# Tratamento de Exceções

Alessandro Botelho Bovo

# Objetivo da aula

- Apresentar o conceito de exceção em Python.
- Mostrar como evitar que o programa seja interrompido com erros inesperados.
- Praticar o uso do `try` e `except` em exemplos simples.

# O que é uma exceção?

- Exceções são erros detectados durante a execução do programa.
- Sem tratamento, o programa é interrompido.
- Exemplo: `numero = int('abc')`
- Esse código gera a exceção:

```
ValueError: invalid literal for int() with base 10: 'abc'
```

# Estrutura básica do `try/except`

**Código *sem* tratamento de exceção:**

```
# bloco de código que pode gerar erro
numero = int(input('Digite um número inteiro: '))
print('Você digitou:', numero)
```

**Código *com* tratamento de exceção:**

```
try:
    # bloco de código que pode gerar erro
    numero = int(input('Digite um número inteiro: '))
    print('Você digitou:', numero)
except:
    # bloco executado se ocorreu um erro
    print('Erro: você não digitou um número válido.')
```

# Boas práticas

**Especifique o tipo de erro quando possível:**

```
try:
    # bloco de código que pode gerar erro
    numero = int(input('Digite um número inteiro: '))
    print('Você digitou:', numero)
except ValueError:
    # bloco executado se ocorreu um erro
    print('Erro: você não digitou um número válido.')
```

**Esse código utiliza a exceção `ValueError`**

**Evite usar apenas `except:` sem indicar o erro → pode esconder problemas não previstos.**

# Erros comuns

```
try:
    valor = int('dez')  # Conversão inválida de string para número
except ValueError:
    print('Erro: não foi possível converter para inteiro.')


try:
    x = 10 / 0          # Divisão por zero
except ZeroDivisionError:
    print('Erro: divisão por zero não é permitida.')


try:
    lista = [1, 2, 3]
    print(lista[5])     # Índice fora da faixa de uma lista
except IndexError:
    print('Erro: índice inexistente na lista.')
```

# Capturando diferentes tipos de exceções

- No Python, podemos tratar **cada tipo de exceção separadamente**, usando vários blocos `except`.

```
try:
    # Código que pode gerar diferentes tipos de erros
    ...

except TipoDeErro1:
    # O que fazer se ocorrer TipoDeErro1
    ...

except TipoDeErro2:
    # O que fazer se ocorrer TipoDeErro2
    ...

except:
    # "Plano B" para qualquer outro erro não previsto
    ...
```

# Exercício 1

- Crie um programa em Python que solicite ao usuário dois números. Tente realizar a divisão do primeiro pelo segundo. Trate os seguintes casos com `try` e `except`: 'entrada inválida' e 'divisão por zero'.

**Execução simulada 1 (valores válidos):**

```
Digite o primeiro número: 20
Digite o segundo número: 4
O resultado da divisão é: 5.0
```

**Execução simulada 2 (divisão por zero):**

```
Digite o primeiro número: 15
Digite o segundo número: 0
Erro: não é possível dividir por zero.
```

**Execução simulada 3 (entrada inválida):**

```
Digite o primeiro número: dez
Erro: você digitou um valor que não é número inteiro.
```

# Solução do Exercício 1

```
try:

    a = int(input('Digite o primeiro número: '))
    b = int(input('Digite o segundo número: '))
    resultado = a / b
    print(f'O resultado da divisão é: {resultado}')

except ValueError:
    print('Erro: você digitou um valor que não é número inteiro.')

except ZeroDivisionError:
    print('Erro: não é possível dividir por zero.')
```

# Uso do `else` no tratamento de exceções

```
try:
    a = int(input('Digite o primeiro número: '))
    b = int(input('Digite o segundo número: '))
    resultado = a / b
except ValueError:
    print('Erro: você digitou um valor que não é número inteiro.')
except ZeroDivisionError:
    print('Erro: não é possível dividir por zero.')
else:
    print(f'O resultado da divisão é: {resultado}')
```

- O **tratamento de erros** está separado.
- O `else` mostra: *"essa parte só roda se não houve nenhum problema"*.
- O código fica mais **organizado e legível**, principalmente em programas grandes.

# Uso do `finally` no tratamento de exceções

- O bloco `finally` é usado junto com `try/except`.
- Ele é executado **sempre**, independentemente de ter ocorrido erro ou não.
- Serve principalmente para **finalizar recursos**: fechar arquivos, encerrar conexões, liberar memória, etc.
- Estrutura:

```
try:
    # código que pode gerar erro
except TipoDeErro:
    # tratamento do erro
finally:
    # código que SEMPRE será executado
```

# Exemplo simples de uso do `finally`

```
try:
    numero = int(input('Digite um número inteiro: '))
    x = 10 / numero
except ZeroDivisionError:
    print("Erro: divisão por zero.")
finally:
    print("Fim do programa (sempre executado).")
```

**Esse comando sempre vai ser executado**

**Exemplo de saída:**

```
Erro: divisão por zero.
Fim do programa (sempre executado).
```

# Sem e com o `finally`

```python
def primeira_linha_com_utfpr_sem_finally():
    f = open('dados.txt', 'r')
    for linha in f:
        if 'UTFPR' in linha:
            return linha   # ⚠ sai antes do f.close()
    f.close()
    return None
```

```python
def primeira_linha_com_utfpr_com_finally():
    f = None
    try:
        f = open('dados.txt', 'r')
        for linha in f:
            if 'UTFPR' in linha:
                return linha  # ok: o finally ainda executa
        return None
    except FileNotFoundError:
        print('Erro: arquivo "dados.txt" não encontrado.')
        return None
    finally:
        if f is not None:
            f.close()
```

# Tratamento de Exceções em algumas linguagens

| Linguagem | Estrutura disponível |
| --- | --- |
| Python | `try / except / else / finally` |
| Java | `try / catch / finally` |
| C++ | `try / catch` |
| JavaScript | `try / catch / finally` |
| C# | `try / catch / finally` |
| C | Não possui exceções |

# Contato

Alessandro Botelho Bovo

alessandrobovo@utfpr.edu.br