Arquivos

CIÊNCIA DA COMPUTAÇÃO

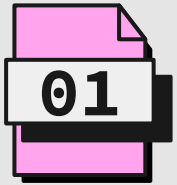
ATITUS EDUCAÇÃO

- Aula 16 -
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto

ATITUS EDUCAÇÃO

Na aula de hoje...



Arquivos

Definição
Streams
Tipos de Arquivo

Arquivos Texto

Abertura / Fechamento
Operador with
Leitura e Escrita de Dados

Até agora...

* Em todos os programas que fizemos, quando o programa termina, o computador “esquece” tudo o que aconteceu

* Todas as variáveis que você criou e usou em seus programas, sejam strings, listas, dicionários deixam de existir

* E se você quiser manter alguns dos dados gerados em um programa e salvá-los para quando o executar mais tarde?

* Ou se você quiser salvar alguns dados para que um programa diferente possa usá-los?

Porque usar Arquivos?

* Permitem que programas façam armazenamento permanente (ou persistente) de informações

* Pode-se utilizar os mesmos dados entre execuções diferentes de um programa ou de programas diferentes

* Arquivos são armazenados em dispositivos de memória secundária (disco rígido, CD, DVD, pendrives)

* Podem armazenar mais dados que a memória principal comporta

Arquivos em Python

Um arquivo é um conjunto de *bytes*, colocados um após o outro (*stream*), armazenados em um dispositivo de memória secundária

* A linguagem não impõe estrutura alguma aos arquivos

* Arquivos podem ser usados de duas formas:
  - Como fonte de dados para o programa
    - Arquivo de entrada (**input**)
  - Como destino de dados de um programa
    - Arquivo de saída (**output**)

Streams

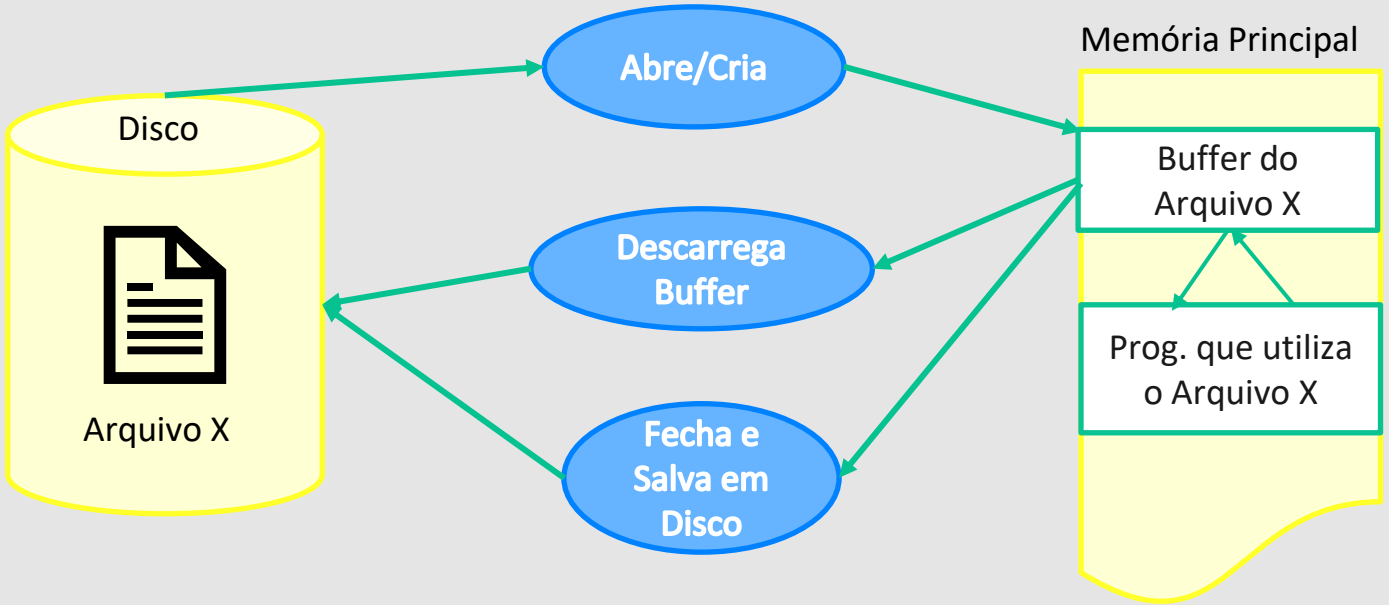
* O sistema de E/S do Python utiliza o conceito de streams.

* Um stream é um dispositivo lógico (um canal de manipulação) que representa um arquivo ou um dispositivo qualquer (monitor, teclado, disco rígido, etc).

* Operações sobre streams são “**bufferizadas**”, ou seja, trabalham com uma área de memória intermediária (buffer) e não propriamente sobre o disco.

* Toda a entrada ou saída de dados é processada através do processamento de streams, independentemente do periférico utilizado (device independent)

Utilização de arquivos



Tipos de Arquivos

* Arquivos em Python podem ser utilizados em dois modos de processamento dos bytes que compõem o stream:
  - → Modo binário (estruturado)
  - → **Modo texto** (caracteres)

Imagens: jpg, png, gif, bmp...

Vídeo: mp4, mkv, avi...

Compactados: zip, rar, 7z...

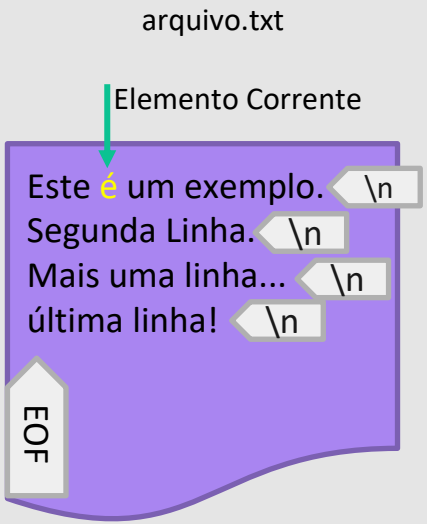
Dados: csv, json, xml...

Web: html, css, svg...

Foco dessa aula!

Tipos de Arquivos

* Armazenam dados sempre em formato de texto (`str`)
  - → Similar ao comportamento dos comandos de entrada do teclado (`input`) e saída na tela (`print`)
  - → Dados de outros tipos (`int`, `float`, `bool`, etc.) devem ser convertidos quando lidos ou armazenados
  - → A unidade de armazenamento no stream não é especificada em bytes mas sim em caracteres
* Sobre caracteres especiais:
  - → Marcador de final de linha é dependente de sistema operacional ("\n" ou "\r" ou "\r\n")
  - → Convencionado "\n" no Python
  - → Outros caracteres, como "\t", também podem ser salvos em arquivos



Abertura do Arquivo

* Para processar um arquivo, deve-se associar uma variável (objeto) de nosso programa ao arquivo, operação denominada de abertura de arquivo

* A função embutida `open(filename [, mode, …])` retorna um objeto que permite manipular o arquivo desejado

A função open recebe outros parâmetros. Veja referência completa em docs.python.org/3/library/functions.html#open

Abertura do Arquivo

* O parâmetro filename deve conter uma string indicando o caminho (relativo ou absoluto) até o arquivo, como por exemplo:

```
open("arquivo.txt")

open("C:/Users/Aluno/Desktop/arquivo.txt")
```

O Windows usa contrabarra (\) como separador de diretórios, mas o Python aceita a barra simples. Assim evitamos ter que usar duas contrabarras (\\).

Abertura do Arquivos

* O parâmetro opcional `mode` (no modo texto) pode ser:

  → `"r"` para indicar abertura de arquivo para leitura (valor padrão)
    - Se o arquivo não existir, ocorrerá um erro de leitura

  → `"w"` para indicar abertura de arquivo para escrita
    - Se o arquivo já existir, este será sobrescrito

  → `"a"` para indicar abertura de arquivo para escrita no modo append
    - Se o arquivo já existir, este será complementado ao final

Abertura do Arquivos

* O parâmetro opcional `mode` (no modo texto) pode ser:

  → `"r+"` para indicar abertura de arquivo para leitura e escrita
    - Se o arquivo não existir, ocorrerá um erro de leitura

  → `"w+"` para indicar abertura de arquivo para leitura e escrita
    - Se o arquivo já existir, este será sobrescrito

  → `"a+"` para indicar abertura de arquivo para leitura e escrita no modo append
    - Se o arquivo já existir, este será complementado ao final

O parâmetro `mode` pode assumir outros valores. Veja referência completa em docs.python.org/3/library/functions.html#open

Controle de Erros

* Se, por alguma razão, o arquivo desejado não puder ser aberto, a função embutida `open(...)` dispara uma exceção do tipo `OSError`

* Usar `try-except` para tratar exceções em arquivos:

```
try:
    f = open("arquivo.txt", "r")
    # Manipula o arquivo
except OSError:
    # Tratar a exceção
    print("Ocorreu um erro!")
```

`OSError` é um erro genérico. Outros erros mais específicos também podem ser tratados. Veja a documentação completa em docs.python.org/3/library/exceptions.html#os-exceptions

Fechando o Arquivo

* Uma vez aberto, depois de manipulado, o arquivo deve ser fechado através do método `close()` do objeto retornado pela função `open()`

```
try:
    f = open("arquivo.txt", "r")
    # Manipula o arquivo
    f.close()
except OSError:
    # Tratar a exceção
    print("Ocorreu um erro!")
```

Operador with

* Uma boa prática na manipulação de arquivos é usar um gerenciador de contexto com a instrução `with`, responsável por chamar implicitamente o método `close()` quando necessário

```
try:
    with open("arquivo.txt", "r") as f:
        # Manipula o arquivo
        pass
except OSError:
    # Tratar a exceção
    print("Ocorreu um erro!")
```

Lendo dados de um arquivo

* No modo texto, os **métodos** mais utilizados para manipulação de arquivos são:
  - `read([size])`: lê uma quantidade de dados do arquivo, indicada por size, de uma só vez. Se o parâmetro size é omitido, a função retorna o conteúdo completo do arquivo. O retorno desse método no fim do arquivo é uma string vazia ("").
  - `readline()`: lê uma linha do arquivo (até o próximo "\n"). O retorno desse método no fim do arquivo é uma string vazia ("").
  - `readlines()`: lê todas as linhas do arquivo e retorna uma lista com o conteúdo.

Lendo dados de um arquivo

```
# Programa que lê todas as linhas de um arquivo
print("FORMA 1: Usando readline")
try:
    with open("arquivo.txt", "r", encoding='utf-8') as arquivo:
        linha = " "
        i = 1
        while linha:
            linha = arquivo.readline()
            if linha != "":
                print(f"{i} - {linha}")
                i += 1
except OSError:
    print("Erro ao abrir o arquivo!")
```

Lendo dados de um arquivo

```
print("FORMA 2: Usando read")
try:
    with open("arquivo.txt", "r", encoding='utf-8') as arquivo:
        linhas = arquivo.read()
        print(linhas)
except OSError:
    print("Erro ao abrir o arquivo!")

print("FORMA 3: Usando readlines")
try:
    with open("arquivo.txt", "r", encoding='utf-8') as arquivo:
        linhas = arquivo.readlines()
        for linha in linhas:
            print(linha)
except OSError:
    print("Erro ao abrir o arquivo!")
```

Escrevendo dados no arquivo

* No modo texto, os métodos mais utilizados para manipulação de arquivos são:
  - → `write(string)`: escreve um conjunto de caracteres (`string`) no arquivo e retorna o número de caracteres escritos.
  - → `tell()`: indica qual a posição corrente da leitura/escrita (número de caracteres desde o início do arquivo).
  - → `seek(offset[, whence])`: altera a posição corrente da leitura/escrita adicionando um deslocamento (offset) relativo a alguma referência (whence):
    - whence = 0 indica o início do arquivo como a referência
    - whence = 1 indica a posição corrente do arquivo como a referência
    - whence = 2 indica o fim do arquivo como a referência

Escrevendo dados no arquivo

```python
# Programa que escreve a lista abaixo em um arquivo
linguagens = ["Python", "Java", "C#", "Php", "C"]

try:
    with open("linguagens.txt", "w", encoding='utf-8') as arquivo:
        for ling in linguagens:
            arquivo.write(ling + "\n") # O \n deve ser incluído explicitamente
except OSError:
    print("Erro ao abrir o arquivo!")
```

Problema 1: Arquivo de Números

Escreva um programa que leia três valores inteiros: `min`, `max` e `qtde`.

Seu programa deve escrever em um arquivo `qtde` números aleatórios no intervalo [`min`, `max`]

Problema 1: Arquivo de Números

```python
import random

qtde = int(input("Informe a quantidade de valores: "))
minimo = int(input("Valor Mínimo: "))
maximo = int(input("Valor Máximo: "))

with open("numeros.txt", mode="w") as arquivo:
    for i in range(qtde):
        num = random.randint(minimo, maximo)
        arquivo.write(str(num) + "\n")
```

Problema 2: Arquivo de Números

Escreva um programa que leia os valores que foram salvos no arquivo do problema anterior.

Apresente o maior valor que está salvo no arquivo.

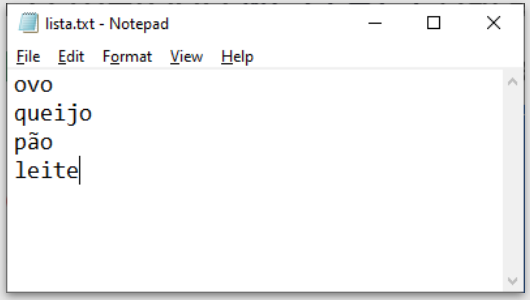
Problema 2: Arquivo de Números

```
maior = float('-inf')   # Inicializa maior com um número muito pequeno
with open("numeros.txt", mode="r") as arquivo:
    linhas = arquivo.readlines()
    for linha in linhas:
        numero = int(linha)
        if numero > maior:
            maior = numero

print(f"O maior número é {maior}")
```

Problema 3: Lista de Compras

Partindo do programa que gerencia uma lista de compras, feito na aula sobre listas, faça com que ele salve a lista de compras em um arquivo chamado “compras.txt”, quando o usuário sair do programa e carregue a lista de um arquivo quando o programa iniciar.



Problema 3: Lista de Compras

```
def carrega_compras(lista_compras):
    try:
        with open(ARQUIVO, "r", encoding='utf-8') as ar
            compras = arquivo.readlines()
            for compra in compras:
                lista_compras.append(compra.strip())
    except OSError:
        pass

def salva_compras(lista_compras):
    try:
        with open(ARQUIVO, "w", encoding="utf-8") as arquivo:
            for compra in lista_compras:
                arquivo.write(compra)
                arquivo.write("\n")
    except OSError:
        print("Erro ao abrir o arquivo!")
```

Se existir, o arquivo `"lista.txt"` será aberto para leitura (`"r"`)

Adicionamos cada linha do arquivo removendo a quebra de linha do final com o método `strip()` do objeto `str`

Se o arquivo não existe um `FileNotFoundError` é gerado. Isso não é nenhum problema, pode ser a primeira execução do programa

É preciso inserir a quebra de linha (`"\n"`) explicitamente para que cada item da lista fique em uma linha diferente do arquivo