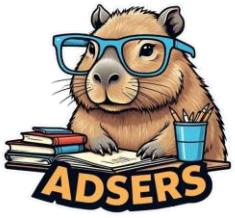
Programação Python - Nível Avançado



Manipulação Básica de Arquivos

A manipulação de arquivos é uma habilidade essencial em Python, permitindo que programas leiam dados de arquivos externos e salvem resultados para uso posterior. Python oferece funções simples e intuitivas para trabalhar com arquivos.

realizar operações (leitura/escrita) e fechar o arquivo. A função principal para isso é open() , que recebe como parâmetros o nome do arquivo e o modo de abertura.

Modos de abertura comuns:

- 'r' : Leitura (padrão)
- 'w' : Escrita (sobrescreve o conteúdo existente)
- 'a' : Adição (append - adiciona ao final do arquivo)
- 'b' : Modo binário (usado junto com outros modos, ex: 'rb')
- 't' : Modo texto (padrão)

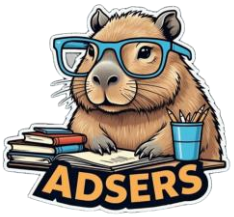
Exemplo de leitura em arquivo:

```
 # Abrindo um arquivo para leitura
arquivo = open('dados.txt', 'r')
conteudo = arquivo.read()  # Lê todo o conteúdo
arquivo.close()  # Fecha o arquivo
```

Exemplo de escrita em arquivo:

```
 # Abrindo um arquivo para escrita
arquivo = open('saida.txt', 'w')
arquivo.write('Olá, mundo!')
arquivo.close()
```

A abordagem recomendada para manipulação de arquivos é usar o gerenciador de contexto with, que garante que o arquivo seja fechado corretamente, mesmo se ocorrerem erros




```
# Usando with para leitura
with open('dados.txt', 'r') as arquivo:
    conteudo = arquivo.read()
    # O arquivo é fechado automaticamente ao sair do bloco with
```

Métodos úteis para leitura:

- read() : Lê todo o conteúdo do arquivo
- readline() : Lê uma linha do arquivo
- readlines() : Lê todas as linhas e retorna uma lista

Lidando com Erros: Exceções

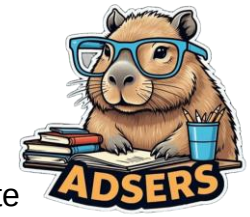
Exceções em Python são eventos que ocorrem durante a execução de um programa e interrompem o fluxo normal de instruções. Elas são usadas para lidar com erros e situações inesperadas, permitindo que o programa continue funcionando mesmo quando problemas ocorrem.

O tratamento de exceções em Python é feito usando bl`try` `except` `else` `finally`:

```
try:
    # Código que pode gerar uma exceção
    resultado = 10 / 0
except ZeroDivisionError:
    # Código executado se ocorrer uma divisão por zero
    print("Erro: Divisão por zero!")
except Exception as e:
    # Código executado para outras exceções
    print(f"Ocorreu um erro: {e}")
else:
    # Código executado se nenhuma exceção ocorrer
    print("Operação realizada com sucesso!")
finally:
    # Código sempre executado, independentemente de exceções
    print("Finalizando operação...")
```

Exceções comuns em Python:

- *ZeroDivisionError* : Ocorre ao dividir por zero
- TypeError : Ocorre quando uma operação é aplicada a um objeto de tipo inadequado
- ValueError : Ocorre quando uma função recebe um argumento com o tipo correto,



mas valor inadequado

- FileNotFoundError : Ocorre quando tenta-se abrir um arquivo que não existe
- IndexError : Ocorre ao tentar acessar um índice inválido em uma sequência
- . KeyError : Ocorre ao tentar acessar uma chave inexistente em um dicionário

Também é possível criar exceções personalizadas, criando classes que herdam de `Exception`:

```
class MeuErroPersonalizado(Exception):
    def __init__(self, mensagem):
        self.mensagem = mensagem
        super().__init__(self.mensagem)

# Usando a exceção personalizada
try:
    if condicao_de_erro:
        raise MeuErroPersonalizado("Ocorreu um erro
específico!")
except MeuErroPersonalizado as e:
    print(e)
```

Introdução à Programação Orientada a Objetos (POO)

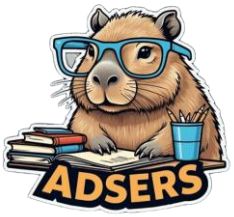
A Programação Orientada a Objetos (POO) é um paradigma de programação baseado no conceito de "objetos", que podem conter dados na forma de atributos e código na forma de métodos. Python é uma linguagem que suporta totalmente a POO, permitindo a criação de programas modulares, reutilizáveis e mais fáceis de manter.

Conceitos fundamentais da POO:

1. **Classes e Objetos:**

- **Classe:** É um modelo ou "molde" que define atributos e métodos comuns a todos os objetos desse tipo.
- **Objeto:** É uma instância de uma classe, representando uma entidade específica com seus próprios valores de atributos.

```python class Cachorro: # Método inicializador (construtor) def init(self, nome, raca): self.nome = nome # Atributo self.raca = raca # Atributo
```




```
    # Método
    def latir(self):
        return f"{self.nome} está latindo!"
```

Criando objetos (instâncias) da classe rex = Cachorro("Rex", "Pastor Alemão") print(rex.nome) # Acessando atributo print(rex.latir()) # Chamando método ```

1. **Encapsulamento:** Oculta os detalhes internos de implementação e expõe apenas o necessário. ```python class ContaBancaria: def init(self, saldo): self.___saldo = saldo # Atributo privado

   def depositar(self, valor): self.__saldo += valor

   def sacar(self, valor): if valor <= self._saldo: self.__saldo -= valor return True return False

   def consultar_saldo(self): return self.__saldo ```

2. **Herança**: Permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse). ```python class Animal: def init(self, nome): self.nome = nome

   def fazer_som(self): pass

   class Gato(Animal): # Gato herda de Animal def fazer_som(self): return "Miau!" ```

1. **Polimorfismo:** Permite que objetos de diferentes classes sejam tratados como objetos de uma classe comum. ```python def emitir_som(animal): return animal.fazer_som()

   gato = Gato("Felix") cachorro = Cachorro("Rex", "Labrador")

   print(emitir_som(gato)) # "Miau!" print(emitir_som(cachorro)) # "Rex está latindo!" ```

Referência

DataCamp. (2024). *Exceção e tratamento de erros em Python*. Disponível em: https://www.datacamp.com/pt/tutorial/exception-handling-python

Python Software Foundation. (2024). *8. Erros e exceções — Documentação Python 3.13.3*. Disponível em: https://docs.python.org/pt-br/3/tutorial/errors.html

Rocketseat. (2024). *Tratamento de Erros em Python na prática*. Disponível em: https://blog.rocketseat.com.br/tratamento-de-erros-em-python-na-pratica/

Mariano, D. (2023). Manipulando arquivos. Disponível em: https://diegomariano.com/manipulando-arquivos/

DIO. (2023). Um Guia Para Devs: Manipulação De Arquivos.txt. Disponível em: https://www.dio.me/articles/um-guia-paradevs-manipulacao-de-arquivostxt

FreeCodeCamp. (2022). Como escrever em um arquivo em Python – open, read, append e outras funções de manipulação explicadas. Disponível em: https://www.freecodecamp.org/portuguese/news/como-escrever-emum-arquivo-em-python-open-read-append-e-outras-funcoes-de-manipulacaoexplicadas/