Lógica de Programação
Manipulando Funções com Python

Prof.: Caio Malheiros

caio.duarte@sp.senai.br

Plano de aula

- Try e Except
- Manipulando arquivos
- Exemplos
- Atividades

Tratamento de exceções

O tratamento de exceções **permite lidar com erros** de forma controlada, **evitando** que o programa seja interrompido abruptamente.

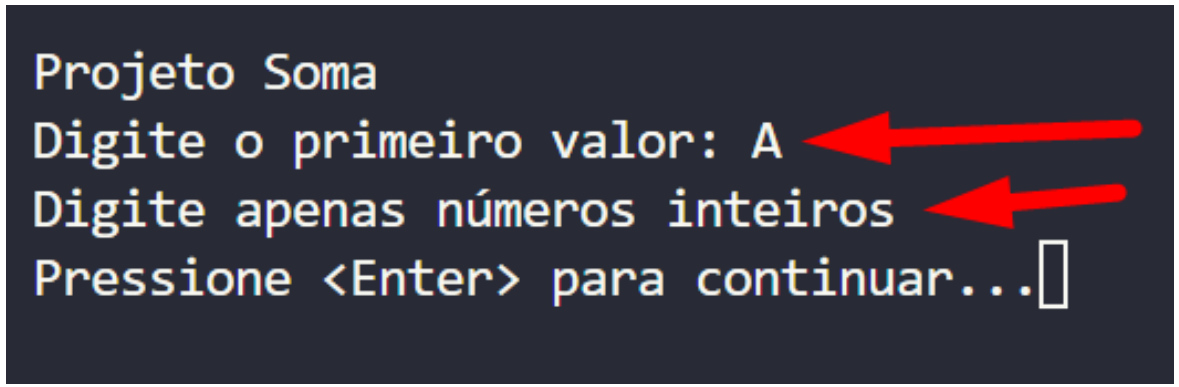
```
valor01 = int(input("Digite o primeiro valor: "))
```

A estrutura **try** e **except** possibilita capturar e tratar esses erros, permitindo a criação de programas mais robustos e confiáveis..

```
Projeto Soma
Digite o primeiro valor: a
  File "c:\Users\Caio Malheiros\Documents\curso-python\exemplo-soma.py", line 9, in <module>
  File "c:\Users\Caio Malheiros\Documents\curso-python\exemplo-soma.py", line 9, in <module>
    valor01 = int(input("Digite o primeiro valor: "))
              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'a'
```

Tratamento de exceções

```python
import os

os.system("cls")

#1 Passo - Entrada de dados
print("Projeto Soma")

try:
    valor01 = int(input("Digite o primeiro valor: "))
    valor02 = int(input("Digite o segundo valor: "))

    #2 Passo - Processamento
    total = valor01 + valor02

    #3 Passo - Saída
    print(f"O resultado é: {total}")

except:
    print("Digite apenas números inteiros")


input("Pressione <Enter> para continuar...")
```



Conceitos básicos

O que são exceções?

Exceções: São erros que ocorrem durante a execução do programa, como divisão por zero, acesso a índice inexistente, conversões inválidas, etc.

Objetivo: Permitir que o programa trate **esses erros de maneira apropriada**, exibindo mensagens amigáveis ou tomando ações corretivas.

Por que utilizar try e except?

Prevenção de falhas: Evita que erros não tratados interrompam a execução do programa.

Melhoria da experiência do usuário: Permite fornecer feedback claro sobre o que deu errado.

Manutenção do fluxo de execução: Mesmo ocorrendo um erro, o programa pode continuar a execução ou realizar operações alternativas.

Estrutura Básica: try e except

Bloco try: Contém o código que pode gerar uma exceção.

Bloco except: É executado se ocorrer uma exceção no bloco try.

```python
#exemplo de try simples
try:
    print(10/0)
except:
    print("Erro ao tentar dividir por zero")

print("Fim do programa")
```

Por que utilizar try e except?

Prevenção de falhas: Evita que erros não tratados interrompam a execução do programa.

Melhoria da experiência do usuário: Permite fornecer feedback claro sobre o que deu errado.

Manutenção do fluxo de execução: Mesmo ocorrendo um erro, o programa pode continuar a execução ou realizar operações alternativas.

Estrutura Básica: try e except

Bloco try: Contém o código que pode gerar uma exceção.

Bloco except: É executado se ocorrer uma exceção no bloco try.

```python
#exemplo de try simples
try:
    print(10/0)
except:
    print("Erro ao tentar dividir por zero")

print("Fim do programa")
```

Por que utilizar try e except?

Você pode estruturar o tratamento de exceções de diversas formas, permitindo capturar erros de maneira específica ou genérica. Eis um resumo das possibilidades e sintaxes:

Exceção Específica:

Você pode indicar uma exceção específica, como **ValueError**, **TypeError**, etc.

```
try:
    # Bloco de código que pode gerar exceções
except ExceptionType:
    # Bloco para tratar a exceção específica "ExceptionType"
```

```
try:
    # código
except ValueError as ve:
    # Aqui, "ve" contém a exceção lançada.
```

Uso de múltiplos blocos

Você pode ter vários blocos **except** para tratar tipos diferentes de exceções:

```
try:
    # código que pode gerar exceções
except ValueError:
    # tratamento específico para ValueError
except TypeError:
    # tratamento específico para TypeError
except Exception as e:
    # captura qualquer outra exceção derivada de Exception
```

Ou uso de **else** e **finally**

```
try:
    # código sem exceção
except Exception:
    # tratamento de erro
else:
    # executado se não ocorrer nenhuma exceção
finally:
    # sempre executad, mesmo se houver uma exceção
```

Tabela com alguns Except

Exceção	Descrição
TypeError	Lançada quando uma operação ou função é aplicada a um objeto de tipo inadequado.
ValueError	Ocorre quando um valor possui tipo correto, mas é inapropriado para a operação.
KeyError	Gerada quando uma chave não existe em um dicionário.
BaseException	Classe base de todas as exceções.
Exception	Base para a maioria das exceções comuns; é o tipo que normalmente você captura.
ArithmeticError	Classe base para erros aritméticos, como divisão por zero ou overflow.
ZeroDivisionError	Ocorre quando há tentativa de divisão por zero.
FileNotFoundError	Erro gerado ao tentar acessar um arquivo ou diretório inexistente.
RuntimeError	Erro genérico que não se encaixa em outras categorias.
NotImplementedError	Usada para sinalizar que uma funcionalidade ou método ainda não foi implementado.
SyntaxError	Indica erros de sintaxe no código.
IndentationError	Subtipo de SyntaxError; ocorre quando há problemas na indentação.

Excercicios

Aplique Try e Except nas atividades realizadas em sala de aula:

- Calculadora
- Cálculo de IMC
- Conversor de temperatura
- Conversor de moeda
- Tabuada

Manipulando Arquivos

Usando a Função Built-in open()

A função **open()** é a maneira básica de abrir arquivos em Python.

Ela retorna um objeto que representa o arquivo e permite realizar

operações como leitura, escrita e anexação de dados.

```
arquivo = open("caminho_do_arquivo.txt", "modo")
```

Manipulando Arquivos

“caminho_do_arquivo.txt”: Indica o caminho e o nome do arquivo a ser aberto

“Modo”: Especifica a forma como o arquivo será manipulado:

“r”: Modo leitura (padrão). Gera erro se o arquivo não existir

“w”: Modo escrita. Cria um novo arquivo ou sobrescreve o existente.

“a”: Modo anexação. Adiciona dados ao final do arquivo existente.

“r+”: Modo leitura e escrita. O arquivo deve existir

Usando a função Built-in open()

A sintaxe geral da função **open()** é:

```
arquivo = open("caminho_do_arquivo.txt", "modo")


# Podendo ser:

arquivo = open("caminho_do_arquivo.txt", "r")
# ou
arquivo = open("caminho_do_arquivo.txt", "w")
# ou
arquivo = open("caminho_do_arquivo.txt", "a")
```

Exemplo

Contexto: Vamos desenvolver um programa que solicitará ao usuário a inserção de dados para o cadastro de pessoas (nome e e-mail). Esses dados serão armazenados em um arquivo de texto, denominado **"pessoa.txt".**

A cada execução do programa, um novo registro será adicionado ao arquivo.

Caso o arquivo não exista, ele será criado automaticamente.

O arquivo de texto será criado ao lado do script Python.

Exemplo

arquivo = open("pessoa.txt", "a") abre o arquivo "pessoa.txt“ no modo de anexação ("a"), o que significa que os novos dados serão adicionados ao final do arquivo. Se o arquivo não existir, ele será criado automaticamente.

arquivo.write(nome + " | " + email + "\n") escreve no arquivo uma linha contendo o nome e o e-mail separados pelo caractere “ | “, finalizando com uma quebra de linha para que Cada novo registro comece em uma linha separada.

arquivo.close() fecha o arquivo, garantindo que todos os dados Escritos sejam salvos e liberando os recursos utilizados Durante a operação.

```
nome = input("Digite o nome: ")
email = input("Digite o e-mail: ")

arquivo = open("pessoa.txt", "a")
arquivo.write(nome + " | " + email + "\n")
arquivo.close()
```

Exemplo

Esquecimento de fechar o arquivo utilizando o método **close()** pode causar os seguintes problemas:

Perda de dados: Os dados podem ficar retidos no buffer e não serem gravados de forma definitiva no arquivo.

Uso desnecessário de recursos: O arquivo permanece aberto, consumindo recursos do sistema que poderiam ser liberados.

Acesso restrito: Outros programas ou mesmo novas operações dentro do seu programa podem ter dificuldade em acessar ou modificar o arquivo, pois ele pode permanecer bloqueado.

Possível corrupção do arquivo: Em casos extremos, se o programa for encerrado abruptamente, o arquivo pode ficar com informações incompletas ou corrompidas.

Para melhorar nosso script, vamos utilizar um comando reservado no Python que é **with**.

Exemplo

O comando with simplifica o gerenciamento de arquivos ao criar um "contexto“ para as operações com o arquivo.

Quando o bloco do with termina, o arquivo é fechado automaticamente, mesmo que ocorram erros durante a execução.

Reduz a quantidade de código e a possibilidade de erros relacionados ao gerenciamento de recursos.

```
nome = input("Digite o nome: ")
email = input("Digite o e-mail: ")

with open("pessoa.txt", "a") as arquivo:
    arquivo.write(nome + " | " + email + "\n")
```

Exemplo

O comando with simplifica o gerenciamento de arquivos ao criar um "contexto“ para as operações com o arquivo.

Quando o bloco do with termina, o arquivo é fechado automaticamente, mesmo que ocorram erros durante a execução.

Reduz a quantidade de código e a possibilidade de erros relacionados ao gerenciamento de recursos.

```
nome = input("Digite o nome: ")
email = input("Digite o e-mail: ")

with open("pessoa.txt", "a") as arquivo:
    arquivo.write(nome + " | " + email + "\n")
```

Código completo

```python
import os

os.system("cls")

#1 Passo - Entrada de dados
print("Manipulação de arquivos")

arquivo = open("arquivo.txt", "w")

try:
   nome = input("Digite o seu nome: ")
   email = input("Digite o seu e-mail: ")

   with open("arquivo.txt", "w") as arquivo:
       arquivo.write(f"Nome: {nome}\n")
       arquivo.write(f"E-mail: {email}\n")
       arquivo.close()

except:
   print("Erro ao manipular o arquivo")

input("Pressione <Enter> para continuar...")
```

Exemplo

1 - Aplique manipulação de arquivos nas atividades realizadas em sala de aula, armazenando os dados trabalhados:

- **Sistema de Avaliação de Desempenho Escolar**
- **Monitoramento de Saúde com Cálculo de IMC**

2 – Desenvolva um programa que solicite ao usuário os dados de um produto (nome, valor e quantidade) e armazene-os em um arquivo de texto chamado "produtos.txt".

Dúvidas?
Ótimo dia para todos!