centro universitário FEI

CC8210 – NCA210 Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

Arquivos

Arquivos

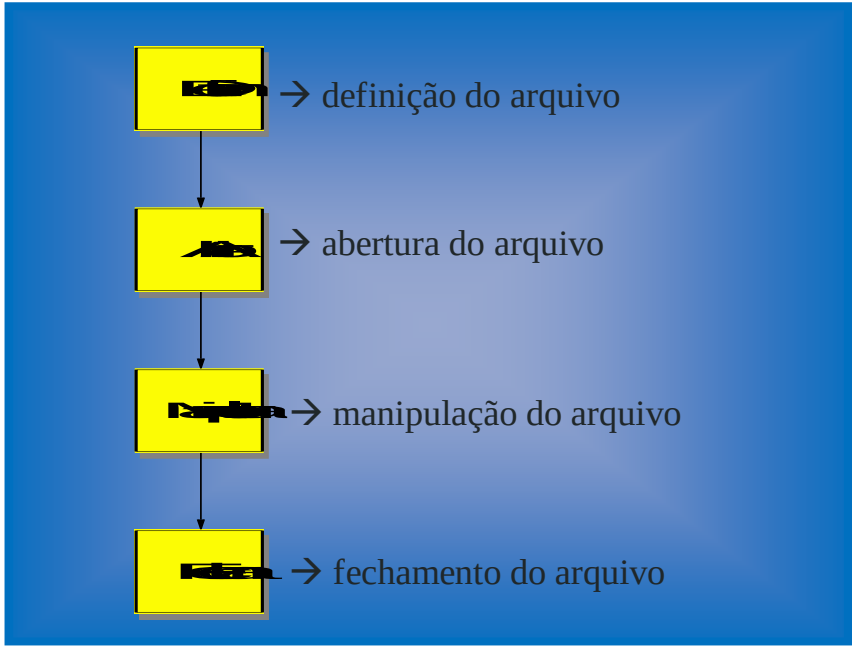
- Utilizar arquivos é uma forma de garantir o armazenamento permanente dos dados que são importantes no seu programa, pois nenhuma variável continua existindo depois que o programa termina.

- Então, utilizar um arquivo é uma maneira excelente de trabalhar com a entrada e a saída de dados para os programas.

Arquivos tipo Texto ou Binário

- Existem dois tipos de arquivos:
- Arquivos texto:
  - Arquivos contendo linhas de texto formadas por strings.
  - A codificação dos caracteres da string pode ser definido pela plataforma ou pelo programador
  - Normalmente salvos com a extensão .txt, .cvs ou .dat
- Arquivos binários:
  - Todos os dados são lidos e gravados exatamente como estão, em bytes, sem qualquer tratamento.

O tratamento de arquivos envolve os seguintes passos:



Abertura de Arquivos – `file.open()`

- Na programação, assim como na nossa interação com o computador, o primeiro passo para acessar um arquivo é abri-lo.
- Para abrir o arquivo, utilizamos a função open
- Sintaxe:

```
arquivo = open("teste.dat", "w")
```

- A variável arquivo salva o arquivo em si.
  - É por meio desta variável que executaremos as funções de escrita e leitura.

Abertura de Arquivos

open( ) tem dois parâmetros:

- nome do arquivo e modo de acesso.
- Modo de acesso:

```
arquivo = open("teste.dat", "w")
```

Modos de abertura de arquivos

- Os modos de acesso mais comuns são:

modo	operação
r	leitura (*read*)
w	escrita (*write*) – um arquivo pre-existente com o mesmo nome será apagado
a	escrita, preservando o conteúdo existente (*append*)
r+	abre o arquivo tanto para leitura como para escrita.

'b' appended to the mode opens the file in *binary mode*: now the data is read and written in the form of bytes objects.

Abrindo o arquivo com uma determinada codificação

- Por padrão, o Python abre o arquivo usando a codificação de caracteres padrão do sistema operacional.
- O terceiro parâmetro da função open() é opcional e nele especificamos a codificação do arquivo:

```
meuFile = open("teste.dat", "w",  encoding="utf-8")
```

Teste de abertura

- E se o arquivo não existir???
- Se o programa tentar abrir um arquivo inexistente para leitura, o interpretador retornará um erro:

```
FileNotFoundError: [Errno 2] No such file or directory: 'teste.dat’
```

- Soluções:
  - Verificar se o arquivo existe antes de abrir
  - Abrir usando controle de exceções

Verificar se o arquivo existe antes de abrir

```
import os
import sys
exists = os.path.isfile('teste.dat')
if exists == False:
    print ("O arquivo não existe. Saindo!")
    sys.exit()
```

Usar controle de exceções

- Não hoje...

```
try:
    report = open(“teste.dat”, 'w')
    report.write('some message')
except Exception as e:
    report.write('an error message')
finally:
    report.close()
```

Fechamento – file.close()

- Depois que escrevemos no arquivo, precisamos fechá-lo, utilizando o método close()

```
arquivo.close()
```

- É sempre importante fechar o arquivo para informar ao Sistema Operacional que não vamos mais utilizá-lo:
  - O Sistema Operacional salva as informações que queremos escrever em uma memória auxiliar chamada buffer e deixa a operação de escrever realmente no arquivo só quando informamos que vamos fechá-lo.

Fechamento

- Se não fechamos um arquivo, corremos o risco de perder o que gostaríamos de escrever.
- Ainda:
- Fechar um arquivo liberará os recursos que estavam vinculados ao arquivo.
    - Python tem um coletor de lixo para limpar objetos não-referenciados, mas não devemos confiar nele para fechar o arquivo.

Manipulação

- A manipulação dos arquivos constitui em ler, escrever, remover e inserir dados dos arquivos.
- Pode ser feita em arquivos texto e binários.
- Entrada:
  - `read(), readline(), readlines()`
- Saída:
  - `write(), writelines()`

Leitura

- Para ler do arquivo, precisamos seguir o procedimento:
  - Abrir o arquivo em modo leitura “r”

```
meuFile = open("teste.txt", "r")
```

- Utilizar um método para ler o arquivo.
- Fechar o arquivo com o método close:

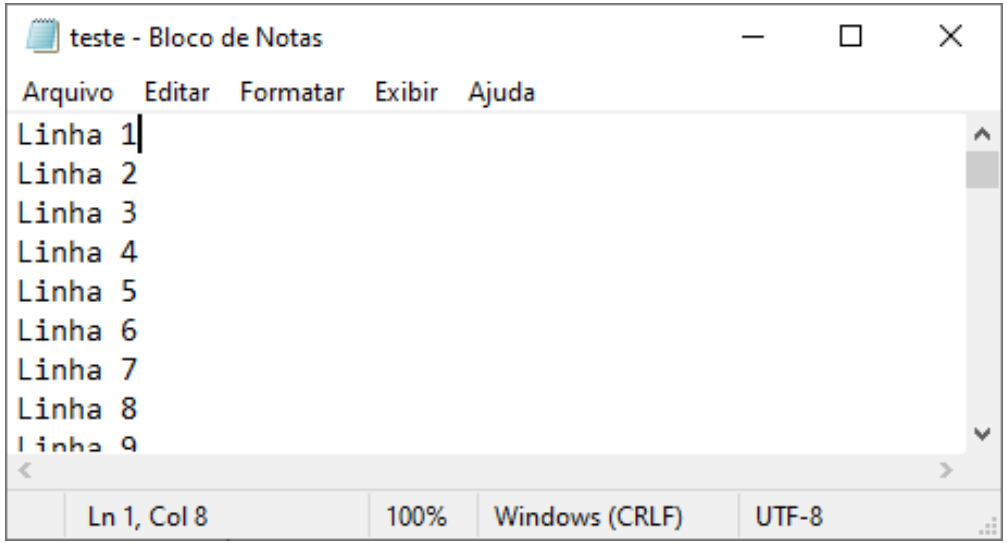
```
meuFile.close()
```

Leitura

Podemos usar os seguintes métodos:

- realine() – retorna uma string com uma linha
- read() – retorna uma string com tudo
- readlines() – retorna uma lista com todas as linhas como itens da lista.

Arquivo teste.txt



Leitura – file.readline()

- Para ler do arquivo, podemos utilizar o método readlines()
- Exemplo:

```
meuFile = open("teste.txt", "r")
for linha in meuFile.readlines():
    print(linha)
meuFile.close()
```

```
Linha 1

Linha 2

Linha 3

Linha 4

Linha 5

Linha 6
```

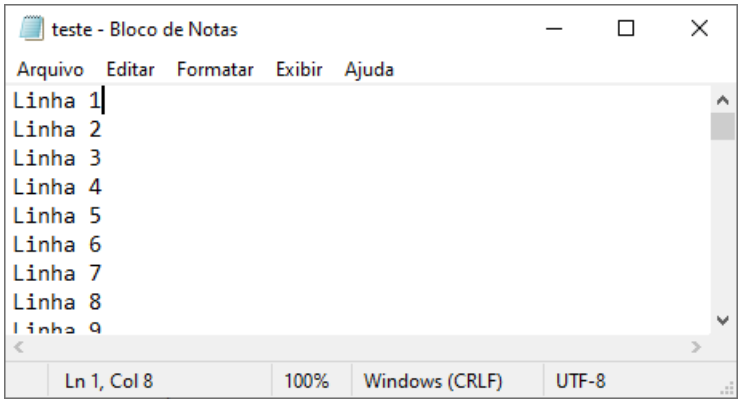
file.readline()

- f.readline() lê uma única linha do arquivo;
  - Um caracter newline (\n) é deixado no final da sequência, e só é omitido na última linha do arquivo se o arquivo não terminar em uma nova linha.
- Isso torna o valor de retorno inequívoco:
  - Se o final do arquivo foi atingido, f.readline() retorna uma string vazia.
  - Se for lida uma linha em branco, f.readline() retorna uma string contendo '\n'.

```
meuFile = open("teste.txt", "r")
for linha in meuFile.readlines():
    print(linha)
meuFile.close()
```

```
Linha 1

Linha 2

Linha 3
```

Pulou uma linha



Por que pula duas linhas?

- O programa anterior pula duas linhas a cada impressão:
  - Executa um newline devido ao '\n' que esta na string lida.
  - Executa um newline devido ao print.
- Como resolver isso?
- Já foi visto em aula:

```
print(linha, end = '')
```

Por que pula duas linhas?

```
meuFile = open("teste.txt", "r")
for linha in meuFile.readlines():
    print(linha, end="")
meuFile.close()
```

```
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
Linha 7
   ⋮
```

Simplificando a leitura

- Para ler linhas de um arquivo, você pode fazer loop sobre o objeto de arquivo.
- Isso é eficiente em memória, rápido e leva a um código simples.
- Usa o file.realine() implicitamente...

```
meuFile = open("teste.txt", "r")
for linha in meuFile:
    print(linha, end = '')
meuFile.close()
```

```
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
```

Leitura – file.read()

- Para ler o conteúdo de um arquivo, pode-se também utilizar f.read(size):
- Lê “size” quantidade de dados e os retorna como uma string (no modo de texto) ou bytes objeto (no modo binário).
- “size” é um argumento numérico opcional.
- Se o final do arquivo tiver sido atingido, f.read() retornará uma sequência vazia ('').

Leitura – file.read()

- Quando “size” for omitido ou negativo, todo o conteúdo do arquivo será lido e devolvido:
  - É seu problema se o arquivo for maior que a memória da sua máquina.

- Quando “size” for menor que o tamanho do arquivo:
  - a função “size” lê quantidade de dados e os retorna como uma string (no modo de texto) ou bytes objeto (no modo binário).

Leitura – file.read()

- Para ler do arquivo, podemos utilizar o método read()
- Exemplo:

```
meuFile = open("teste.txt", "r")
texto = meuFile.read()
print (texto)
meuFile.close()
```

```
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
   ⋮
```

Leitura – file.read()

- Para ler do arquivo, podemos utilizar o método read()
- Exemplo:

```
meuFile = open("teste.txt", "r")
texto = meuFile.read(10)
print (texto)
meuFile.close()
```

```
Linha 1
Li
```

Leitura – file.read()

- Para ler do arquivo, podemos utilizar o método read()
- Exemplo:

```
meuFile = open("teste.txt", "r")
texto = meuFile.read(20)
print (texto)
meuFile.close()
```

```
Linha 1
Linha 2
Linh
```

Leitura – file.read()

- Para ler do arquivo, podemos utilizar o método read()
- Exemplo:

```
meuFile = open("teste.txt", "r")
texto = meuFile.read(50)
print (texto)
meuFile.close()
```

```
Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
Li
```

Leitura – file.readlines()

- Devolve todas as linhas do arquivo, como uma lista onde cada linha é um item no objeto da lista.

```
meuFile = open("teste.txt", "r")
texto = meuFile.readlines()
print (texto)
meuFile.close()
```

```
['Linha 1\n', 'Linha 2\n', 'Linha 3\n', 'Linha 4\n',
'Linha 5\n', 'Linha 6\n', 'Linha 7\n', 'Linha 8\n',
'Linha 9\n', 'Linha 10\n', 'Linha 11\n', 'Linha 12
\n', 'Linha 13\n', 'Linha 14\n', 'Linha 15\n', 'Linha
1\n', 'Linha 2\n', 'Linha 3\n', 'Linha 4\n', 'Linha 5
\n', 'Linha 6\n', 'Linha 7\n', 'Linha 8\n', 'Linha 9
\n', 'Linha 10\n', 'Linha 11\n', 'Linha 12\n', 'Linha
```

Escrita – file.write()

- Para escrever no arquivo, utilizamos o método write():

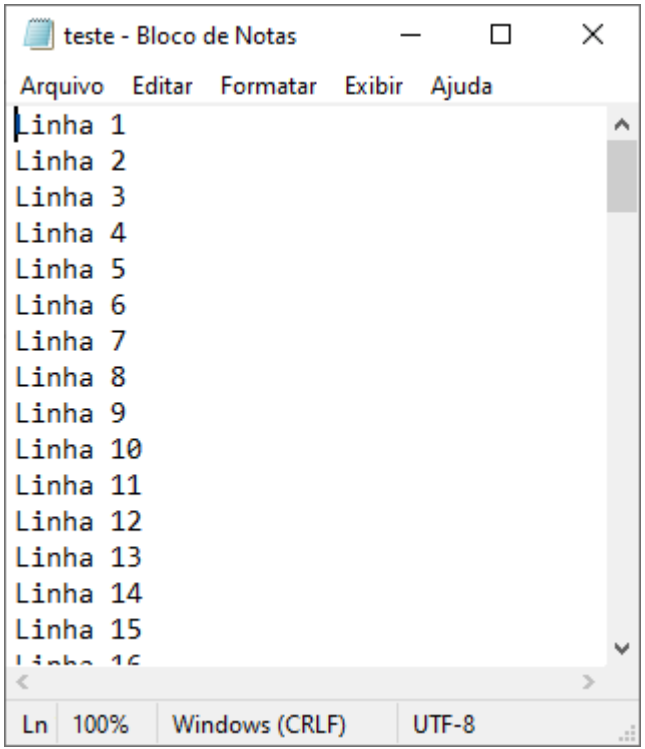
```
file.write("Texto a ser escrito no arquivo")
```

O método write funciona de maneira similar que o print com marcadores (%d, %f, %s),

  - IMPORTANTE!
  - Precisamos sempre incluir o "\n" quando queremos ir para a próxima linha.
- Retorna o número de caracteres escrito no arquivo

Escrita – file.write()

```
meuFile = open("teste.txt", "w")
for linha in range (1, 101):
    meuFile.write('Linha %d\n' % linha)
meuFile.close()
```



Escrita – file.write()

```
meuFile = open("teste.txt", "w")
for linha in range (1, 101):
    meuFile.write(linha)
meuFile.close()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-a1f8e4a9eb18> in <module>
      1 meuFile = open("teste.txt", "w")
      2 for linha in range (1, 101):
----> 3     meuFile.write(linha)
      4 meuFile.close()

TypeError: write() argument must be str, not int
```

Erro...

- f.write(string) writes the contents of *string* to the file, returning the number of characters written.
- Other types of objects need to be converted before writing them :
  - to a string (in text mode) or
  - a bytes object (in binary mode).

Escrita – file.write()

```
meuFile = open("teste.txt", "w")
for linha in range (1, 101):
    meuFile.write(str(linha))
meuFile.close()
```

Arquivo: teste.txt:

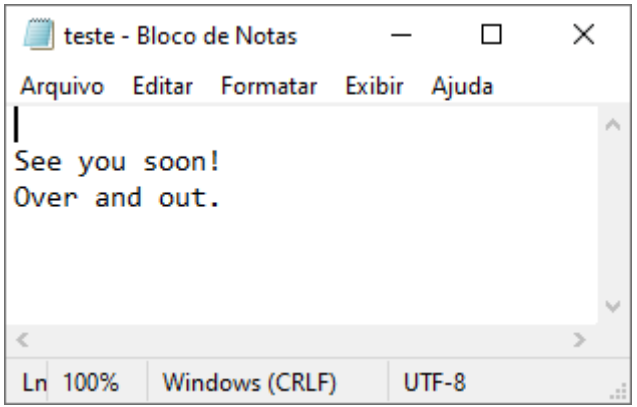
```
12345678910111213141516171819202122232425262
728293031323334353637383940414243444546474 84
950515253545556575859606162636465666768697 07
172737475767778798081828384858687888990919 29
3949596979899100
```

Escrita – file.writelines()

- O método writelines() grava os itens de uma lista para o arquivo.
- Onde os textos serão inseridos depende do modo de abertura do arquivo:
  - "a": Os textos serão inseridos na posição atual do arquivos, por padrão no final do arquivo.
  - "w": O arquivo será esvaziado antes que os textos sejam inseridos na posição inicial do arquivos.

Escrita – file.writelines()

```
meuFile = open("teste.txt", "w")
meuFile.writelines(["\nSee you soon!", "\nOver and out."])
meuFile.close()
```



Escrita - Exemplo

- Exemplo: gerar e gravar números pares e ímpares em arquivos separados. Números de 0 a 999.

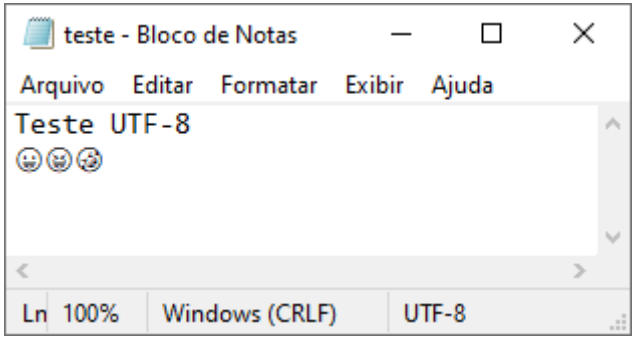
```
impares = open("impares.txt", "w")

pares = open("pares.txt", "w")

for n in range(1000):
    if n % 2 == 0:
        pares.write("%d\n" % n)
    else:
        impares.write("%d\n" % n)
impares.close()
pares.close()
```

Arquivos - Codificação

- Python 3 sempre armazena strings de texto como sequências em codificação Unicode.
  - Estes são valores na faixa 0-0x10FFFF.
  - Eles nem sempre correspondem diretamente aos caracteres que você lê na tela, mas essa distinção não importa para a maioria das tarefas de manipulação de texto.
- Para armazenar texto em outro formato, você deve especificar uma codificação para esse arquivo usando “encoding = xxx”.

Escrita – Arquivo Unicode

```
meuFile = open("teste.txt", "w", encoding="utf-8")
meuFile.write('Teste UTF-8\n')
# grinning face
meuFile.write("\U0001f600")
# grinning squinting face
meuFile.write("\U0001F606")
# rolling on the floor laughing
meuFile.write("\U0001F923")
meuFile.close()
```



Escrita – Arquivo ASCII

```
meuFile = open("teste.txt", "w", encoding="ascii")
meuFile.write('Teste UTF-8\n')
# grinning face
meuFile.write("\U0001f600")
# grinning squinting face
meuFile.write("\U0001F606")
# rolling on the floor laughing
meuFile.write("\U0001F923")
meuFile.close()
```

```
---------------------------------------------------------------------------
UnicodeEncodeError                        Traceback (most recent call last)
<ipython-input-42-85717a4bdcf7> in <module>
      2 meuFile.write('Teste UTF-8\n')
      3 # grinning face
----> 4 meuFile.write("\U0001f600")
      5 # grinning squinting face
      6 meuFile.write("\U0001F606")

UnicodeEncodeError: 'ascii' codec can't encode character '\U0001f600' in po
sition 0: ordinal not in range(128)
```

Arquivos

- Podemos realizar diversas operações com os arquivos
- Por exemplo:
  - Ler
  - Processar
  - Gerar novos arquivos

Arquivos

- Exemplo: Utilizando o arquivo *"pares.txt",* gerado no último exemplo, vamos criar outro arquivo que deve conter somente os números múltiplos de 4.

```
multiplos4 = open("multiplos_4.txt", "w")

pares = open("pares.txt", "r")

for linha in pares.readlines():
    if int(linha) % 4 == 0:
            multiplos4.write(linha)
pares.close()
multiplos4.close()
```

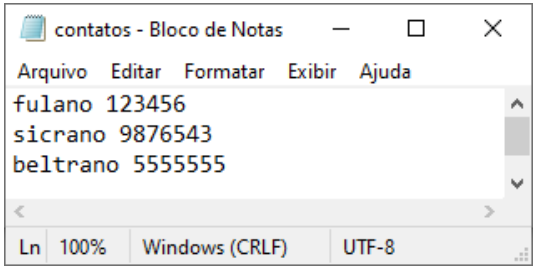
Arquivos

- Até agora, estamos utilizando somente um dado por linha.
- Porém, podemos salvar informações correlatas na mesma linha.
- Exemplo:
  - Criar um arquivo com o nome e o telefone de pessoas, conforme são digitados pelo usuário.
  - O programa deve funcionar em loop até que o nome digitado seja vazio.

Arquivos - Escrita de dois dados na mesma linha

```
1  contatos = open("contatos.dat", "w")
2  nome = input("Nome: ")
3  telefone = input("Telefone: ")
4
5  while nome != "":
6      contatos = open("contatos.dat", "a")
7      contatos.write("%s %s\n" % (nome, telefone))
8      contatos.close()
9      nome = input("Nome: ")
10     telefone = input("Telefone: ")
11
```

```
Nome: fulano
Telefone: 123456
Nome: sicrano
Telefone: 9876543
Nome: beltrano
Telefone: 5555555
```



Arquivos - Leitura de dois dados na mesma linha

- Entendendo melhor o readlines()
  - O readlines() retorna uma lista onde cada uma das linhas ocupa uma posição/ índice:

```
1  contatos = open("contatos.dat", "r")
2
3  conteudo_do_arquivo = contatos.readlines()
4
5  print(conteudo_do_arquivo)
```

```
['fulano 123456\n', 'sicrano 9876543\n', 'beltrano 5555555\n']
```

Método split(x)

- Divide as informações no caractere informado como parâmetro
- Exemplo:

```
nome, telefone = input("Entre com o nome e o telefone: ").split(" ")

print(nome)
print(telefone)
```

```
Entre com o nome e o telefone: fulano 1234567
fulano
1234567
```

Arquivos

- Como ler uma linha com duas informações?

```
1  contatos = open("contatos.dat", "r")
2
3  contato = []
4
5  for linha in contatos.readlines():
6      linha_separada = linha.split(" ")
7      contato.append(linha_separada)
8
9  print(contato)
10 print(contato[0])
11 print(contato[0][0])
12 print(contato[0][1])
```

Lista de listas

```
[['fulano', '123456\n'], ['sicrano', '9876543\n'], ['beltrano', '5555555\n']]
['fulano', '123456\n']
fulano
123456
```

Python File Methods

Function	Explanation
open()	To open a file
close()	Close an open file
fileno()	Returns an integer number of the file
read(n)	Reads ‘n’ characters from the file till end of the file
readable()	Returns true if the file is readable
readline()	Read and return one line from the file
readlines()	Reads and returns all the lines from the file
seek(offset)	Change the cursor position by bytes as specified by the offset
seekable()	Returns true if the file supports random access
tell()	Returns the current file location
writable()	Returns true if the file is writable
write()	Writes a string of data to the file
writelines()	Writes a list of data to the file

https://www.softwaretestinghelp.com/python/python-file-reading-writing/#Python_File_Methods

Python File Methods

Function	Explanation
open() ✔	To open a file
close() ✔	Close an open file
fileno()	Returns an integer number of the file
read(n) ✔	Reads ‘n’ characters from the file till end of the file
readable()	Returns true if the file is readable
readline() ✔	Read and return one line from the file
readlines() ✔	Reads and returns all the lines from the file
seek(offset)	Change the cursor position by bytes as specified by the offset
seekable()	Returns true if the file supports random access
tell()	Returns the current file location
writable()	Returns true if the file is writable
write() ✔	Writes a string of data to the file
writelines() ✔	Writes a list of data to the file

https://www.softwaretestinghelp.com/python/python-file-reading-writing/#Python_File_Methods

file.readable() e file.writable()

```
f = open("teste.txt", "r")
print("O arquivo é legivel: ", f.readable())
print("O arquivo é escrevivel: ", f.writable())
f.close()
```

```
O arquivo é legivel:  True
O arquivo é escrevivel:  False
```

file.readable() e file.writable()

```
f = open("teste.txt", "w")
print("O arquivo é legivel: ", f.readable())
print("O arquivo é escrevivel: ", f.writable())
f.close()
```

```
O arquivo é legivel:  False
O arquivo é escrevivel:  True
```

file.seek() e file.seekable()

```
f = open("teste.txt", "r")
print("O arquivo é buscavel: ", f.seekable())
f.seek(0)
print (f.readline())
f.seek(1)
print (f.readline())
f.seek(2)
print (f.readline())
print (f.readline())
f.seek(102)
print (f.readline())
f.seek(0)
print (f.readline())
f.close()
```

Saída

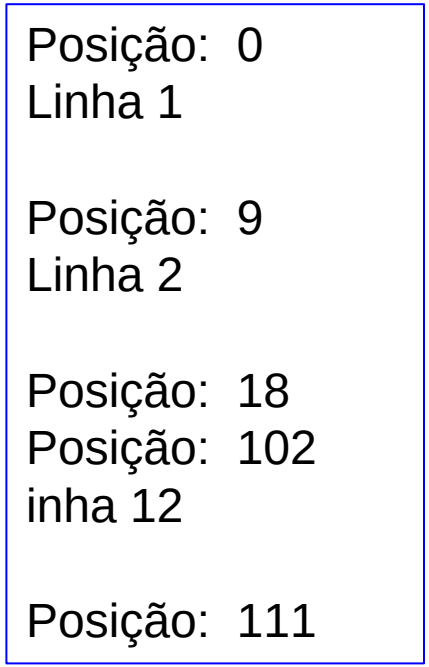
```
O arquivo é buscavel:  True
Linha 1
inha 1
nha 1
Linha 2
inha 12
Linha 1
```

file.seek()

- To change the file object's position, use f.seek(offset, whence).
- The position is computed from adding *offset* to a reference point.
- The reference point is selected by the *whence* argument.
  - A *whence* value of 0 uses the beginning of the file (default).
  - 1 uses the current file position.
  - 2 uses the end of the file as the reference point.

file.tell()

```
f = open("teste.txt", "r")
print("Posição: ", f.tell())
print (f.readline())
print("Posição: ", f.tell())
print (f.readline())
print("Posição: ", f.tell())
f.seek(100)
print("Posição: ", f.tell())
print (f.readline())
print("Posição: ", f.tell())
f.close()
```



Arquivos binários

- Um arquivo binário é qualquer arquivo que não seja arquivo de texto padrão.
- Os arquivos binários somente devem ser processados por uma aplicação que compreenda a estrutura do arquivo.
- São abertos adicionando “b” durante a chamada da função open().
- São manipulados utilizando read(), write(), seek() e tell().

Escrevendo arquivos binários – file.write()

```
matrix = [[1, 2, 3] , [4, 5, 6], [7, 8, 9]]
arquivo_novo = open('arquivo_matriz.bin','wb')
for linha in matrix:
    for elemento in linha:
        arquivo_novo.write(elemento)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-8d425e56fa47> in <module>
      3 for linha in matrix:
      4     for elemento in linha:
----> 5         arquivo_novo.write(elemento)

TypeError: a bytes-like object is required, not 'int'
```

Escrevendo arquivos binários – file.write()

```
matrix = [[1, 2, 3] , [4, 5, 6], [7, 8, 9]]
arquivo_novo = open('arquivo_matriz.bin','wb')
for linha in matrix:
    for elemento in linha:
        arquivo_novo.write(elemento.to_bytes(4, byteorder='big'))
```

Lendo arquivos binários

```
arquivo_novo = open('arquivo_matriz.bin','rb')
for linha in matrix:
    for elemento in linha:
        print(arquivo_novo.read(4))
arquivo_novo.close()
```

```
b'\x00\x00\x00\x01'
b'\x00\x00\x00\x02'
b'\x00\x00\x00\x03'
b'\x00\x00\x00\x04'
b'\x00\x00\x00\x05'
b'\x00\x00\x00\x06'
b'\x00\x00\x00\x07'
b'\x00\x00\x00\x08'
b'\x00\x00\x00\t'
```

Tabela ASCII - 7 bits

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Lendo arquivos binários

```
arquivo_novo = open('arquivo_matriz.bin','rb')
for linha in matrix:
    for elemento in linha:
        x = arquivo_novo.read(4)
        print(int.from_bytes(x, byteorder='big'))
arquivo_novo.close()
```

```
1
2
3
4
5
6
7
8
9
```

Conclusão

- Arquivos provém uma maneira simples de manter os dados guardados de maneira permanente no computador.
- Em Python, é muito simples abrir, fechar, manipular arquivos.
- Arquivos binários exigem o conhecimento da estrutura do arquivo.
  - Outras bibliotecas provém melhores ferramentas para manipulação de arquivos binários – JSON, PANDAS, NumPy.

Exercícios

Exercício 01

Crie um programa que inverta a ordem das linhas do arquivo pares.txt. A primeira linha deve conter o maior número e a última linha o menor. Salve o resultado em outro arquivo, chamado pares_invertido.txt.

pares.txt

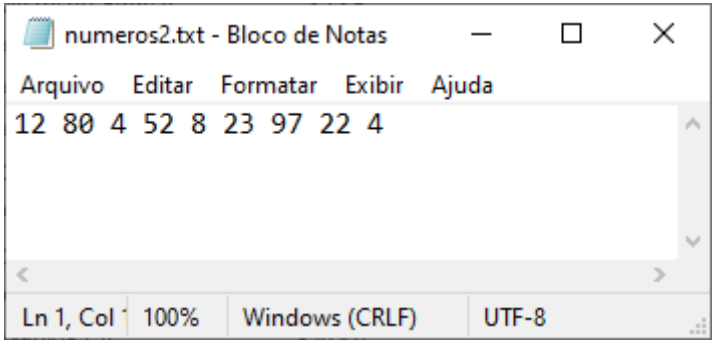
```
0
2
4
6
8
10
.
.
.
998
```

pares_invertido.txt

```
998
996
994
992
990
988
.
.
.
0
```

Exercício 02

Escreva uma função em Python para retornar a somatória de todos os números que estão armazenados no arquivo “numeros2.txt”. Todos os números do arquivo estão na mesma e única linha, separados por espaço.



Exercício 03

Escreva uma função que leia uma sequência numérica do arquivo “numeros3.txt” e salva os números na lista num. Esta função deve retornar num. Escreva outra função que recebe a lista num como parâmetro e retorna uma nova lista num_unicos, sem os elementos repetidos. Escreva uma terceira função que recebe a lista num_unicos e grava os números no arquivo “numeros3unicos.txt”