

# Algoritmos e Lógica de Programação

80 horas // 4 h/semana

---

## ***Arquivos em Python***

### **Aula 14**

Prof. Piva

# Para começar...

---

Até aqui, todos os programas e scripts que fizemos carregavam ou manipulavam os dados apenas na Memória Principal do computador.

Uma vez desligado, todos os dados são perdidos.

Hoje vamos aprender a como ler e salvar dados em arquivos (e guarda-los permanentemente) com a linguagem Python.

# Arquivos...pra que servem?

Nós já aprendemos como obter dados e trabalharmos com eles...



# Arquivos...pra que servem?

Nós já aprendemos como obter dados e trabalharmos com eles...



**E depois?**

**Como manter esses dados?**

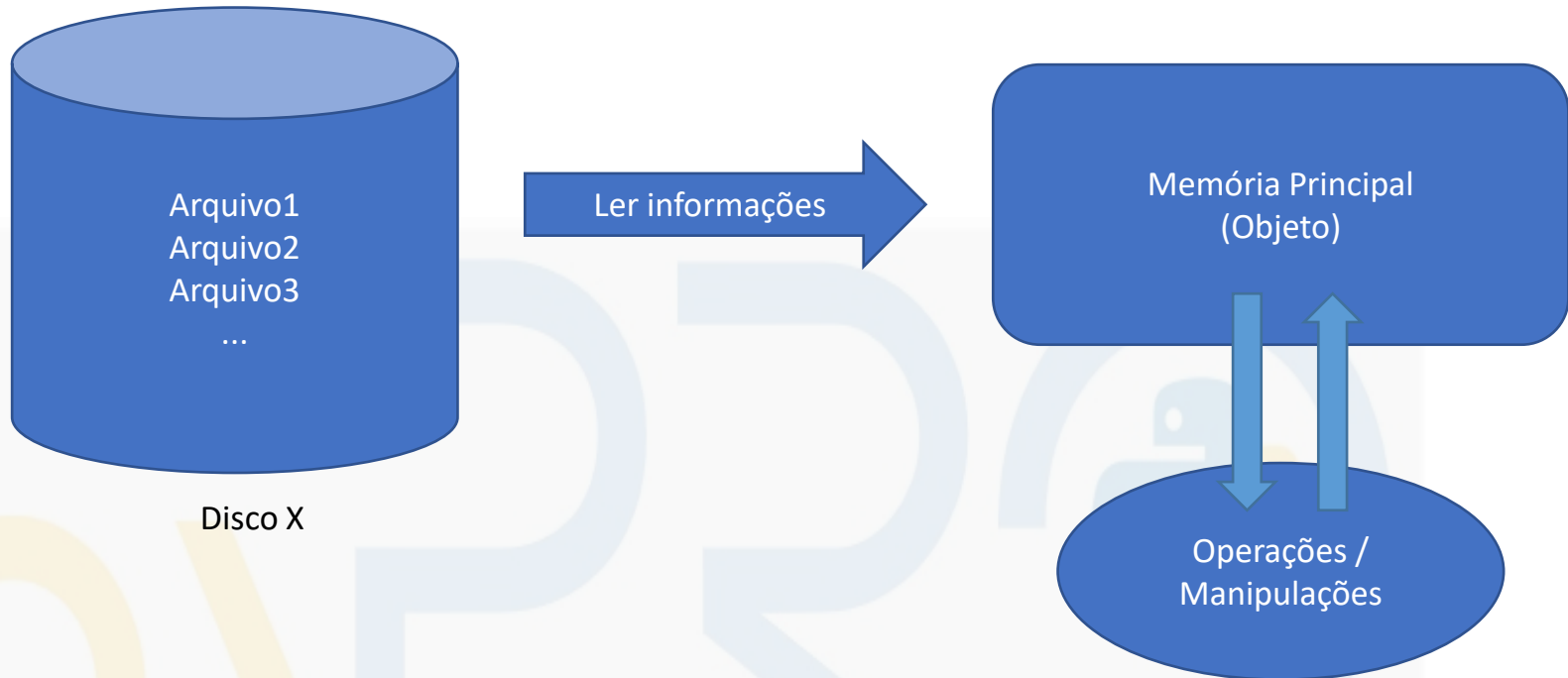
# Arquivos...pra que servem?



# Arquivos...pra que servem?



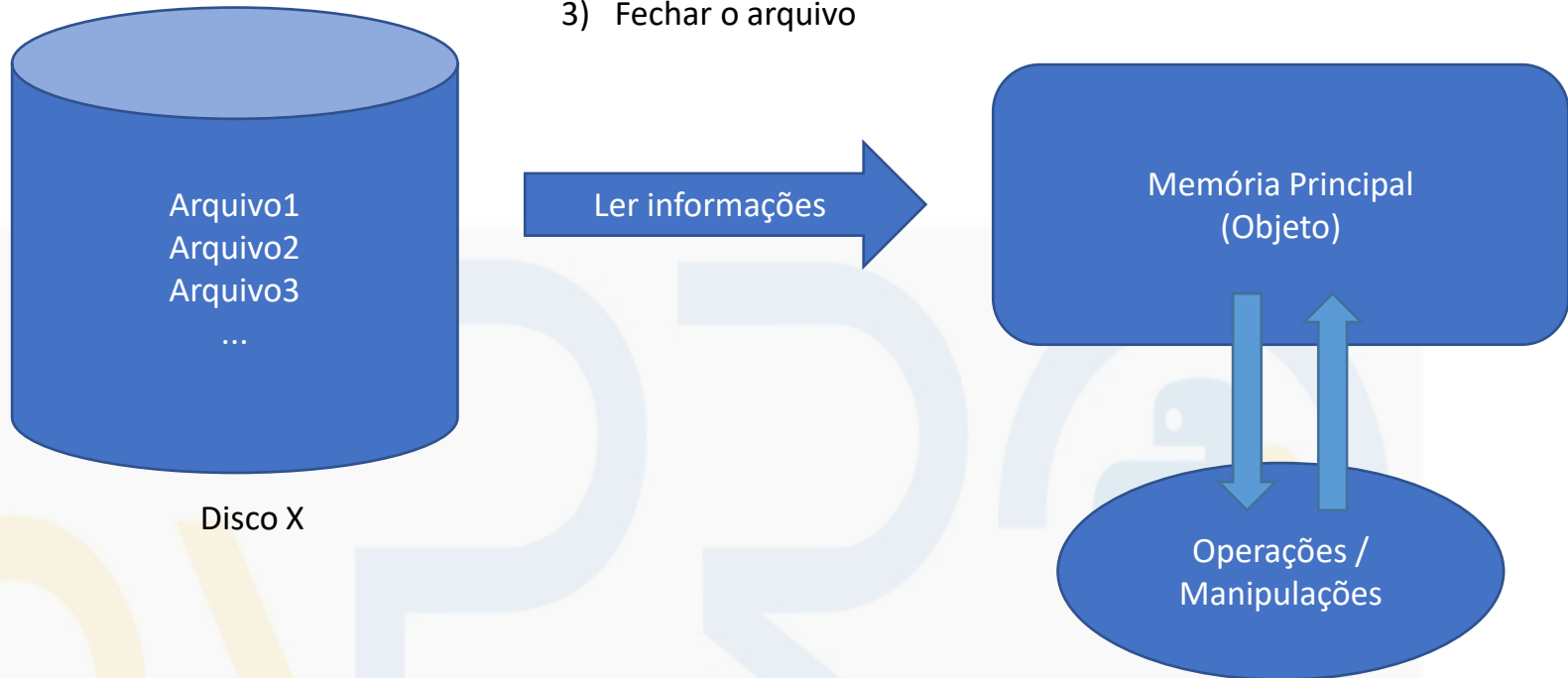
# Arquivos... Operações/Manipulações



# Arquivos... Operações/Manipulações

Etapas:

- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) Fechar o arquivo



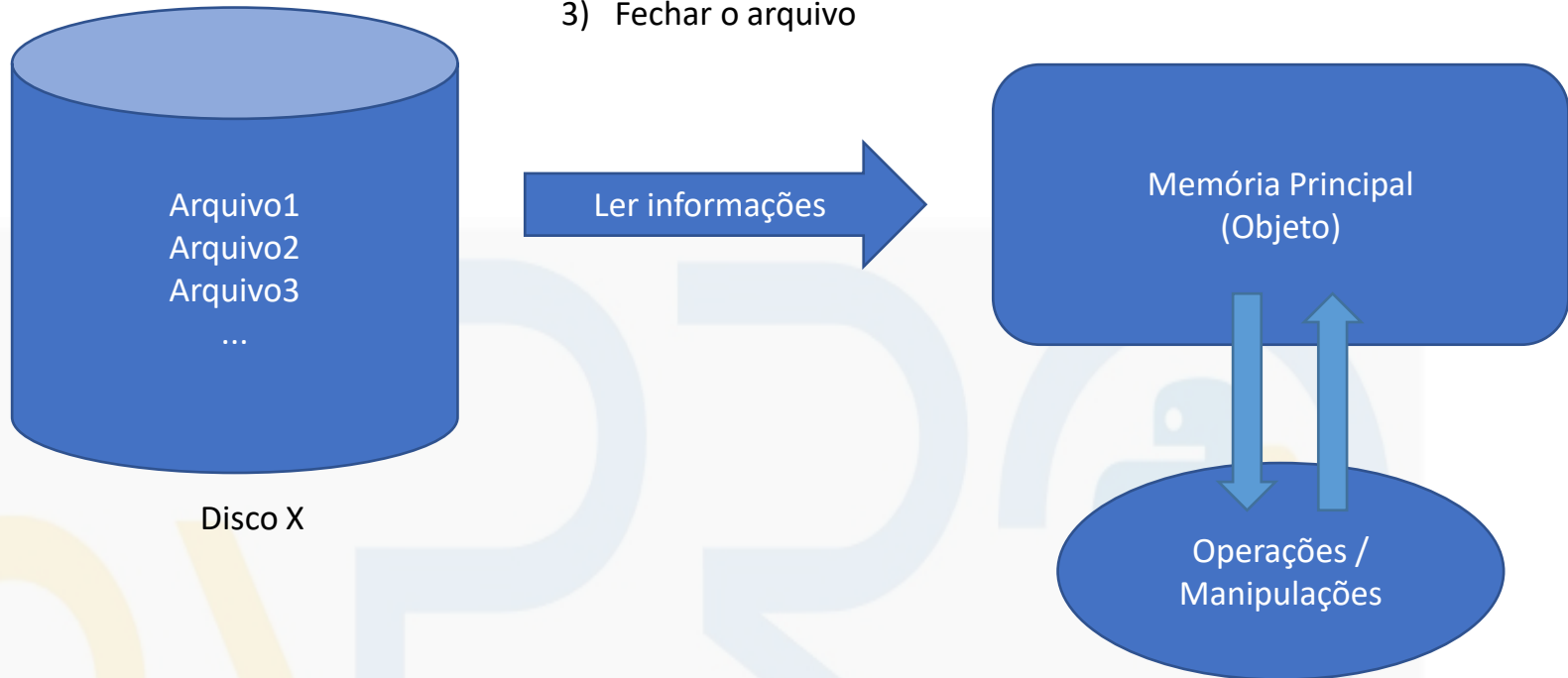


# Arquivos... Operações/Manipulações

Etapas:

- 1) **Abrir o arquivo**
- 2) Realiza as operações
- 3) Fechar o arquivo

**Temos que informar qual o arquivo queremos abrir e quais operações queremos fazer com esse arquivo.**



# Arquivos... Operações/Manipulações

Etapas:

- 1) **Abrir o arquivo**
- 2) Realiza as operações
- 3) Fechar o arquivo

Temos que informar qual o arquivo queremos abrir quais operações queremos fazer com esse arquivo

Modo de abertura do arquivo



Disco X

Ler informações

Memória Principal  
(Objeto)

Operações /  
Manipulações

# Arquivos... Operações/Manipulações

Etapas:

- 1) **Abrir o arquivo**
- 2) Realiza as operações
- 3) Fechar o arquivo



**Temos que informar qual o arquivo**  
**queremos abrir** **quais operações**  
**queremos fazer com esse arquivo**



Modo de  
abertura  
do arquivo

## Modos de Abertura de arquivos...

Modo	Operações
r	leitura
w	escrita, apaga o conteúdo se já existir
a	escrita, mas preserva o conteúdo se já existir
b	modo binário
+	atualização (leitura e escrita)

# Arquivos... Operações/Manipulações

Etapas:

- 1) **Abrir o arquivo**
- 2) Realiza as operações
- 3) Fechar o arquivo



**Temos que informar qual o arquivo**  
**queremos abrir** **quais operações**  
**queremos fazer com esse arquivo**



Modo de  
abertura  
do arquivo

## Modos de Abertura de arquivos...

Modo	Operações
r	leitura
w	escrita, apaga o conteúdo se já existir
a	escrita, mas preserva o conteúdo se já existir
b	modo binário
+	atualização (leitura e escrita)

"r+"

"w+"

"a+"

"r+b"

"w+b"

"a+b"

# Arquivos... Abertura de Arquivos

Etapas:

- 1) **Abrir o arquivo** →
- 2) Realiza as operações
- 3) Fechar o arquivo

Modo	Operações
r	leitura
w	escrita, apaga o conteúdo se já existir
a	escrita, mas preserva o conteúdo se já existir
b	modo binário
+	atualização (leitura e escrita)

“r+”  
 “w+”  
 “a+”  
 “r+b”  
 “w+b”  
 “a+b”

Comando para abrir um arquivo: **open()**

**open**(file, mode= 'r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

- **file** - identifica o arquivo que se quer abrir
- **mode** – o modo de abertura. Caso não seja passado um modo, o padrão é ‘r’
- **buffering** – é um inteiro opcional usado para definir a política de bufferização. 0 (desativa), 1 (seleciona o buffer de linha), >1 (o tamanho do buffer). O padrão é -1 (usar as configurações do sistema).
- **encoding** – codificação do arquivo (caso exista) – Exemplo: “utf-8”
- **errors** – string opcional que indica como os erros de codificação devem ser tratados
- **newline** – como controla o modo de nova linha (\n \r... \r\n...)
- **closefd** – para indicar se o arquivo fornecido é um arquivo ou um descritor de arquivo
- **opener** – arquivo que se refere o descritor.

Exemplo: **arquivo = open(“texto.txt”, “r”)**

# Arquivos... Abertura de Arquivos

Etapas:

- 1) **Abrir o arquivo** →
- 2) Realiza as operações
- 3) Fechar o arquivo

Modo	Operações
r	leitura
w	escrita, apaga o conteúdo se já existir
a	escrita, mas preserva o conteúdo se já existir
b	modo binário
+	atualização (leitura e escrita)

“r+”  
 “w+”  
 “a+”  
 “r+b”  
 “w+b”  
 “a+b”

Comando para abrir um arquivo: **open()**

**open**(file, mode= 'r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

- **file** - identifica o arquivo que se quer abrir
- **mode** – o modo de abertura. Caso não seja passado um modo, o padrão é ‘r’
- **buffering** – é um inteiro opcional usado para definir a política de bufferização. 0 (desativa), 1 (seleciona o buffer de linha), >1 (o tamanho do buffer)
- **encoding** – codificação do arquivo (caso exista)
- **errors** – string opcional que indica como os erros de codificação devem ser tratados
- **newline** – como controla o modo de nova linha (\n \r... \r\n...)
- **closefd** – para indicar se o arquivo fornecido é um arquivo ou um descritor de arquivo
- **opener** – arquivo que se refere o descritor.

Exemplo: **arquivo = open(“texto.txt”, “r”)**  
 type(arquivo) → TextIOWrapper

# Arquivos... Abertura de Arquivos

Etapas:

- 1) **Abrir o arquivo** →
- 2) Realiza as operações
- 3) Fechar o arquivo

Modo	Operações
r	leitura
w	escrita, apaga o conteúdo se já existir
a	escrita, mas preserva o conteúdo se já existir
b	modo binário
+	atualização (leitura e escrita)

"r+"  
"w+"  
"a+"  
"r+b"  
"w+b"  
"a+b"

Comando para

**open**(file, mode)

- **file** – nome do arquivo
- **mode** – modo de abertura
- **buffering** – tamanho do buffer
- **encoding** – codificação de caracteres
- **errors** – tratamento de erros
- **newline** – tratamento de quebras de linha
- **close** – fecha o arquivo
- **open** – abre o arquivo

## TextIOWrapper

Classe de implementação de manipulação de arquivos de texto.

Manipulam strings (que são sequências de bytes).

Quando escrevem ou lêem dados em arquivos existe a conversão automática por meio de um *encoder*.

Nos arquivos do tipo texto, isso já é feito automaticamente.

type(arquivo) → TextIOWrapper

# Arquivos... Leitura

Exemplo:

```
arquivo = open("texto.txt", "r")  
type(arquivo) → TextIOWrapper
```



Etapas:

- 1) Abrir o arquivo
- 2) **Realiza as operações**
- 3) Fechar o arquivo



**Vamos começar com a  
operação de leitura**

```
texto = arquivo.read()  
print(texto)
```



# Arquivos... Leitura

Etapas:

- 1) Abrir o arquivo
- 2) **Realiza as operações**
- 3) Fechar o arquivo



**Vamos começar com a operação de leitura**

Exemplo:

```
arquivo = open("texto.txt", "r")  
type(arquivo) → TextIOWrapper
```



```
texto = arquivo.read()  
print(texto)
```



**Carrega em memória (na variável) o conteúdo que está no arquivo**

# Arquivos... Leitura

Etapas:

- 1) Abrir o arquivo
- 2) **Realiza as operações**
- 3) Fechar o arquivo

**Vamos começar com a operação de leitura**

Exemplo:

```
arquivo = open("texto.txt", "r")
type(arquivo) → TextIOWrapper
```



```
texto = arquivo.read()
print(texto)
```



**Carrega em memória (na variável) o conteúdo que está no arquivo**

**type(texto) → <class 'str'>**

# Arquivos... Leitura

Etapas:

- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) **Fechar o arquivo**

**Para finalizar o processo temos que fechar (liberar) o arquivo**

Exemplo:

```
arquivo = open("texto.txt", "r")
type(arquivo) → TextIOWrapper
texto = arquivo.read()
print(texto)
```

`arquivo.close()`

**Libera o arquivo para outros usuários**

# Arquivos... Leitura (Processo)

Etapas:

1) Abrir o arquivo

2) Realiza as operações

3) Fechar o arquivo



```
arquivo=open("texto.txt",  
"r")
```



```
texto = arquivo.read()  
print(texto)
```



```
arquivo.close()
```

# VAMOS PARA A PRÁTICA ?!!!

---



# **Python: Arquivos**

# Arquivos

- Entrada e saída são operações de comunicação de um programa com o mundo externo
- Essa comunicação se dá usualmente através de *arquivos*
- Arquivos estão associados a dispositivos
  - Por exemplo, disco, impressora, teclado
- Em Python, um arquivo pode ser lido/escrito através de um objeto da classe file

# Arquivos default

- Já usamos, sem saber, três arquivos *default*
  - Sempre que um comando `print` é executado, o resultado vai para um arquivo chamado `sys.stdout`
  - Sempre que lemos um dado através do comando `input`, na verdade estamos lendo de um arquivo chamado `sys.stdin`
  - Mensagens de erro ou de rastreamento de exceções são enviadas para um arquivo chamado `sys.stderr`



# Exemplo

```
>>> import sys
```

```
>>> sys.stdout.write("alo")
```

```
alo
```

```
>>> print("alo")
```

```
alo
```

```
>>> sys.stdin.readline()
```

```
sfadfas
```

```
'sfadfas\n'
```

```
>>> input()
```

```
fasdfadsf
```

```
'fasdfadsf'
```

# Redirecionamento

- Os arquivos `sys.stdin`, `sys.stdout` e `sys.stderr` normalmente estão associados ao teclado e ao display do terminal sendo usado, mas podem ser reassociados a outros dispositivos
  - Em *Unix/Linux* e *Windows*:
    - `programa > arquivo`
      - Executa programa redirecionando `stdout` para arquivo
    - `programa < arquivo`
      - Executa programa redirecionando `stdin` de arquivo
    - `programa1 | programa2`
      - Executa `programa1` e `programa2` sendo que a saída de `programa1` é redirecionada para a entrada de `programa2`

# Redirecionamento

- Em *Linux* com shell **bash**
  - programa 2> arquivo
    - Executa programa redirecionando stderr para arquivo

# Abrindo arquivos

- `open (name, mode, buffering)`
  - *name* : nome do arquivo a abrir
  - *mode* : (opcional) modo de abertura – string contendo
    - `r` : leitura (default)
    - `w` : escrita
    - `b` : binário
    - `a` : escrita a partir do final
    - `+` : (usado com `r`) indica leitura e escrita

# Abrindo arquivos

- *buffering* : (opcional) indica se memória (*buffers*) é usada para acelerar operações de entrada e saída
  - 0 : buffers não são usados
  - 1 (ou qq número negativo): um buffer de tamanho padrão (default)
  - 2 ou maior: tamanho do buffer em bytes

# O objeto *file*

- O comando `open` retorna um objeto do tipo *file* (arquivo)
  - Na verdade, em Python 2.4 em diante, `open` é o mesmo que `file`, e portanto o comando é um construtor
- O objeto retornado é usado subsequentemente para realizar operações de entrada e saída:

```
>>> arq = open ("teste", "w")
```

```
>>> arq.write ("Oi")
```

```
>>> arq.close ()
```

```
>>> arq = open ("teste")
```

```
>>> x = arq.read()
```

```
>>> x
```

```
'Oi'
```

# Métodos *Read*, *Write* e *Close*

- `read(num)`
  - Lê *num* bytes do arquivo e os retorna numa string
  - Se *num* não é especificado, todos os bytes desde o ponto atual até o fim do arquivo são retornados
- `write(string)`
  - Escreve *string* no arquivo
  - Devido ao uso de buffers, a escrita pode não ser feita imediatamente
    - Use o método `flush()` ou `close()` para assegurar a escrita física
- `close()`
  - Termina o uso do arquivo para operações de leitura e escrita

# Convenção de fim de linha

- Arquivos de texto são divididos em linhas usando caracteres especiais
  - Linux/Unix: `\n`
  - Windows: `\r\n`
  - Mac: `\r`
- Python usa sempre `\n` para separar linhas
  - Ao se ler/escrever um arquivo aberto em modo texto (não binário) faz traduções de `\n` para se adequar ao sistema operacional
  - Em modo binário, entretanto, a conversão não é feita



# Interação com o Sistema Operacional

- Operações de entrada e saída são na verdade realizadas pelo sistema operacional
- O módulo `os` possui diversas variáveis e funções que ajudam um programa Python a se adequar ao sistema operacional, por exemplo:
  - `os.getcwd()` retorna o diretório corrente
  - `os.chdir(dir)` muda o diretório corrente para *dir*
  - `os.sep` é uma string com o caractere que separa componentes de um caminho ('/' para *Unix*, '\\' para *Windows*)
  - `os.path.exists(path)` diz se *path* se refere ao nome de um arquivo existente

# Lendo e escrevendo linhas

## ■ `readline(n)`

- Se *n* não é especificado, retorna exatamente uma linha lida do arquivo
- Caso contrário, lê uma linha, mas busca no máximo *n* caracteres pelo final de linha

## ■ `readlines(n)`

- Se *n* não é especificado, retorna o restante do conteúdo do arquivo em uma lista de strings
- Caso *n* seja especificado, a leitura é limitada a *n* caracteres no máximo

# Lendo e escrevendo linhas

- `writelines(seqüência)`
  - Escreve a lista (ou qualquer seqüência) de strings, uma por uma no arquivo
  - Caracteres terminadores de linha *não são* acrescentados

# Acesso direto

- É possível ler e escrever não seqüencialmente em alguns tipos de arquivo
  - Devem estar associados a dispositivos que permitem acesso direto, como discos, por exemplo
- `seek(offset, whence)`
  - *offset* indica o número do byte a ser lido e escrito pela próxima operação de entrada e saída
  - *whence* indica a partir de onde *offset* será contado
    - 0 (default) : do início
    - 1 : do ponto corrente
    - 2 : do final

# Acesso direto

- tell()
  - Indica a posição corrente (número de bytes a partir do início do arquivo)

# Comando "with"

Forma "pythônica" de trabalhar com arquivos...



# Arquivos... Forma "normal"

Etapas:

1) Abrir o arquivo

2) Realiza as operações

3) Fechar o arquivo

```
arquivo=open("texto.txt", "r")
```

```
texto = arquivo.read()
```

```
print(texto)
```

```
arquivo.close()
```

# Arquivos... Forma “normal” um pouco melhor...

Etapas:

- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) Fechar o arquivo

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



# Arquivos... Forma “normal” um pouco melhor...

Etapas:

- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) Fechar o arquivo

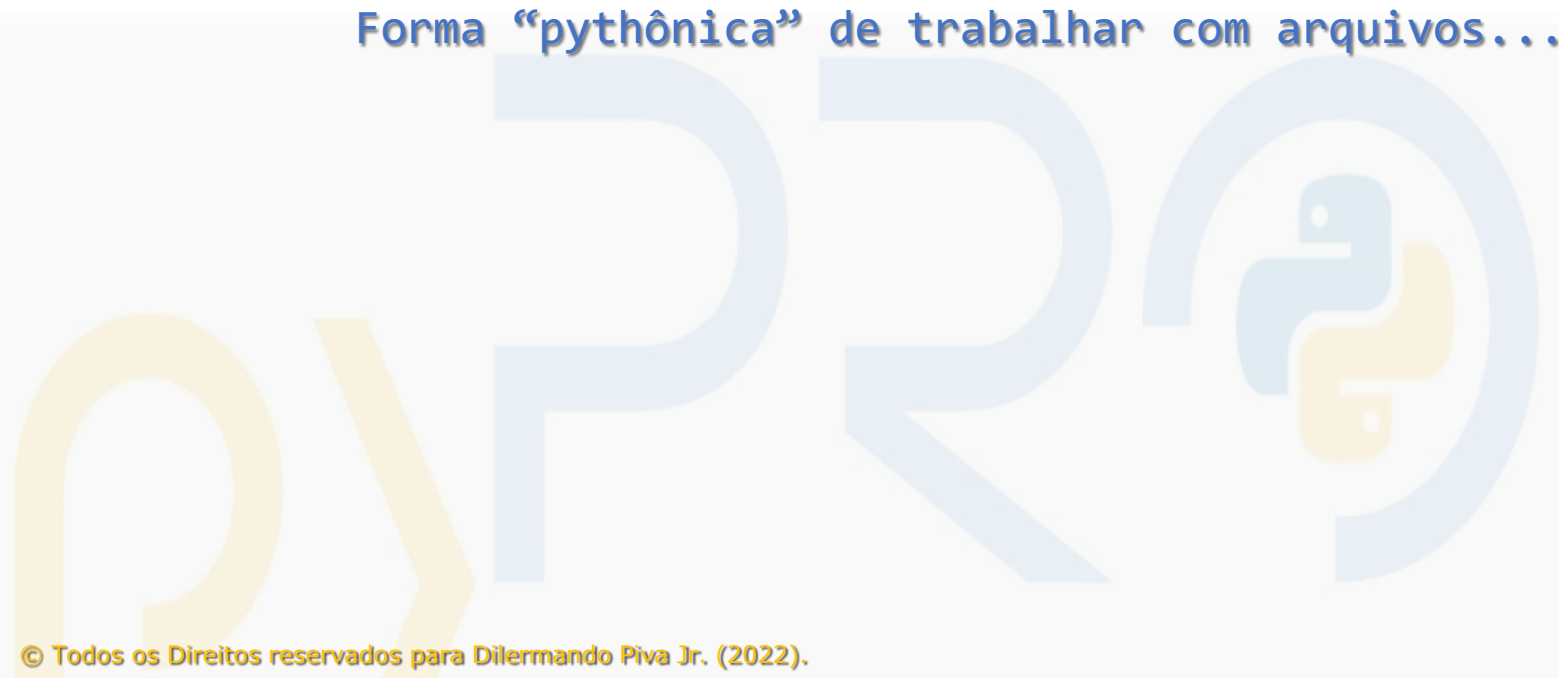
```
arquivo=open("texto.txt", "r")  
  
for linha in arquivo:  
    print(linha.strip())  
  
arquivo.close()
```

**EM AMBAS...**

**ABRIMOS... FAZEMOS AS OPERAÇÕES E FECHAMOS O ARQUIVO...**

# Comando "with"

Forma "pythônica" de trabalhar com arquivos...



# Arquivos... Forma “pythônica”

Etapas:

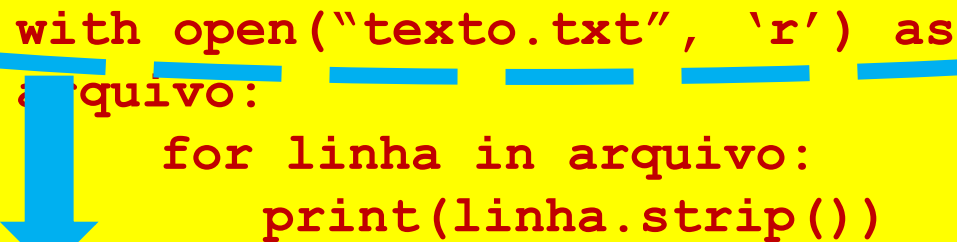
- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) Fechar o arquivo

```
with open("texto.txt", 'r') as  
arquivo:  
    for linha in arquivo:  
        print(linha.strip())
```

# Arquivos... Forma “pythônica”

Etapas:

- 1) Abrir o arquivo
- 2) Realiza as operações
- 3) Fechar o arquivo



```
with open("texto.txt", 'r') as
arquivo:
    for linha in arquivo:
        print(linha.strip())
```

**Comando WITH → introduzido na PEP-343**

**Gerenciador de contexto → que gerencia o ciclo de manipulação de arquivos.**

**NOTE QUE NÃO HÁ A NECESSIDADE DE FECHAR O ARQUIVO... ISSO É FEITO AUTOMATICAMENTE PELO GERENCIADOR DE CONTEXTO QUANDO SAI DO BLOCO**

# Arquivos...Atualização de conteúdo

## Atualizar o arquivo de Frutas

```
with open("texto.txt", "w") as arquivo:
    while True:
        fruta = input("Digite um texto:")
        if fruta == 'sair':
            break
        else:
            arquivo.write(fruta)
            arquivo.write("\n")
```

# EXERCÍCIOS

---

Arquivos em  
Python...

# EXERCÍCIO 1

A ACME Inc., uma empresa de 500 funcionários, está tendo problemas de espaço em disco no seu servidor de arquivos. Para tentar resolver este problema, o Administrador de Rede precisa saber qual o espaço ocupado pelos usuários, e identificar os usuários com maior espaço ocupado. Através de um programa, baixado da Internet, ele conseguiu gerar o seguinte arquivo, chamado "usuarios.txt":

```
alexandre      456123789
anderson       1245698456
antonio        123456456
carlos         91257581
cesar          987458
rosemary       789456125
```

Neste arquivo, o nome do usuário possui 15 caracteres. A partir deste arquivo, você deve criar um programa que gere um relatório, chamado "relatório.txt", no seguinte formato:

```
ACME Inc.                Uso do espaço em disco pelos usuários
-----
Nr.  Usuário        Espaço utilizado     % do uso
1    alexandre      434,99 MB            16,85%
2    anderson       1187,99 MB           46,02%
3    antonio        117,73 MB            4,56%
4    carlos         87,03 MB             3,37%
5    cesar          0,94 MB              0,04%
6    rosemary       752,88 MB            29,16%
```

```
Espaço total ocupado: 2581,57 MB
Espaço médio ocupado: 430,26 MB
```