chamada definir_idade(idade) que:
    - Recebe um número inteiro representando a idade.
    - Verifica se a idade é válida:
    - Se for menor que 0, deve lançar a exceção personalizada.
    - Se for válida (0 ou maior), exibir uma mensagem como: "Idade registrada com sucesso!“
    - Use try/except para capturar o erro e exibir a mensagem fornecida pela exceção.

Exercício 1 - continuação

- No programa principal:
  - Chame a função definir_idade duas vezes:
    - Uma vez com uma idade inválida (ex.: -5) → deve ocorrer a exceção tratada.
    - Uma vez com uma idade válida (ex.: 20).

Exercício 2

- **Crie uma exceção personalizada**
  - Implemente uma classe chamada **NotaInvalidaError**, herdando de Exception.
  - Essa exceção deve ser disparada sempre que alguém tentar informar uma nota fora do intervalo permitido.
  - O construtor da classe deve receber o valor da nota inválida e produzir uma mensagem no seguinte formato:
  - "Nota 'X' inválida! A nota deve estar entre 0 e 10."
  - onde **X** é o valor da nota informada.

Exercício 2 - Continuação

- **Crie uma classe chamada Aluno**
- A classe deve conter:
  - Um atributo **nome**, informado no momento da criação do objeto.
  - Um atributo **nota**, inicializado com o valor **0**.

Exercício 2 - Continuação

- **Crie o método definir_nota(self, n)**
- Esse método será responsável por registrar a nota do aluno. Ele deve:
  - Receber um valor numérico `n`, representando a nota.
  - No interior de um bloco `try`, verificar se a nota é válida:
    - Se **n < 0 ou n > 10**, deve ser lançada a exceção `NotaInvalidaError(n)`.
  - Se a nota for válida, atualizar o atributo `nota` do objeto e exibir a mensagem:
  - A nota X foi registrada com sucesso para NOME!
  - Caso ocorra a exceção personalizada, capturá-la dentro de um bloco `except` e exibir:
  - Erro ao definir nota: <mensagem da exceção>

Exercício 2 - Continuação

- No programa principal:
  - Crie um objeto da classe **Aluno**, por exemplo:
  - aluno = Aluno("Marcos")
  - Chame o método definir_nota duas vezes:
    - Primeiro com uma nota inválida, como **12**, para demonstrar o funcionamento da exceção.
    - Depois com uma nota válida, como **8**, para mostrar o registro bem-sucedido.
- O programa deve exibir mensagens adequadas para cada caso.