

# Operações em Arquivos

Organização e Recuperação de Dados

Profa. Valéria

UEM – CTC – DIN

# Arquivos

- Um arquivo é uma estrutura criada em uma unidade de armazenamento secundário (HD, SSD, etc.)
  - Permite o armazenamento permanente dos dados, ao contrário das variáveis, que são armazenadas em RAM
- Do ponto de vista físico, um arquivo corresponde a uma sequência de bytes armazenados em um dispositivo secundário
  - Mas os aspectos físicos dos arquivos são transparentes para as aplicações

# Diferentes visões do arquivo

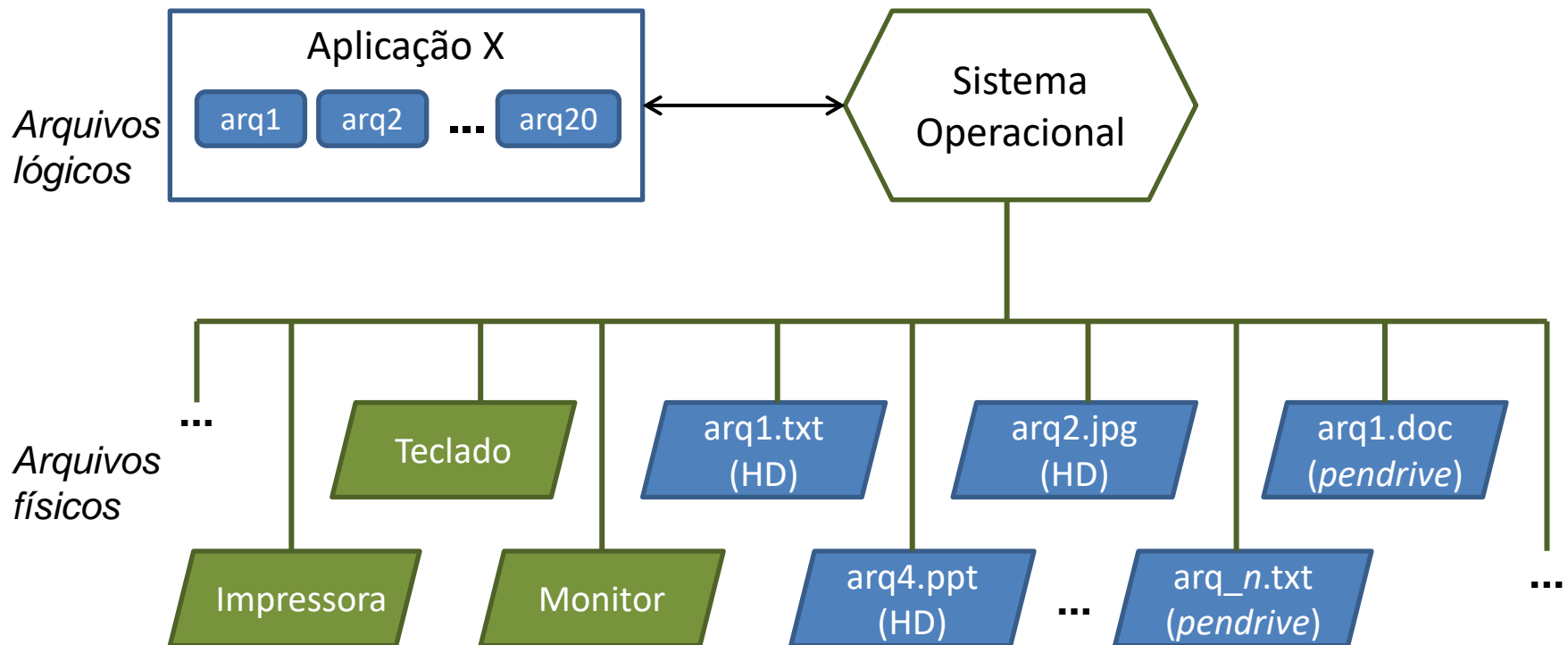
- Arquivo físico e arquivo lógico
  - Arquivo físico: é o arquivo do ponto de vista do armazenamento
    - Corresponde a uma sequência particular de bytes armazenados no dispositivo
  - Arquivo lógico: é o arquivo do ponto de vista da aplicação que o acessa
    - Cada arquivo manipulado pela aplicação é tratado como um canal de comunicação estabelecido entre a aplicação e o arquivo físico
- O sistema operacional é quem faz a interface entre o arquivo lógico e o arquivo físico

# Diferentes visões do arquivo

- Apesar de um dispositivo secundário poder conter milhares de arquivos físicos, as aplicações, em geral, só podem abrir um número pré-definido de arquivos lógicos de forma simultânea
- Um arquivo lógico pode estar associado a um arquivo físico ou a outros dispositivos de E/S, como o teclado (***stdin***) e o vídeo (***stdout*** e ***stderr***)
- Quando uma aplicação solicita a abertura de um arquivo, o S.O. é informado sobre o modo como a associação entre o arquivo lógico e o físico deve ocorrer
  - A forma como essa informação é passada varia de acordo com o S.O. e a linguagem de programação utilizada

# Conectando arquivo lógico ao físico

- A aplicação solicita que um nome lógico seja associado a um arquivo físico específico e o S.O. estabelece a associação



# Arquivo lógico

- Do ponto de vista da aplicação, um arquivo é uma entidade lógica vista como uma sequência de bytes
  - A leitura e a escrita de informação no arquivo não se preocupa com os aspectos físicos do armazenamento
  - Podemos enviar bytes para o arquivo indefinidamente
  - De forma análoga, podemos ler bytes do arquivo enquanto houverem bytes a serem lidos
- Para todo arquivo lógico, é mantido um ponteiro de L/E que indica a posição do próximo byte a ser lido/escrito
  - Esse ponteiro se move automaticamente conforme operações de leitura e escrita são realizadas e também pode ser movido por comandos de reposicionamento

# Arquivo lógico

- Quando um arquivo é aberto, o ponteiro de L/E é posicionado no início (**byte 0**)

T	h	i	s		i	s		a	n		a	p	p	l	e	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ponteiro de L/E

- Se for feita uma leitura de 9 bytes, o ponteiro de L/E se move automaticamente para a próxima posição de L/E (**byte 9**)

T	h	i	s		i	s		a	n		a	p	p	l	e	.
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ponteiro de L/E

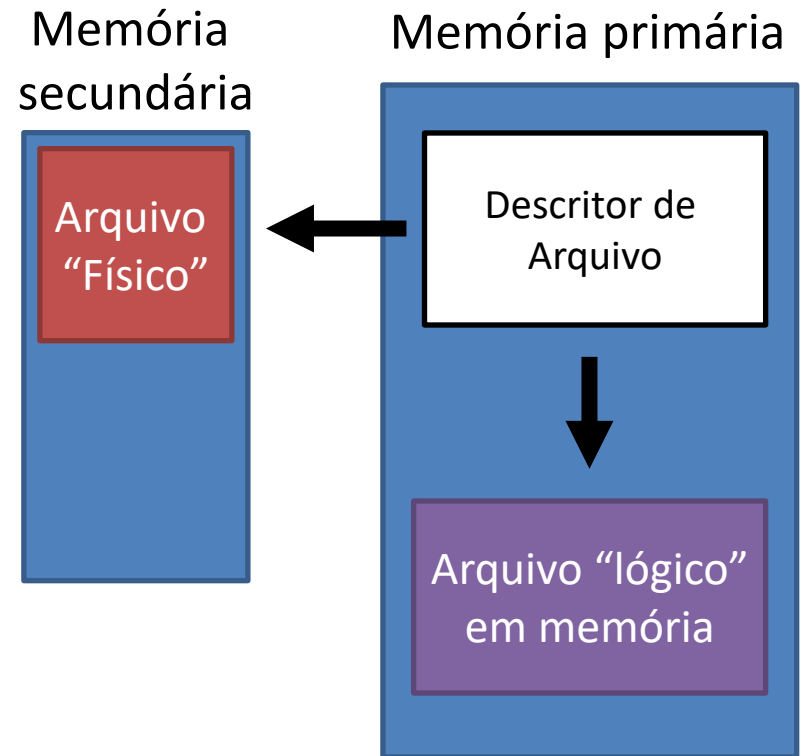
- Se for feita uma escrita de 8 bytes na sequência, o ponteiro de L/E se moverá para a próxima posição de L/E (**byte 17**)

T	h	i	s		i	s		a		s	a	m	p	l	e	.	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

ponteiro de L/E

# Arquivo lógico

- O arquivo lógico é identificado pelo S.O. por um número inteiro chamado de descriptor
- É comum que as linguagens encapsulem o descriptor do arquivo em estruturas mais complexas, que armazenam outras informações úteis para a manipulação do arquivo
  - O tipo dessa estrutura descritora é dependente da linguagem
  - Em Python, essas informações ficam em um **objeto de arquivo**





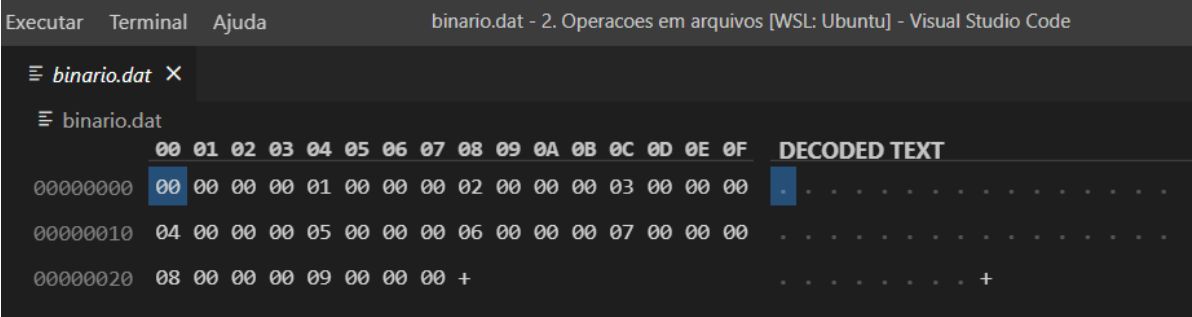
# Tipos de arquivos

- Basicamente, são dois os tipos de arquivos
  - **Texto**
    - Os caracteres são armazenados sequencialmente
    - É possível determinar o primeiro, segundo, terceiro ... caracteres que compõem o arquivo
    - Nesse tipo de arquivo, a sequência 123 corresponde à string “123”
  - **Binário**
    - Formado por uma sequência de bytes sem correspondência direta com um tipo de dado
    - Cabe ao programador fazer essa correspondência quando lê e escreve nesses arquivos
    - Nesse tipo de arquivo, o valor 123 será gravado como um número binário com uma quantidade pré-definida de bytes

# Exemplo

- As figuras mostram um arquivo binário e um arquivo texto em um editor hexadecimal

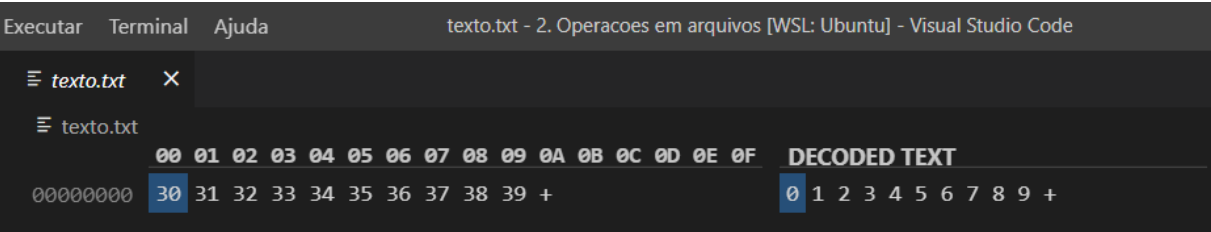
No arquivo binario.dat, foi gravada a sequência de dígitos de 0 a 9 como valores inteiros de 4 bytes



```
Executar  Terminal  Ajuda  binario.dat - 2. Operacoes em arquivos [WSL: Ubuntu] - Visual Studio Code

≡ binario.dat ×
≡ binario.dat
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  DECODED TEXT
00000000 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 . . . . .
00000010 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 . . . . .
00000020 08 00 00 00 09 00 00 00 + . . . . .
```

No arquivo texto.txt foi gravada a sequência de dígitos de 0 a 9 como texto → '0', '1', ..., '9'



```
Executar  Terminal  Ajuda  texto.txt - 2. Operacoes em arquivos [WSL: Ubuntu] - Visual Studio Code

≡ texto.txt ×
≡ texto.txt
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  DECODED TEXT
00000000 30 31 32 33 34 35 36 37 38 39 + 0 1 2 3 4 5 6 7 8 9 +
```

# Operações em arquivos

- Basicamente, as operações em arquivos são as seguintes:
  1. Criar um arquivo
  2. Abrir um arquivo
  3. Ler de um arquivo
  4. Escrever em um arquivo
  5. Movimentar o ponteiro de L/E
  6. Fechar um arquivo
- Veremos como essas operações são feitas em **Python**

# Operações em arquivos

- As funções da linguagem Python para manipular arquivos são implementadas de duas formas:
  - Como função nativa
  - Como **métodos** de objetos de arquivo
- A função ***open*** é nativa e utilizada para abrir/criar um arquivo, retornando um **objeto de arquivo**
  - A classe desse objeto varia de acordo com o tipo de arquivo a ser manipulado (leitura/escrita textual, leitura/escrita binária, etc.)
  - A partir do objeto de arquivo é possível acessar **métodos** para leitura, escrita, posicionamento, etc.

**Métodos** são funções declaradas no contexto de uma classe e são acessadas por meio de um objeto da classe

# Operações em arquivos em Python

- A associação entre arquivo lógico e físico é feita no momento da abertura/criação do arquivo
  - O modo como a associação se dará depende do parâmetro utilizado para especificar o modo de abertura
  - O modo de abertura também especifica se um novo arquivo será criado ou se um arquivo existente será aberto

# Criar e abrir arquivos

- **def open** (filename: **str**, mode: **str**) -> *file object*
  - *filename* é uma *string* contendo o nome do arquivo físico
  - *mode* também é uma *string* e define o modo de abertura
  - **Caso a operação falhe, um erro ocorrerá e abortará a execução**
- Exemplo:  

```
arq = open('meuarq.txt', 'w')
```
- modos de abertura → 'r' (read), 'w' (write), 'a' (append), 'x' (exclusive),  
'r+', 'w+', 'a+', 'x+' 'rb', 'wb', 'ab', 'r+b', 'w+b', 'x+b', 'a+b'
- Por padrão, os arquivos são abertos em modo texto (*flag* 't')
  - A *flag* 'b' na string *mode* faz com que os arquivos sejam abertos em modo binário

# *flags* da função `open()`

Caractere	Descrição do comportamento
'r'	Abre para leitura. O arquivo deve existir. <b>É o modo padrão.</b>
'w'	Abre para escrita. Se o arquivo existir, ele será truncado. Senão será criado.
'x'	Abre um novo arquivo para escrita. O arquivo não deve existir.
'a'	Abre para escrita no final do arquivo. Operações de reposicionamento do ponteiro de L/E são ignoradas. Se não existir, o arquivo será criado.
't'	Abre em modo texto. Admite apenas dados do tipo <i>str</i> . <b>É o modo padrão.</b>
'b'	Abre em modo binário. Admite apenas dados do tipo <i>bytes/bytearray</i> .
'+'	Altera o comportamento da <i>flag</i> inicial para permitir leitura e escrita.

**Fique atento:** as flags 't', 'b' e '+' só podem ser utilizadas em conjunto com outras

Mais informações: <https://docs.python.org/3/library/functions.html#open>

# Criar e abrir arquivos

- Como tratar um erro na abertura do arquivo?
  - Em Python, erros de execução são chamados de **exceções**
  - As exceções podem ser capturadas e tratadas utilizando a instrução **try-except**

**try:**

'''código a ser executado'''

**except:**

'''código a ser executado caso uma exceção ocorra'''

- Exemplo:

**try:**

```
arq = open('meuArq.txt', 'r')  
print('abertura ok')
```

**except:**

```
print('Erro na abertura')  
exit()
```

} Este trecho será  
executado apenas se  
ocorrer uma exceção



# Fechamento de arquivos

- Uma vez que o arquivo foi aberto, o seu fechamento se dá pela chamada do método **close()**
  - Todos os objetos de arquivo possuem esse método
  - Encerra a associação entre o arquivo lógico e o físico
  - O descritor fica disponível para uso por outro arquivo
  - Garante que todas as informações do arquivo foram atualizadas (o fechamento garante que o conteúdo bufferizado foi enviado para o dispositivo físico)
  - O S.O. normalmente fecha qualquer arquivo que ficou aberto quando um programa termina corretamente, mas o ideal é fechar arquivos que não estão mais sendo utilizados
    - Previne a perda de dados bufferizados no caso de erro
    - Libera a estrutura descritora para novos arquivos

# Fechamento de arquivos

- Exemplo:

```
arq = open('meuarq.txt', 'r')  
'''comandos'''  
arq.close()
```

Sintaxe para a chamada de **métodos**: o nome do objeto seguido por ponto sempre precede o nome do método

- Quando o arquivo é aberto em um comando **with**, não é necessário o fechamento explícito

```
with open('meuarq.txt', 'r') as arq:  
    '''comandos'''
```

**Atenção:** neste caso, o objeto de arquivo **arq** só existirá no escopo do comando **with**

# Leitura e escrita em arquivos

- Em Python, as funções de leitura e escrita são métodos do objeto de arquivo
  - Portanto, dependem do tipo do objeto
- Arquivos em modo texto são representados por objetos do tipo **TextIOWrapper**
  - Só admitem leitura e escrita de strings, i.e., objetos do tipo *str*
- Arquivos em modo binário são representados por objetos **BufferedReader** (leitura), **BufferedWriter** (escrita) e **BufferedRandom** (leitura e escrita)
  - Só admitem leitura e escrita de objetos do tipo *bytes* ou *bytearray*

*bytes* é um tipo imutável e *bytearray* é um tipo mutável.  
Ambos representam sequências de bytes.

# Leitura em modo texto

- **def read(size=-1) -> str**
  - Lê e retorna no máximo *size* caracteres como uma única string
  - Se *size* for negativo ou None, lê até o fim do arquivo (EOF), i.e., retorna todo o conteúdo do arquivo como uma única string
- **def readline(size=-1) -> str**
  - Lê até uma quebra de linha ou EOF e retorna o conteúdo como uma única string
  - Se o ponteiro de L/E já estiver em EOF, retorna uma string vazia
  - Se *size* for especificado, no máximo *size* caracteres serão lidos
- **def readlines(hint=-1) -> list[str]**
  - Retorna as linhas do arquivo como uma lista de strings
  - Se *hint* for especificado, linhas serão lidas até o máximo de *hint* caracteres no total

# Escrita em modo texto

- **def write(s) -> int**
  - Escreve a string *s* no arquivo
  - Retorna o número de caracteres escritos
- **def writelines(lines) -> **None****
  - Escreve uma lista de strings no arquivo
  - Não são adicionados separadores entre as strings, então é comum que cada string contenha um caractere separador (p.e., '\n') no final

# Leitura em modo binário

- **def read(size=-1) -> bytes**
  - Lê e retorna até *size* bytes
  - Se o argumento for omitido, None ou negativo, o conteúdo do arquivo será lido até EOF e retornado.
  - Um objeto bytes vazio (b'') é retornado se o ponteiro de L/E estiver em EOF
- **def readinto(b) -> int**
  - Lê bytes para uma variável *b*, pré-alocada como um bytearray, e retorna o número de bytes lidos
- **def readlines(hint=-1) -> list[bytes]**
  - Retorna o conteúdo do arquivo como uma lista de linhas
  - Considera o valor b'\n' como delimitador de linha
  - Se *hint* for especificado, linhas serão lidas até o máximo de *hint* bytes no total

# Escrita em modo binário

- **def write(b) -> int**
  - Escreve o objeto *b* do tipo bytes ou bytearray no arquivo
  - Retorna o número de bytes escritos (sempre será igual ao comprimento de *b* em bytes, pois uma falha na escrita irá gerar uma exceção)
- **def writelines(lines) -> None**
  - Escreve uma lista de linhas no arquivo
  - Não são adicionados separadores entre as linhas, então é comum que cada linha seja adicionada de um caractere separador (p.e., b'\n')

Mais informações: <https://docs.python.org/3/library/io.html>

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```



# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Receba o nome do arquivo a ser aberto e armazene-o em **nomeArq**

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Abra o arquivo **nomeArq** para leitura em modo texto e chame-o de **arq**

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Leia um caractere de **arq** e armazeno-o em **c**.  
Quando não houverem mais caracteres a serem lidos, a função **read** retornará "" (string vazia)

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

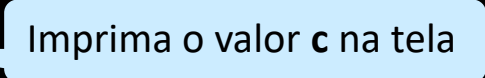
```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Enquanto `c` for verdadeiro.  
Equivalente a `while c != ""`:

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```



Imprima o valor c na tela

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Leia um novo caractere de **arq** e armazene-o em **c**.

# Exemplo leitura/escrita

Escreva um programa em Python que leia um dado arquivo texto caractere por caractere e os escreve na tela.

```
nomeArq = input('Digite o nome do arquivo: ')
arq = open(nomeArq, 'r')
c = arq.read(1)
while c:
    print(c)
    c = arq.read(1)
arq.close()
```

Feche arq

# Exemplo leitura/escrita

## Versão alternativa 1:

```
nomeArq = input('Digite o nome do arquivo: ')
with open(nomeArq, 'r') as arq:
    c = arq.read(1)
    while c:
        print(c)
        c = arq.read(1)
```

## Versão alternativa 2:

```
nomeArq = input('Digite o nome do arquivo: ')
try:
    with open(nomeArq, 'r') as arq:
        c = arq.read(1)
        while c:
            print(c)
            c = arq.read(1)
except:
    print(f'Não foi possível abrir o arquivo {nomeArq}.')
```



# Detecção de fim de arquivo

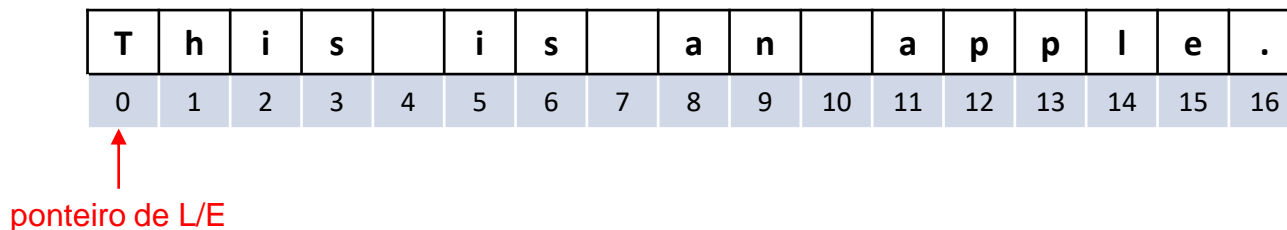
- Durante a leitura de um arquivo, o S.O. monitora a posição do ponteiro de L/E do arquivo
- Em Python, não existe uma função específica para isso, mas as funções de leitura detectam o fim de arquivo
  - Nesse caso, as funções de leitura retornam um objeto vazio, seja uma string (") ou um byte (b")

```
nomeArq = input('Digite o nome do arquivo: ')
with open(nomeArq, 'r') as arq:
    c = arq.read(1)
    while c:
        print(c)
        c = arq.read(1)
```

Neste exemplo, note que como controlamos a leitura até o fim do arquivo testando o retorno da função **read**.

# Posicionamento do ponteiro de L/E

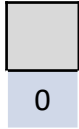
- Funções de leitura e escrita movimentam o ponteiro de L/E do arquivo de forma automática
  - O ponteiro de L/E sempre avança quando lê/escreve
- Podemos movimentar o ponteiro de L/E para uma posição específica do arquivo utilizando método **seek**
  - Chamamos o tamanho do deslocamento a ser realizado de **byte-offset**
  - Quando esse deslocamento é contado a partir do início do arquivo, podemos pensar no *offset* como um “endereço” para onde se deseja mover o ponteiro de L/E



# Posicionamento do ponteiro de L/E

- **def seek (*offset*, *whence*=os.SEEK\_SET) -> int**
  - Posiciona o ponteiro de L/E no byte *offset* calculado a partir da origem dada (*whence*)
  - Retorna o valor absoluto da nova posição
  - O parâmetro *whence* aceita os seguintes valores:
    - os.SEEK\_SET ou 0 → início do arquivo. **É o valor padrão**
    - os.SEEK\_CUR ou 1 → posição corrente do ponteiro de L/E
    - os.SEEK\_END ou 2 → fim do arquivo
- Arquivos em modo texto só aceitam **seek** com *offsets* positivos a partir de os.SEEK\_SET
  - Nesse caso, se *whence* for os.SEEK\_CUR ou os.SEEK\_END, o *offset* deverá ser 0

# Exemplo *seek*



ponteiro de L/E  
antes da 1ª escrita

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

O arquivo está sendo criado em **modo binário** para termos mais liberdade com o seek.

# Exemplo seek

T	h	i	s		i	s		a	n		a	p	p	l	e	.	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

↑  
ponteiro de L/E  
após a 1ª escrita

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

**encode** é um método da classe **str** que retorna a string codificada como bytes.

# Exemplo seek

T	h	i	s		i	s		a	n		a	p	p	l	e	.	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

↑  
ponteiro de L/E  
após o 1º seek

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

*Byte-offset = 9*

*Origem = 0 (SEEK\_SET)*

*Nova posição do ponteiro de L/E:*

*$0 + 9 = 9$*

# Exemplo seek

T	h	i	s		i	s		a		s	a	m	p	l	e	.	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

↑  
ponteiro de L/E  
após a 2ª escrita

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

*Escrita a partir da nova posição,  
avançando o ponteiro de L/E*

# Exemplo seek

T	h	i	s		i	s		a		s	a	m	p	l	e	.	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

↑  
ponteiro de L/E  
ANTES do seek

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

Qual será a nova posição  
do ponteiro de L/E?



# Exemplo seek

T	h	i	s		i	s		a		s	a	m	p	l	e	.				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

↑  
ponteiro de L/E  
após o seek

```
import os

file = open('example.txt', 'wb')
s = 'This is an apple.'
file.write(s.encode())
file.seek(9, os.SEEK_SET)
s = ' sam'
file.write(s.encode())
file.seek(3, os.SEEK_END)
s = 'ok'
file.write(s.encode())
file.close()
```

*offset = 3*

*whence = 17 (SEEK\_END)*

*Nova posição do ponteiro de L/E:*

*17 + 3 = 20*

*O ponteiro avançará 3 bytes à frente do fim (offset 17), mesmo não havendo nada lá.*

*Pararia no **offset 20**.*

**Offsets positivos SEMPRE avançam em direção ao fim do arquivo.**

# Exemplo seek

T	h	i	s		i	s		a		s	a	m	p	l	e	.	\0	\0	\0	0	k	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

↑  
ponteiro de L/E  
após a 3ª escrita

```
import os
```

```
file = open('example.txt', 'wb')
```

```
s = 'This is an apple.'
```

```
file.write(s.encode())
```

```
file.seek(9, os.SEEK_SET)
```

```
s = ' sam'
```

```
file.write(s.encode())
```

```
file.seek(3, os.SEEK_END)
```

```
s = 'ok'
```

```
file.write(s.encode())
```

```
file.close()
```

*As posições vazias são  
preenchidas com zeros*

# Posicionamento do ponteiro de L/E

- Como voltar o ponteiro de L/E para o início?
  - `arq.seek(0)`
- Como posicionar o ponteiro de L/E no fim?
  - `arq.seek(0, os.SEEK_END)`
- Como saber em qual posição o ponteiro de L/E está?
  - Todo objeto de arquivo possui um método **`tell`**
  - **`def tell () -> int`**
    - Retorna a posição atual do ponteiro de L/E como um número inteiro

T	h	i	s		i	s		a		s	a	m	p	l	e	.	\0	\0	\0	0	k	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

```
>>>file.tell()  
22
```

↑  
ponteiro de L/E  
após a 3ª escrita

# Arquivos e o S.O.

- Marcação de fim de linha
  - Dependendo do S.O. utilizado, a representação de fim de linha muda
    - DOS/Windows utiliza dois bytes
      - Par CR-LF (decimal ASCII 13 e 10 → hex 0D 0A)
    - Unix/Linux utiliza um byte
      - LF (decimal ASCII 10 → hex 0A)
  - Para vermos essa marcação precisamos usar um editor hexadecimal
    - <https://hexed.it/>
    - <https://www.onlinehexeditor.com/>

# Arquivos e o S.O.

- Visualização do arquivo em um editor de texto

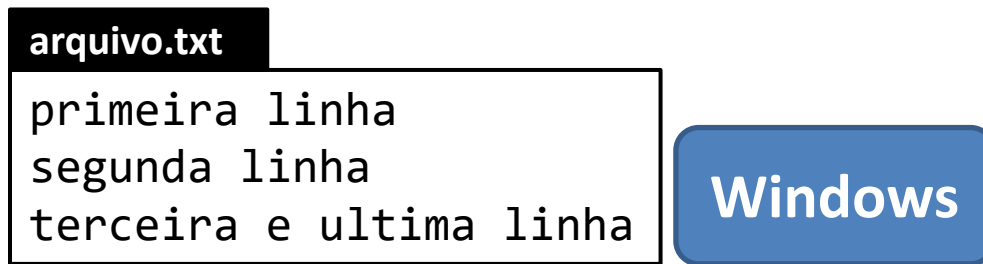


Tabela ASCII

Bin	Oct	Dec	Hex	Sinal
...				
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b
0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h

- Visualização do arquivo em um editor hexadecimal

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 70 72 69 6D 65 69 72 61 20 6C 69 6E 68 61 0D 0A
00000010: 73 65 67 75 6E 64 61 20 6C 69 6E 68 61 0D 0A 74
00000020: 65 72 63 65 69 72 61 20 65 20 75 6C 74 69 6D 61
00000030: 20 6C 69 6E 68 61
```

primeira.linha..  
segunda.linha..t  
erceira.e.ultima  
.linha

# Arquivos e o S.O.

- Visualização do arquivo em um editor de texto

**arquivo.txt**  
primeira linha  
segunda linha  
terceira e ultima linha

Linux

Tabela ASCII

Bin	Oct	Dec	Hex	Sinal
...				
0110 0000	140	96	60	`
0110 0001	141	97	61	a
0110 0010	142	98	62	b
0110 0011	143	99	63	c
0110 0100	144	100	64	d
0110 0101	145	101	65	e
0110 0110	146	102	66	f
0110 0111	147	103	67	g
0110 1000	150	104	68	h

- Visualização do arquivo em um editor hexadecimal

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 70 72 69 6D 65 69 72 61 20 6C 69 6E 68 61 0A 73
00000010: 65 67 75 6E 64 61 20 6C 69 6E 68 61 0A 74 65 72
00000020: 63 65 69 72 61 20 65 20 75 6C 74 69 6D 61 20 6C
00000030: 69 6E 68 61
```

'\n'

primeira.linha.s  
egunda.linha.ter  
ceira.e.ultima.l  
inha

# Arquivos binários em Python

- Os arquivos binários em Python realizam escrita e leitura apenas com os tipos *bytes* e *bytearray*
- Alguns tipos fornecem métodos de conversão prontos para os tipos binários
  - Conversão de inteiros:
    - `<int>.to_bytes(n)` → converte inteiro para n bytes
    - `int.from_bytes(<bytes>)` → converte bytes para inteiro
  - Conversão de strings:
    - `<str>.encode()` → converte string para bytes
    - `<bytes>.decode()` → converte bytes para string

# Outros exemplos

**arquivo.txt**

Olha que coisa mais linda,  
mais cheia de graca.

```
arq = open('arquivo.txt', 'r')
```

Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	TEXT	DECODE
00000000:	4F	6C	68	61	20	71	75	65	20	63	6F	69	73	61	20	6D	Olha	que
00000010:	61	69	73	20	6C	69	6E	64	61	2C	0A	6D	61	69	73	20	coisa	.
00000020:	63	68	65	69	61	20	64	65	20	67	72	61	63	61	2E		mais	.

cheia.de.graca.

```
s = arq.readline()  
print(s)
```

Olha que coisa mais linda,



# Outros exemplos

**arquivo.txt**

Olha que coisa mais linda,  
mais cheia de graca.

Leitura binária!

```
arq = open('arquivo.txt', 'rb')
```

Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	TEXTO DECODIFICADO
00000000:	4F	6C	68	61	20	71	75	65	20	63	6F	69	73	61	20	6D	Olha.que.coisa.m
00000010:	61	69	73	20	6C	69	6E	64	61	2C	0A	6D	61	69	73	20	ais.linda,.mais.
00000020:	63	68	65	69	61	20	64	65	20	67	72	61	63	61	2E		cheia.de.graca.

```
s = arq.readline()  
print(s)
```

b'Olha que coisa mais linda,\n'

# Outros exemplos

**arquivo.txt**

Olha que coisa mais linda,  
mais cheia de graca.

Leitura binária!

```
arq = open('arquivo.txt', 'rb')
```

Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	TEXTO DECODIFICADO
00000000:	4F	6C	68	61	20	71	75	65	20	63	6F	69	73	61	20	6D	Olha.que.coisa.m
00000010:	61	69	73	20	6C	69	6E	64	61	2C	0A	6D	61	69	73	20	ais.linda,mais.
00000020:	63	68	65	69	61	20	64	65	20	67	72	61	63	61	2E		cheia.de.graca.

```
s = arq.readline()  
print(s.decode())
```

Olha que coisa mais linda

# Outros exemplos

**arquivo.txt**

Olha que coisa mais linda,  
mais cheia de graca.

Leitura binária!

```
arq = open('arquivo.txt', 'rb')
```

Offset:	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	TEXTO DECODIFICADO
00000000:	4F	6C	68	61	20	71	75	65	20	63	6F	69	73	61	20	6D	Olha.que.coisa.m
00000010:	61	69	73	20	6C	69	6E	64	61	2C	0A	6D	61	69	73	20	ais.linda,mais.
00000020:	63	68	65	69	61	20	64	65	20	67	72	61	63	61	2E		cheia.de.graca.

v1: 0x4F = 79

v2: 0x6C = 108

```
v1 = int.from_bytes(arq.read(1))  
v2 = int.from_bytes(arq.read(1))  
print(v1 + v2)
```

187