Revisão de Python: Arquivos

> Arquivos são usados para armazenar dados em um dispositivo de armazenamento permanente, como um disco rígido.

- Python possui várias funções para criar, ler, atualizar e excluir arquivos diretamente no sistema de arquivos.
- Os arquivos podem ser de texto ou binários.
  - Os arquivos de **texto** podem ser editados com um editor de texto.
  - Os arquivos **binários** precisam ser manipulados com um programa específico.

Arquivos de Texto

> Arquivos de texto são usados para armazenar dados legíveis por humanos. Ex.: arquivos .txt , .c, .py, .html, .xml, .json, etc

- **Criação de Arquivos:** A função `open()` é usada para criar um arquivo.

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

- **Modos de Abertura:** Há vários modos de abertura de arquivos.
  - `r` : Leitura (padrão).
  - `w` : Escrita.
  - `a` : Anexação.
  - `x` : Somente criação. Se o arquivo já existir, a operação falhará.

- **Close:** A função `close()` é usada para fechar um arquivo.

> É importante fechar um arquivo após a leitura ou escrita para liberar recursos do sistema operacional e garantir que os dados sejam gravados corretamente.

- **Verificação de Fechamento:** A função `closed` é usada para verificar se um arquivo está fechado.

```
arquivo = open("arquivo.txt", "w")
print(arquivo.closed)
arquivo.close()
print(arquivo.closed)
```

- **Escrita em Arquivos:** A função `write()` é usada para escrever em um arquivo.

```
arquivo = open("arquivo.txt", "w")
arquivo.write("Olá, Mundo!")
arquivo.close()
```

> Para escrever em um arquivo, é necessário abrir o arquivo em um modo que permita a escrita (por exemplo, `w` ou `a`).

- Modo `w`: Cria novos arquivos ou sobrescreve arquivos existentes.
- Modo `a`: Cria novos arquivos ou anexa ao final de arquivos existentes.
- Modo `x`: Cria novos arquivos. Se o arquivo já existir, a operação falhará.

- **flush():** A função `flush()` é usada para antecipar a gravação de dados em um arquivo.

```
arquivo = open("arquivo.txt", "w")
for i in range(1000):
    # (...) suposto processo demorado sujeito a falhas
    arquivo.write(str(i))
    arquivo.flush()
arquivo.close()
```

> Deve ser usado em situações em que é importante garantir que os dados sejam gravados imediatamente, mesmo que o arquivo não seja fechado.

- **Leitura de Arquivos:** A função `read()` é usada para ler um arquivo.

```
arquivo = open("arquivo.txt", "r")
conteudo = arquivo.read()
arquivo.close()
print(conteudo)
```

> read() lê todo o conteúdo do arquivo e o armazena em uma string. No entanto, se o arquivo for muito grande, isso pode consumir muita memória. Para evitar isso, é possível ler o arquivo linha por linha ou em pedaços menores.

- Parâmetro size: O parâmetro `size` é usado para especificar o número máximo de bytes a serem lidos.

```
arquivo = open("arquivo.txt", "r")
while True:
    conteudo = arquivo.read(10) # Lê 10 bytes por vez
    if not conteudo:
        break
    print(conteudo)
arquivo.close()
print(conteudo)
```

- **Leitura de Linhas:** A função `readline()` é usada para ler uma linha de cada vez.

```
arquivo = open("arquivo.txt", "r")
linha = arquivo.readline()
while linha:
    print(linha)
    linha = arquivo.readline()
arquivo.close()
```

- **Leitura de Linhas:** A função `readlines()` é usada para ler todas as linhas de uma vez e armazená-las em uma lista.

```
arquivo = open("arquivo.txt", "r")
linhas =
for linha in linhas:
    print(linha)
arquivo.close()
```

- **Uso do `with`**: O bloco `with` é usado para garantir que o arquivo seja fechado corretamente.

```
with open("arquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

> O bloco `with` garante que o arquivo seja fechado automaticamente após a execução do bloco, mesmo se ocorrer uma exceção.

> Também é possível usar `finnaly` para garantir que o arquivo seja fechado, mas o bloco `with` é mais elegante.

```
arquivo = open("arquivo.txt", "r")
try:
    conteudo = arquivo.read()
    print(conteudo)
finally:
    arquivo.close()
```

> Em ocasiões onde é necessário percorrer um arquivo de forma randômica, é possível usar o método `seek()` para mover o cursor para uma posição específica no arquivo e o método `tell()` para obter a posição atual do cursor.

- **Posição do Cursor:** O método `tell()` é usado para obter a posição atual do cursor.

```
with open("arquivo.txt", "r") as arquivo:
    print(arquivo.tell())
```

- **Movimentação do Cursor:** O método `seek()` é usado para mover o cursor para uma posição específica.

```
with open("arquivo.txt", "r") as arquivo:
    arquivo.seek(5)
    print(arquivo.read())
```

Arquivos de Texto com Dados Estruturados

- Esses arquivos possuem uma estrutura arbitrária, mas bem definida.
- Eles são usados para armazenar dados que podem ser usados para testar algoritmos e programas.
- Exemplos:
    - O site The Matrix Market é um repositório de dados de matrizes esparsas.
    - O site The DIMACS Graph Format é um repositório de dados de problemas de otimização.
    - O site TSPLIB é um repositório de dados de problemas de otimização.

- Para ler esses dados, é necessário entender a estrutura do arquivo que está, geralmente, descrita no site onde os dados estão disponíveis.
- Para algumas fontes muito populares, como o TSPLIB, existem bibliotecas que podem ser usadas para ler os dados.

Arquivos de Texto Separados por Vírgula (CSV)

> Arquivos CSV são usados para armazenar dados tabulares.

- Cada linha do arquivo é uma linha da tabela.
- Cada valor é separado por vírgula.
- O primeiro linha pode ser um cabeçalho.

- **Leitura de Arquivos CSV**: A biblioteca `csv` é usada para ler arquivos CSV.

```
import csv
with open("arquivo.csv", "r") as arquivo:
    leitor = csv.reader(arquivo)
    for linha in leitor:
        print(linha)
```

- **Escrita de Arquivos CSV**: A biblioteca `csv` é usada para escrever arquivos CSV.

```
import csv
with open("arquivo.csv", "w") as arquivo:
    escritor = csv.writer(arquivo)
    escritor.writerow(["Nome", "Idade"])
    escritor.writerow(["Albert", 30])
```

Arquivos de Texto Campos de largura fixa

- Arquivos de texto com campos fixos são usados para armazenar dados tabulares.
- Cada linha do arquivo é uma linha da tabela.
- Cada campo tem um tamanho fixo.

- É possível usar slicing para ler arquivos com campos de largura fixa.
- A função `strip()` é usada para remover espaços em branco.

```
with open("arquivo.txt", "r") as arquivo:
    for linha in arquivo:
        nome = linha[0:10].strip()
        idade = linha[10:12].strip()
        print(nome, idade)
```

- Para escrever arquivos com campos de largura fixa, é possível usar a função `format()`.

```
with open("arquivo.txt", "w") as arquivo:
    arquivo.write("{:10}{:2}\n".format("Albert", 30))
```

Arquivos texto JSON

- Arquivos JSON são usados para armazenar dados estruturados.
- A biblioteca `json` é usada para ler e escrever arquivos JSON.
- A função `dump()` é usada para escrever um arquivo JSON.

```
import json
dados = {"nome": "Albert", "idade": 30}
with open("arquivo.json", "w") as arquivo:
    json.dump(dados, arquivo)
```

- A função `load()` é usada para ler um arquivo JSON.

```
import json
with open("arquivo.json", "r") as arquivo:
    dados = json.load(arquivo)
    print(dados)
```

> Um outro formato de arquivo estruturado é o XML. Ao contrário do JSON, este formato é complexo e não é nativo do Python. Não abordaremos este formato aqui.

Ver mais

Arquivos Binários

> Arquivos binários são usados para armazenar dados não legíveis por humanos.

- Vantagens:
  - Compactação.
  - Velocidade.
  - Segurança.
- Desvantagens:
  - Não legíveis por humanos.
  - Difíceis de editar.

- Escrevendo uma lista de números inteiros em um arquivo binário.

```
numeros = [1, 2, 3, 4, 5]
with open("arquivo.bin", "wb") as arquivo:
    for numero in numeros:
        arquivo.write(numero.to_bytes(4, "little"))
```

- Lendo uma lista de números inteiros de um arquivo binário.

```
numeros = []
with open("arquivo.bin", "rb") as arquivo:
    while True:
        numero = arquivo.read(4)
        if not numero:
            break
        numeros.append(int.from_bytes(numero, "little"))
print(numeros)
```

- Escrevendo um texto em um arquivo binário.

```
texto = "Olá, Mundo!"
with open("arquivo.bin", "wb") as arquivo:
    arquivo.write(texto.encode("utf-8"))
```

- Lendo um texto de um arquivo binário.

```
with open("arquivo.bin", "rb") as arquivo:
    texto = arquivo.read().decode("utf-8")
    print(texto)
```

> A função `encode()` é usada para converter uma string em bytes.
> A função `decode()` é usada para converter bytes em uma string.

- A biblioteca `pickle` é usada para ler e escrever arquivos binários.
- A função `dump()` é usada para escrever um arquivo binário.

```
import pickle
dados = {"nome": "Albert", "idade": 30}
with open("arquivo.bin", "wb") as arquivo:
    pickle.dump(dados, arquivo)
```

- A função `load()` é usada para ler um arquivo binário.

```
import pickle
with open("arquivo.bin", "rb") as arquivo:
    dados = pickle.load(arquivo)
    print(dados)
```

Exercícios