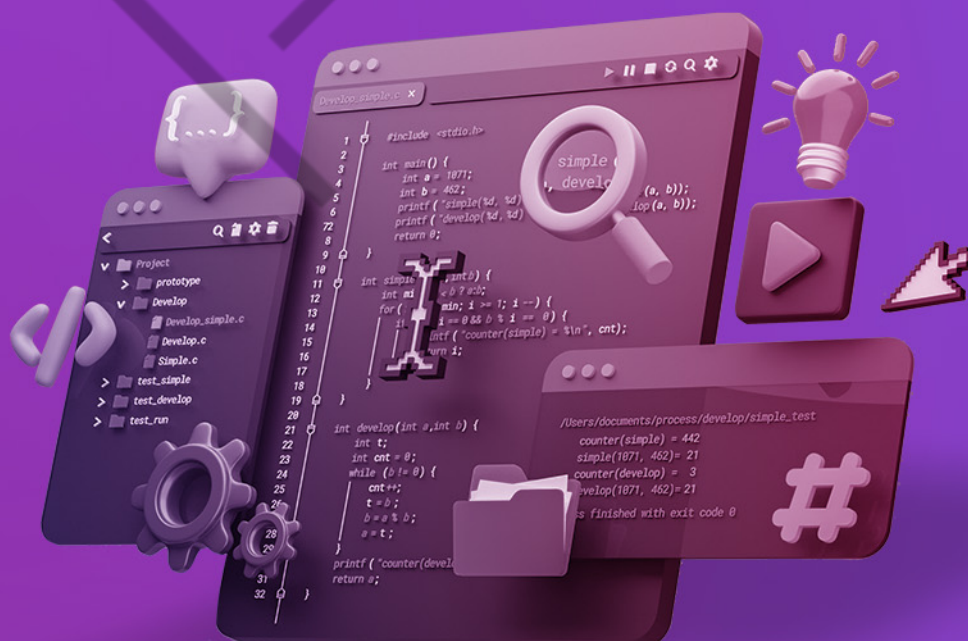


PRIMEIRO PASSO PARA FINALIZAR
O PRIMEIRO ANO

MANIPULAÇÃO DE ARQUIVOS TEXTOS E JSON



5

LISTA DE FIGURAS

Figura 1 – Tabela ASCII	10
Figura 2 – Tabela ASCII	16
Figura 3 – Conteúdo do arquivo.txt vazio com o cursor no início	17
Figura 4 – Conteúdo do arquivo.txt com texto gravado e cursor no final do arquivo	17
Figura 5 – Exibição do conteúdo do arquivo.txt	19
Figura 6 – Exibição do conteúdo do arquivo.txt com o cursor a partir da posição 9.	20

EDSON

LISTA DE TABELAS

Tabela 1 – Caracteres de abertura de arquivo texto.....	10
---	----

EXEMPLO

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Sintaxe da função open().....	11
Código-fonte 2 – Exemplo de abertura e fechamento de arquivo.....	11
Código-fonte 3 – Abertura de arquivo em modo escrita.....	12
Código-fonte 4 – Abertura de arquivo em modo leitura	13
Código-fonte 5 – Abertura de arquivo em modo de edição (append)	14
Código-fonte 6 – Conteúdo do arquivo.txt depois do append	14
Código-fonte 7 – Utilizando o modo de abertura x (exclusivo).....	15
Código-fonte 8 – Utilizando o modo de abertura + (leitura e escrita).....	16
Código-fonte 9 – Abertura para escrita e leitura	17
Código-fonte 10 – Gravando duas linhas.....	17
Código-fonte 11 – Exibindo e fechando o arquivo	18
Código-fonte 12 – Aplicando o seek() no programa.....	18
Código-fonte 13 – Aplicando o seek(9) no programa.....	19
Código-fonte 14 – Aplicando o método readline().....	21
Código-fonte 15 – Aplicando o método readline() para exibir três linhas.....	22
Código-fonte 16 – Gravações das linhas com quebras	23
Código-fonte 17 – Gravações das linhas sem quebras	23
Código-fonte 18 – Leitura das linhas sem quebras.....	24
Código-fonte 19 – Código completo aplicando readline()	24
Código-fonte 20 – Código completo aplicando readlines()	26
Código-fonte 21 – Parte do código aplicando readlines()	26
Código-fonte 22 – Exibindo o conteúdo a lista gerada pelo readlines().....	26
Código-fonte 23 – Exibindo o conteúdo a lista gerada pelo readlines().....	27
Código-fonte 24 – Criação do procedimento exibe_linhas_impares()	27
Código-fonte 25 – Código utilizando writelines().....	29
Código-fonte 26 – Código utilizando writelines().....	30
Código-fonte 27 – Aplicando o gerenciador de contexto with	30
Código-fonte 28 – Aplicando o gerenciador de contexto with gravando com Lista...	31
Código-fonte 29 – Arquivo XML	32
Código-fonte 30 – Arquivo JSON	33
Código-fonte 31 – Código para criar o dicionário pessoas	34
Código-fonte 32 – Exibição do dicionário pessoas	34
Código-fonte 33 – Importando a biblioteca JSON	35
Código-fonte 34 – Aplicando o método dumps().....	36
Código-fonte 35 – linha de aplicação do método dumps().....	37
Código-fonte 36 – Gravação do arquivo JSON.....	38
Código-fonte 37 – Carregando dados do arquivo JSON.....	38
Código-fonte 38 – Código de exibição formatada do Dicionário	40
Código-fonte 39 – Código fonte JSON na íntegra.....	42

LISTA DE COMANDOS DE PROMPT DO SISTEMA OPERACIONAL

Comando de prompt 1 – Conteúdo do arquivo.txt	12
Comando de prompt 2 – Exibição do conteúdo do arquivo.txt.....	13
Comando de prompt 3 – Exibição do conteúdo do arquivo.txt editado	14
Comando de prompt 4 – Exibição da falha na abertura em modo 'x'	15
Comando de prompt 5 – Exibição do conteúdo do arquivo.txt na tela	19
Comando de prompt 6 – Execução do programa com readline()	21
Comando de prompt 7 – Execução do programa com espaçamento duplo entre linhas	22
Comando de prompt 8 – Execução do programa sem espaçamento duplo entre linhas	25
Comando de prompt 9 – Exibição da Lista lista_linhas_arquivo	26
Comando de prompt 10 – Exibição da terceira linha da Lista lista_linhas_arquivo ..	27
Comando de prompt 11 – Exibição das linhas ímpares	28
Comando de prompt 12 – Exibição do dicionário pessoas	35
Comando de prompt 13 – Exibição do dicionário o objeto JSON	36
Comando de prompt 14 – Exibição objeto JSON.....	37
Comando de prompt 15 – Exibição dos dados do dicionário na tela	39
Comando de prompt 16 – Exibição dos dados do objeto JSON	39
Comando de prompt 17 – Exibição dos dados do objeto JSON formatado	41
Comando de prompt 18 – Exibição na íntegra – parte 1/2.....	43
Comando de prompt 19 – Exibição na íntegra – parte 2/2.....	44

SUMÁRIO

1 MANIPULAÇÃO DE ARQUIVOS TEXTOS E JSON	7
1.1 Gravação magnética de dados	7
1.2 Antes uma história.....	7
1.3 Gravar ou não gravar? Eis a questão	8
2 MANIPULAÇÃO DE ARQUIVOS TEXTO	9
2.1 Formas de abertura de arquivos	10
2.1.1 Função open()	10
2.1.2 Abertura modo escrita write ('w')	11
2.1.3 Abertura modo leitura read ('r')	12
2.1.4 Abertura modo edição append ('a')	13
2.1.5 Abertura modo escrita write ('x')	14
2.1.6 Abertura modo escrita + leitura write + read ('+')	15
2.1.7 Utilizando o método seek()	16
2.2 Outras formas de leitura e escrita de arquivos.....	20
2.2.1 Método de leitura readline()	20
2.2.2 Método de leitura readlines()	25
2.2.3 Método de escrita writelines()	28
2.3 Gerenciador de contexto with.....	29
3 UTILIZAÇÃO DE ARQUIVOS .JSON.....	32
3.1 Antes um pouco de XML	32
3.2 Definição de arquivo JSON	33
3.3 Criando uma aplicação que gera arquivo JSON	33
3.3.1 Definindo um dicionário	34
3.3.2 Importando a biblioteca JSON	35
3.3.3 JSON: Método dumps()	35
3.3.4 JSON: Método dumps() com o parâmetro indent	36
3.3.5 JSON: Gravando no arquivo	37
3.3.6 JSON: Carregando dados do arquivo	38
3.3.7 JSON: Exibindo os dados do arquivo na tela	38
3.3.8 JSON: Exibindo os dados formatados na tela	40
3.3.9 Código JSON na íntegra	41
REFERÊNCIAS	46
GLOSSÁRIO	47

1 MANIPULAÇÃO DE ARQUIVOS TEXTOS E JSON

1.1 Gravação magnética de dados

No mundo da programação, existem três pilares principais: Entrada, Saída e Processamento de dados. O último é complexo porque abrange muitas coisas além de processamento, como registradores, memórias, armazenamento, entre outros.

Até então, trabalhamos estes pilares em CTP – Python. Neste capítulo, vamos detalhar um dos itens do pilar de processamento, que é o armazenamento de dados.

1.2 Antes uma história

Por que o armazenamento de dados é importante?

Vou explicar contando uma história (verídica).

Em 1988, no auge dos meus 14 anos, apesar de tecnologia não ser algo tão acessível como hoje, eu tinha acesso a um terminal, disponibilizado pela Telesp (empresa onde meu pai trabalhava), com a assinatura do serviço chamado “vídeo texto”.

Lembro-me que para utilizar este serviço eu tinha que discar para um número de telefone fornecido pela TELESP – através de um telefone fixo (óbvio) – e, quando o sinal soasse (na maioria das vezes dava ocupado), eu deveria apertar o botão do modem e depois colocar o telefone no gancho (hoje este termo soa estranho) para que o serviço “vídeo texto” fosse conectado.

Conectado, ele abria um ambiente texto com alguns itens de menu como esportes, notícias, bate-papo (neste último só podiam entrar 20 pessoas - de todo Brasil - no máximo) entre outros, mas ainda não era a Internet.

Neste ambiente podíamos navegar sobre os itens do menu e usufruir dos serviços.

Para quem tivesse conhecimento em programação, o serviço tinha à disposição a linguagem de programação BASIC e ali o programador (que ainda não era o meu caso) poderia desenvolver os seus códigos e executá-los.

Junto com o kit do Terminal, vinham livros e um deles continha códigos-fontes prontos de jogos. Com este livro em mãos, o jogo que eu quisesse jogar deveria ser digitado, ele tinha centenas de linhas de código [levava horas para digitar] e nem sempre a digitação era assertiva, logo nem sempre os jogos rodavam. Os poucos que rodavam, apesar da alegria de ver o jogo funcionando, não causavam muita emoção.

O que essa história tem a ver com este capítulo?

Vamos à associação. Apesar do jogo (que funcionava) rodar normalmente no terminal, caso eu desligasse o computador, o jogo sumia! Por quê? Ele estava na memória RAM [meio que é ativo enquanto há energia elétrica] e, ao desligar, todo código digitado era perdido. Basicamente foi o que fizemos até o capítulo anterior: todos os dados fornecidos pelo usuário eram perdidos assim que parasse de executar o programa.

Hoje, facilmente gravamos as informações dos arquivos em meios magnéticos (como hd, pendrive e afins...), mas, naquela época não tínhamos estes hardwares para facilitar o nosso lado; então, como gravávamos os códigos se o terminal não tinha HD?

Gravávamos em fitas cassetes, através de um aparelho que reproduzisse som e tivesse o recurso REC, ou seja, este era o meio utilizado para gravar e recuperar os arquivos em outro momento. Mesmo assim, tínhamos a insegurança de ocorrer falha na gravação sem prévio aviso e perdermos tudo.

Neste capítulo, veremos a facilidade com que o Python consegue gravar, manipular e ler arquivos nos meios magnéticos disponíveis.

1.3 Gravar ou não gravar? Eis a questão

Depois de aprendermos a manipular arquivos todos os nossos programas utilizarão este novo conceito? Não necessariamente! Gravar um arquivo não é uma necessidade, mas sim uma opção.

Por exemplo, em um hospital precisamos imprimir em uma folha a frase **“Triagem à direita →”**; por que gravar um arquivo com esta mensagem se a minha necessidade é a de apenas imprimir uma folha com estes dizeres para organizar as filas na recepção?

Manipulação de arquivos texto e JSON

Fazemos o mesmo questionamento quando vamos gerar códigos de programação. Se eu desejo criar um [módulo] programa que verifique se um CPF é válido ou não, para que gravar? Ou pensando em outra necessidade oposta: Ao gerar log de acesso dos usuários em um sistema, seria plausível efetuar a gravação? Sim, com isso teríamos um melhor controle dos acessos.

O bom cenário é o de analisarmos o problema e ver se ele pede a necessidade ou não de armazenamento.

2 MANIPULAÇÃO DE ARQUIVOS TEXTO

Antes de manipularmos arquivos no formato texto, devemos entender o seu conceito. Arquivos textos são aqueles capazes de reproduzir diretamente os caracteres gravados no arquivo na tela ou vice-versa.

Esta afirmação pode parecer óbvia, mas o mecanismo de gravação de **Editor de texto** é diferente do **Processador de texto**.

O Editor de Texto permite a gravação dos 256 caracteres da tabela ASCII (imprimíveis ou não) sem formatações ou inserção de elementos não texto (como figuras) no arquivo.

ASCII control characters			ASCII printable characters			
00	NULL	(Null character)	32	space	64	@
01	SOH	(Start of Header)	33	!	65	A
02	STX	(Start of Text)	34	"	66	B
03	ETX	(End of Text)	35	#	67	C
04	EOT	(End of Trans.)	36	\$	68	D
05	ENQ	(Enquiry)	37	%	69	E
06	ACK	(Acknowledgement)	38	&	70	F
07	BEL	(Bell)	39	'	71	G
08	BS	(Backspace)	40	(72	H
09	HT	(Horizontal Tab)	41)	73	I
10	LF	(Line feed)	42	*	74	J
11	VT	(Vertical Tab)	43	+	75	K
12	FF	(Form feed)	44	,	76	L
13	CR	(Carriage return)	45	-	77	M
14	SO	(Shift Out)	46	.	78	N
15	SI	(Shift In)	47	/	79	O
16	DLE	(Data link escape)	48	0	80	P
17	DC1	(Device control 1)	49	1	81	Q
18	DC2	(Device control 2)	50	2	82	R
19	DC3	(Device control 3)	51	3	83	S
20	DC4	(Device control 4)	52	4	84	T
21	NAK	(Negative acknowl.)	53	5	85	U
22	SYN	(Synchronous idle)	54	6	86	V
23	ETB	(End of trans. block)	55	7	87	W
24	CAN	(Cancel)	56	8	88	X
25	EM	(End of medium)	57	9	89	Y
26	SUB	(Substitute)	58	:	90	Z
27	ESC	(Escape)	59	;	91	[
28	FS	(File separator)	60	<	92	\
29	GS	(Group separator)	61	=	93]
30	RS	(Record separator)	62	>	94	^
31	US	(Unit separator)	63	?	95	_
127	DEL	(Delete)				

Figura 1 – Tabela ASCII
Fonte: Treina Web (2022)

Agora, o Processador de Texto consegue formatar um texto de diversas formas (negrito, itálico, fontes, cores...), inserindo, inclusive, elementos não texto entre outras formatações.

2.1 Formas de abertura de arquivos

A forma de abertura dos arquivos diz o que pode ser feito com ele. Dependendo da letra/caractere utilizado, o arquivo terá um modo específico de funcionamento.

Vejam os principais caracteres:

Caractere	Significado
'w'	Abertura em modo escrita. Ele reinicia o arquivo, caso exista, assim o conteúdo anterior é perdido.
'r'	Abertura em modo Leitura.
'x'	Abertura em modo escrita. Caso o arquivo, já exista ele retorna um erro.
'a'	Abertura em modo escrita. Caso o arquivo exista, ele insere os novos elementos no final.
'+'	Abertura para edição, seja em modo leitura ou escrita.

Tabela 1 – Caracteres de abertura de arquivo texto
Fonte: Docs.python.org (2022)

Quando vamos aprender alguma linguagem, é bom pegarmos o hábito de saber onde está e ler a documentação oficial da ferramenta, porque ela apresenta toda minuciosidade de cada conceito. A documentação oficial do Python fica em <<https://docs.python.org/3/library/functions.html#open>>.

2.1.1 Função open()

A função open, como diz o nome, “abre” um arquivo, ela será utilizada para a abertura de praticamente todos os arquivos em Python.

A sua sintaxe básica é (caso queira saber todos os parâmetros da função open(), consulte a documentação):

```
<objeto> = open("<nome do arquivo>", "<forma de abertura>")
```

Código-fonte 1 – Sintaxe da função open()

Fonte: Elaborado pelo autor (2023)

Sendo <objeto> o nome do objeto que vai representar o arquivo físico dentro do programa; "<nome do arquivo>" o nome físico do arquivo que será gravado ou recuperado e "<forma de abertura>" o parâmetro que diz a forma com que você deseja manipular este arquivo.

O complemento da função open() é a close(), ou seja, é de bom grado que tudo que for aberto em um programa em algum momento seja fechado.

Veja o exemplo:

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

Código-fonte 2 – Exemplo de abertura e fechamento de arquivo

Fonte: Elaborado pelo autor (2023)

No exemplo acima, o 'arq' representa o objeto que representa o arquivo aberto dentro do programa; 'arquivo.txt', o arquivo que será gravado no dispositivo magnético, 'w' diz que o arquivo será aberto em modo de escrita (write) e a função close() fecha o objeto que representa o arquivo.

2.1.2 Abertura modo escrita | write ('w')

A abertura do arquivo em modo escrita (w) pode confundir muitos por associação a outros aplicativos.

Por exemplo, quando abrimos um arquivo do word para "escrevermos", partimos do pressuposto que podemos editá-lo [e isso é verdade!] dentro de um conteúdo pré-existente, porém, isso não ocorre no modo de abertura 'w' da função open().

Manipulação de arquivos texto e JSON

Este modo de abertura ignora o arquivo existente. Como assim? Se o arquivo existir, o ele recria apagando todo o conteúdo existente, permitindo a nova escrita. Agora, se o arquivo não existir, ele o cria e permite a escrita;

Vejam um exemplo:

PYTHON:

```
# Abre o arquivo em modo Escrita
arq = open("arquivo.txt", "w")
# Grava a primeira linha
arq.write("Gravando a linha 1 no arquivo\n")
# Grava a segunda linha
arq.write("Gravando a linha 2 no arquivo\n")
# Grava a terceira linha
arq.write("Gravando a linha 3 no arquivo\n")
# fecha o arquivo
arq.close()
```

Código-fonte 3 – Abertura de arquivo em modo escrita
Fonte: Elaborado pelo autor (2023)

O nome do objeto 'arq' seguido do método 'write' permite a escrita de algo dentro do arquivo.

O exemplo acima criou magneticamente o "arquivo.txt" e nele gravou três linhas, fechando-o na conclusão.

No disco o "arquivo.txt" ficou:

PROMPT:

Gravando	a	linha	1	no	arquivo
Gravando	a	linha	2	no	arquivo
Gravando a linha 3 no arquivo					

Comando de prompt 1 – Conteúdo do arquivo.txt
Fonte: Elaborado pelo autor (2023)

2.1.3 Abertura modo leitura | read ('r')

A abertura do arquivo em modo escrita (r), como diz o nome, só pode abrir o arquivo para leitura, ou seja, não é possível efetuar qualquer tipo de edição.

Manipulação de arquivos texto e JSON

Para exemplificarmos este modo, vamos considerar que o tópico anterior foi executado, ou seja, há um arquivo gravado com três linhas chamado 'arquivo.txt' no disco.

Veja como fica simples:

PYTHON:

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

Código-fonte 4 – Abertura de arquivo em modo leitura
Fonte: Elaborado pelo autor (2023)

Será apresentado na tela:

PROMPT:

Gravando	a	linha	1	no	arquivo
Gravando	a	linha	2	no	arquivo
Gravando a linha 3 no arquivo					

Comando de prompt 2 – Exibição do conteúdo do arquivo.txt
Fonte: Elaborado pelo autor (2023)

O método `read()` apresenta na tela o conteúdo de todo arquivo. Veremos detalhes dos métodos de abertura mais a frente.

2.1.4 Abertura modo edição | `append` ('a')

A abertura do arquivo em modo `append` (a), permite que um arquivo existente seja editado sem perder o conteúdo original.

Veja a continuidade:

PYTHON:

Manipulação de arquivos texto e JSON

```
# Abre o arquivo em modo edição
arq = open("arquivo.txt", "a")
# Insere uma nova linha depois da última
arq.write("Inserindo uma nova linha")
# Fecha o arquivo
arq.close()
```

Código-fonte 5 – Abertura de arquivo em modo de edição (append)
Fonte: Elaborado pelo autor (2023)

Considerando o arquivo criado anteriormente, ele adiciona um novo conteúdo.

Digitando o código abaixo, veremos o novo conteúdo inserido no final do arquivo existente:

PYTHON:

```
# Abre o arquivo em modo edição
arq = open("arquivo.txt", "r")
# Exibe o conteúdo do arquivo
print(arq.read())
# Fecha o arquivo
arq.close()
```

Código-fonte 6 – Conteúdo do arquivo.txt depois do append
Fonte: Elaborado pelo autor (2023)

Apresentando na tela o conteúdo:

PROMPT:

```
Gravando a linha 1 no arquivo
Gravando a linha 2 no arquivo
Gravando a linha 3 no arquivo
Inserindo uma nova linha
```

Comando de prompt 3 – Exibição do conteúdo do arquivo.txt editado
Fonte: Elaborado pelo autor (2023)

2.1.5 Abertura modo escrita | write ('x')

A abertura do arquivo no modo 'x' é similar a abertura no modo 'w'.

Manipulação de arquivos texto e JSON

A diferença é que no modo 'w' o arquivo é sobreposto, caso ele exista, ou será criado, caso não exista; já no modo 'x', caso o arquivo exista o interpretador Python retorna um erro; considere que o arquivo "arquivo.txt" já exista no disco, caso o modo de abertura 'x' seja aplicado, veja o que ocorrerá:

PYTHON:

```
arq = open("arquivo.txt", "x")
arq.write("teste")
arq.close()
```

Código-fonte 7 – Utilizando o modo de abertura x (exclusivo)
Fonte: Elaborado pelo autor (2023)

Retornou o erro dizendo que o arquivo já existe:

PROMPT:

```
Traceback (most recent call last):
  "File fase6.py", line 1, in <module>
    arq = open("arquivo.txt", "x")
FileExistsError: [Errno 17] File exists: 'arquivo.txt'
```

Comando de prompt 4 – Exibição da falha na abertura em modo 'x'
Fonte: Elaborado pelo autor (2023)

Caso o arquivo não exista, a escrita funciona normalmente, como no modo 'w', criando o arquivo.

O modo 'x' é útil nas situações em que há diferença no processo, caso o arquivo exista ou não previamente.

2.1.6 Abertura modo escrita + leitura | write + read ('+')

Há situações em que, por algum motivo, precisamos gravar ou recuperar informações de um arquivo. Se nos limitássemos a usar o 'w' ou o 'r' exclusivamente, teríamos que fechar o arquivo e abrir em outro modo para fazer este processo.

Manipulação de arquivos texto e JSON

Para otimizar este problema, temos o modo de abertura incluindo o '+', seja no write ou read ('w+' ou 'r+'). Utilizando este recurso, podemos fazer ambas as operações sem precisar fechar e reabrir o arquivo em outro modo.

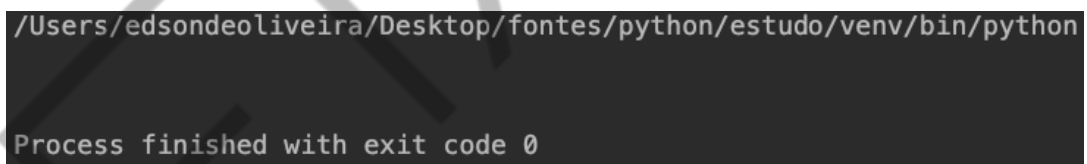
No código abaixo, abrimos o arquivo com o '+' (não importa se com 'w' ou 'r', tem o mesmo efeito), gravamos duas linhas e apresentamos o conteúdo do arquivo na tela:

PYTHON:

```
# Abre o arquivo em modo Escrita + Leitura
arq = open("arquivo.txt", "w+")
# Gravando duas linhas
arq.write("Gravando no arquivo utilizando o +\n")
arq.write("Desta forma, podemos tanto escrever quanto ler o\n")
arq.write("arquivo\n")
# Lendo o arquivo
print(arq.read())
# Fechando o arquivo
arq.close()
```

Código-fonte 8 – Utilizando o modo de abertura + (leitura e escrita)
Fonte: Elaborado pelo autor (2023)

Resultando na tela:



```
/Users/edsondeoliveira/Desktop/fontes/python/estudo/venv/bin/python
Process finished with exit code 0
```

Figura 2 – Tabela ASCII
Fonte: Elaborado pelo autor (2023)

Opa, tem algo estranho! Por que o conteúdo do arquivo não aparece na tela? Responderei no próximo tópico.

2.1.7 Utilizando o método seek()

Quando gravamos/lemos algo em um arquivo, a operação é feita a partir de onde está o cursor. Vamos narrar o que ocorreu no código anterior:

PYTHON:

Manipulação de arquivos texto e JSON

```
# Abre o arquivo em modo Escrita + Leitura  
arq = open("arquivo.txt", "w+")
```

Código-fonte 9 – Abertura para escrita e leitura
Fonte: Elaborado pelo autor (2023)

As linhas acima recriaram o arquivo.txt, logo, o cursor ficou na posição 0 (zero) do arquivo:

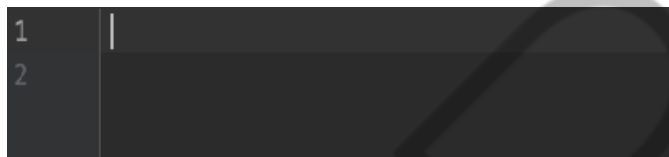


Figura 3 – Conteúdo do arquivo.txt vazio com o cursor no início
Fonte: Elaborado pelo autor (2023)

Digitando as próximas linhas:

PYTHON:

```
# Gravando duas linhas  
arq.write("Gravando no arquivo utilizando o +\n")  
arq.write("Desta forma, podemos tanto escrever quanto ler o  
arquivo\n")
```

Código-fonte 10 – Gravando duas linhas
Fonte: Elaborado pelo autor (2023)

Serão gravadas no arquivo as frases:

Figura 4 – Conteúdo do arquivo.txt com texto gravado e cursor no final do arquivo
Fonte: Elaborado pelo autor (2023)

Repare que o cursor ficou no final do arquivo, depois das duas frases.

Desta forma, ao digitarmos o comando abaixo, apesar do arquivo estar gravado com as frases, não será apresentado nada na tela, porque o método read exibe o conteúdo a partir do cursor:

PYTHON:

```
# Lendo o arquivo
print(arq.read())
# Fechando o arquivo
arq.close()
```

Código-fonte 11 – Exibindo e fechando o arquivo
Fonte: Elaborado pelo autor (2023)

Não há nada depois do cursor e não foi exibido nada.

Para resolvermos este problema, utilizamos o método seek.

A sua principal função é a de posicionar o cursor em algum ponto do arquivo. A posição 0 indica a primeira posição do arquivo (há outro parâmetro 'from_what', que determina o ponto de referência - início, atual ou fim do arquivo -, consulte a documentação para maiores detalhes).

Então, acrescentamos a linha de código com o seek no programa e ele funcionará como esperado, vejam:

PYTHON:

```
# Abre o arquivo em modo Escrita + Leitura
arq = open("arquivo.txt", "w+")
# Gravando duas linhas
arq.write("Gravando no arquivo utilizando o +\n")
arq.write("Desta forma, podemos tanto escrever quanto ler o\narquivo\n")
# Posicionando o cursor na posição zero do arquivo
arq.seek(0)
# Lendo o arquivo
print(arq.read())
# Fechando o arquivo
arq.close()
```

Código-fonte 12 – Aplicando o seek() no programa
Fonte: Elaborado pelo autor (2023)

O resultado será:

```
Gravando no arquivo utilizando o +  
Desta forma, podemos tanto escrever quanto ler o arquivo
```

Figura 5 – Exibição do conteúdo do arquivo.txt
Fonte: Elaborado pelo autor (2023)

Depois de gravar e antes de exibir o arquivo na íntegra, ele posicionou o cursor no início do arquivo (antes da letra 'G').

Para entendermos melhor o método seek(), vamos pegar o mesmo arquivo gravado e modificar parte do código:

PYTHON:

```
# Abre o arquivo em modo Escrita + Leitura  
arq = open("arquivo.txt", "r+")  
# Posicionando o cursor na posição zero do arquivo  
arq.seek(9)  
# Lendo o arquivo  
print(arq.read())  
# Fechando o arquivo  
arq.close()
```

Código-fonte 13 – Aplicando o seek(9) no programa
Fonte: Elaborado pelo autor (2023)

O conteúdo do arquivo.txt é:

PROMPT:

```
Gravando no arquivo utilizando o +  
Desta forma, podemos tanto escrever quanto ler o arquivo
```

Comando de prompt 5 – Exibição do conteúdo do arquivo.txt na tela
Fonte: Elaborado pelo autor (2023)

Quando colocamos seek(0), o cursor ficou antes do 'G' de Gravando. Sugerimos no código modificado o seek(9), então, o cursor será posicionado nove posições depois do início do arquivo, ficando antes de 'n' de 'no' e a exibição do arquivo na tela será a partir deste ponto.

Apesar do conteúdo do arquivo não ter sido modificado, será apresentado na tela:

```
no arquivo utilizando o +  
Desta forma, podemos tanto escrever quanto ler o arquivo  
|
```

Figura 6 – Exibição do conteúdo do arquivo.txt com o cursor a partir da posição 9
Fonte: Elaborado pelo autor (2023)

Reparem que a palavra “Gravando” foi ignorada.

2.2 Outras formas de leitura e escrita de arquivos

Até o momento, o único método que vimos para leitura de um arquivo foi o método `read()`. Ele possibilita a leitura do arquivo na íntegra.

Caso desejemos ler parte do arquivo, utilizaremos os métodos `readline()` e `readlines()`.

2.2.1 Método de leitura `readline()`

O método `readline()` captura apenas uma linha do arquivo. Esta será a linha onde o cursor está posicionado, independentemente de ser no início, meio ou fim dela.

Veja o código:

PYTHON:

Manipulação de arquivos texto e JSON

```
# Abre o arquivo em modo Escrita + Leitura
arq = open("arquivo.txt", "w+")

# Grava 3 linhas
arq.write("Linha 1 do arquivo\n")
arq.write("Linha 2 do arquivo\n")
arq.write("Linha 3 do arquivo\n")

# Posicionando o cursor na posição zero do arquivo
arq.seek(0)

# o método readline() captura uma linha do arquivo
print(arq.readline())

# Fechando o arquivo
arq.close()
```

Código-fonte 14 – Aplicando o método readline()
Fonte: Elaborado pelo autor (2023)

Este programa gravou três linhas no arquivo.txt, posicionou o cursor no início da primeira linha e utilizou o método readline() para exibir a primeira linha, a apresentação na tela será:

PROMPT:

```
Linha 1 do arquivo
```

Comando de prompt 6 – Execução do programa com readline()
Fonte: Elaborado pelo autor (2023)

Para imprimir as próximas linhas basta repetir os códigos com readline():

PYTHON:

Manipulação de arquivos texto e JSON

```
# Abre o arquivo em modo Escrita + Leitura
arq = open("arquivo.txt", "w+")

arq.write("Linha 1 do arquivo\n")
arq.write("Linha 2 do arquivo\n")
arq.write("Linha 3 do arquivo\n")

# Posicionando o cursor na posição zero do arquivo
arq.seek(0)

# o método readline() captura uma linha do arquivo
print(arq.readline())
print(arq.readline())
print(arq.readline())

# Fechando o arquivo
arq.close()
```

Código-fonte 15 – Aplicando o método readline() para exibir três linhas
Fonte: Elaborado pelo autor (2023)

Fazendo dessa forma, a apresentação ficará:

PROMPT:

```
Linha 1 do arquivo

Linha 2 do arquivo

Linha 3 do arquivo
```

Comando de prompt 7 – Execução do programa com espaçamento duplo entre linhas
Fonte: Elaborado pelo autor (2023)

Por que este espaço adicional entre as linhas na exibição?

Repare que nas linhas de código:

PYTHON:

Manipulação de arquivos texto e JSON

```
...
arq.write("Linha 1 do arquivo\n")
arq.write("Linha 2 do arquivo\n")
arq.write("Linha 3 do arquivo\n")
...
```

Código-fonte 16 – Gravações das linhas com quebras
Fonte: Elaborado pelo autor (2023)

As frases foram gravadas com ‘\n’ no final e este caractere de função faz o cursor mudar de linha (quebra de linha). Se ele não fosse colocado, o texto das três linhas ficaria em uma linha só, porque o write() grava a partir de onde está o cursor. Por isso esta quebra de linha. Até aí tudo bem, mas porque na exibição da tela houve um espaço adicional entre as linhas?

Repare no Código:

PYTHON:

```
...
print(arq.readline())
print(arq.readline())
print(arq.readline())
...
```

Código-fonte 17 – Gravações das linhas sem quebras
Fonte: Elaborado pelo autor (2023)

O método readline() recupera uma linha do arquivo (lembre-se que cada linha for gravada com um ‘\n’ [quebra de linha] no final); o **print**, por natureza, já quebra a linha depois de exibir a mensagem, assim, o ‘\n’ e a quebra de linha do print são exibidos na sequência e, consequentemente, é dado um espaço a mais entre as linhas.

Há alguma solução? Sim! Para tudo há.

Vamos modificar o código acrescentando o end="":

PYTHON:

```
print(arq.readline(), end=' ')
print(arq.readline(), end=' ')
print(arq.readline(), end=' ')
```

Manipulação de arquivos texto e JSON

Código-fonte 18 – Leitura das linhas sem quebras
Fonte: Elaborado pelo autor (2023)

No final de cada linha acrescentamos `end=' '` e isso significa que a quebra de linha do `print` será substituída no final da linha pelo caractere vazio, ou seja, o caractere 13 (que é o ENTER na tabela ASCII), é substituído pelo caractere " " que é o vazio.

Também podemos utilizar o `.strip()` no lugar do `end=' '` para subtrair a quebra de linha. Ficando:

```
print(arq.readline().strip())
```

Segue agora o código completo:

PYTHON:

```
# Abre o arquivo em modo Escrita + Leitura
arq = open("arquivo.txt", "w+")

arq.write("Linha 1 do arquivo\n")
arq.write("Linha 2 do arquivo\n")
arq.write("Linha 3 do arquivo\n")

# Posicionando o cursor na posição zero do arquivo
arq.seek(0)

# o método readline() captura uma linha do arquivo
print(arq.readline(), end=' ')
print(arq.readline(), end=' ')
print(arq.readline(), end=' ')

# Fechando o arquivo
arq.close()
```

Código-fonte 19 – Código completo aplicando `readline()`
Fonte: Elaborado pelo autor (2023)

E a apresentação:

PROMPT:


```
Linha 1 do arquivo  
Linha 2 do arquivo  
Linha 3 do arquivo
```

Comando de prompt 8 – Execução do programa sem espaçamento duplo entre linhas
Fonte: Elaborado pelo autor (2023)

2.2.2 Método de leitura `readlines()`

O método `readline()` captura a linha onde está o cursor. Partindo deste conceito, seria difícil posicionar o cursor em uma de diversas linhas para capturá-la. Por exemplo, por algum motivo eu preciso exibir somente a linha 5 (ou a partir da linha 5 de um arquivo com diversas linhas), como fazer?

O método `readlines()` é a solução! Ele pega cada linha e coloca em um item de uma lista e, com a lista, podemos manipular os itens como desejarmos.

Segue o código:

PYTHON:

```
# Abre o arquivo em modo Escrita + Leitura  
arq = open("arquivo.txt", "w+")  
  
# Gravando 5 linhas no arquivo  
arq.write("Linha 1\n")  
arq.write("Linha 2\n")  
arq.write("Linha 3\n")  
arq.write("Linha 4\n")  
arq.write("Linha 5\n")  
  
# Posicionando o cursor na posição zero do arquivo  
arq.seek(0)  
  
# Capturando todas as linhas do arquivo e jogando em uma  
# lista  
lista_linhas_arquivo = arq.readlines()  
  
# Exibindo o conteúdo da lista  
print(lista_linhas_arquivo)  
  
# Fechando o arquivo  
arq.close()
```

Manipulação de arquivos texto e JSON

Código-fonte 20 – Código completo aplicando readlines()
Fonte: Elaborado pelo autor (2023)

Neste programa gravamos 5 linhas no arquivo.txt. Depois de posicionar o cursor no início do arquivo, todas as linhas são capturadas e armazenadas em uma lista.

A linha de código que faz isso é:

PYTHON:

```
# Capturando todas as linhas do arquivo e jogando em uma lista
lista_linhas_arquivo = arq.readlines()
```

Código-fonte 21 – Parte do código aplicando readlines()
Fonte: Elaborado pelo autor (2023)

Já que as linhas estão gravadas na lista lista_linhas_arquivo, se dermos um print nesta lista:

PYTHON:

```
# Exibindo o conteúdo da lista
print(lista_linhas_arquivo)
```

Código-fonte 22 – Exibindo o conteúdo a lista gerada pelo readlines()
Fonte: Elaborado pelo autor (2023)

Será exibido na tela:

PROMPT:

```
['Linha 1\n', 'Linha 2\n', 'Linha 3\n', 'Linha 4\n', 'Linha 5\n']
```

Comando de prompt 9 – Exibição da Lista lista_linhas_arquivo
Fonte: Elaborado pelo autor (2023)

Esta exibição é a bruta da lista, e, como vimos em listas, podemos trabalhar com cada elemento separadamente.

Veja o código para exibir a terceira linha:

PYTHON:

Manipulação de arquivos texto e JSON

```
# Exibindo a linha 3
print("Exibindo a linha 3.....: ", end='')
print(lista_linhas_arquivo[2])
```

Código-fonte 23 – Exibindo o conteúdo a lista gerada pelo readlines()
Fonte: Elaborado pelo autor (2023)

O resultado será:

PROMPT:

```
Exibindo a linha 3.....: Linha 3
```

Comando de prompt 10 – Exibição da terceira linha da Lista lista_linhas_arquivo
Fonte: Elaborado pelo autor (2023)

Vamos pensar numa situação em que precisamos exibir as linhas ímpares do arquivo. Para facilitar, vamos criar um procedimento:

PYTHON:

```
... linhas anteriores
# Procedimento que exibe as linhas ímpares
def exibe_linhas_impares(l):
    for i in range(0, len(l), 2):
        print(f"Exibindo a linha {i + 1}.....: {l[i]}",
              end='')

# Chamada do procedimento
exibe_linhas_impares(lista_linhas_arquivo)

# Fechando o arquivo
arq.close()
```

Código-fonte 24 – Criação do procedimento exibe_linhas_impares()
Fonte: Elaborado pelo autor (2023)

A exibição será:

PROMPT:

```
Exibindo a linha 1.....: Linha 1
Exibindo a linha 3.....: Linha 3
Exibindo a linha 5.....: Linha 5
```

2.2.3 Método de escrita writelines()

Da mesma forma que o modo de abertura de leitura tem no método `read()` e os seus derivados [`readline()` e `readlines()`], o modo de abertura escrita tem como derivado o método `writelines()`.

O método `write()` grava linha por linha no arquivo. O seu derivado – o `writelines()` – permite gravar múltiplas linhas de uma vez.

Estas linhas devem estar em uma Tupla ou em uma Lista.

Veja o exemplo da aplicação do `writelines()` utilizando tuplas:

PYTHON:

```
# Abre o arquivo em modo Edição + Leitura
arq = open("arquivo.txt", "w+")

# Gravando 5 linhas individualmente
arq.write("Linha 1\n")
arq.write("Linha 2\n")
arq.write("Linha 3\n")
arq.write("Linha 4\n")
arq.write("Linha 5\n")

# Gravando 3 linhas de uma vez
arq.writelines(
    ("Nova linha 6\n",
     "Nova linha 7\n",
     "Nova linha 8\n") # utilizando tupla
)

# Exibindo o conteúdo do arquivo
arq.seek(0)
print(arq.read())

# Fecha o arquivo
arq.close()
```

Código-fonte 25 – Código utilizando writelines()
Fonte: Elaborado pelo autor (2023)

O arquivo é aberto em modo escrita (+ leitura), são gravadas 5 linhas independentes com o método write(), depois são gravadas três linhas – que estão dentro de uma Tupla – de uma vez, o arquivo é exibido na íntegra e, por fim, ele é fechado.

2.3 Gerenciador de contexto with

Em programação, todos os comandos estruturados têm início, meio e fim. Estes comandos são montados pelos programadores de acordo com as necessidades da resolução do problema.

O gerenciador de contexto tem um objetivo parecido, mas utilizando objetos e arquivos externos ao código fonte em si (diferentemente dos comandos estruturados).

Quando usamos arquivos, basicamente seguimos o fluxo:

- Abrimos o arquivo em algum modo.
- Manipulamos (escrevemos ou lemos) o arquivo.
- Fechamos o arquivo.

Com os comandos estruturados, o programador deve se precaver sobre quais utilizar para montar o seu raciocínio, ou seja, ele trata as situações (erros) lógicas nestas estruturas. Agora, quando usamos arquivos (objetos externos), não basta a lógica do algoritmo estar correta, muitos erros de **operação** podem ocorrer.

Erro de compilação: Escrita errada do comando.

Erro de lógica: Quando o programador não prevê uma situação e resulta algo falacioso.

Erro de operação: Algo alheio a aplicação ocorre como falta de energia, internet ou inexistência de um arquivo.

Considere que um arquivo deve ser aberto em modo append e o arquivo original (com dados) não está no disco; ou um arquivo é aberto e no meio da sua manipulação

Manipulação de arquivos texto e JSON

ocorre algum tipo de erro; o fluxo é terminado sem que tenha sido efetuada a sua última parte: fechamento do arquivo.

Caso isso ocorra, o que pode acontecer com o arquivo? Depende do SO, mas, de qualquer forma, o arquivo pode ser corrompido.

O gerenciador de contexto **with** (em Python) resolve este problema porque, independentemente da parte do fluxo de operação, ele fecha o arquivo seja em meio a uma falha ou não.

Sintaxe:

```
with open("<arquivo>", "modo") as <alias>:  
    <manipulação do arquivo>
```

Código-fonte 26 – Código utilizando writelines()
Fonte: Elaborado pelo autor (2023)

O gerenciador de contexto é iniciado com a palavra **with** seguido da função **open()** com o modo de abertura de arquivo. No final da linha, a palavra **as** permite colocar um **alias** (apelido) na representação do arquivo texto dentro do código fonte. Todos os próximos comandos relacionados a este contexto devem ser escritos endentados tomando como base o **with**.

Exemplo parcial:

PYTHON:

```
with open("arquivo.txt", "w") as arq:  
    arq.write("Teste")  
    ...
```

Código-fonte 27 – Aplicando o gerenciador de contexto with
Fonte: Elaborado pelo autor (2023)

Vamos aplicar o **with** em um exemplo no qual gravamos o conteúdo de uma lista em um arquivo:

PYTHON:

Manipulação de arquivos texto e JSON

```
# aplicando o gerenciador de contexto 'with'
with open("arquivo.txt","w+") as arq:
    # Criando uma lista cujos itens serão linhas
    lista = ["Gravando\n", "varias\n", "linhas\n", "através de\n", "lista\n"]

    # Grava a lista no arquivo
    arq.writelines(lista)

    # Posicionando o cursor na posição zero do arquivo
    arq.seek(0)

    # Exibindo o arquivo
    print(arq.read())
```

Código-fonte 28 – Aplicando o gerenciador de contexto with gravando com Lista
Fonte: Elaborado pelo autor (2023)

No exemplo acima, não precisamos mais de utilizar o método close() para fechar o arquivo, a própria finalização do with se encarrega de fechar o arquivo.

3 UTILIZAÇÃO DE ARQUIVOS .JSON

Os arquivos no formato texto tem as suas limitações, principalmente quando desejamos tratar da **organização** dos dados dentro de um arquivo. Para tal, usamos os arquivos .JSON.

3.1 Antes um pouco de XML

Há um formato de arquivo chamado .XML (Extensible Markup Language), que surgiu na década de 90, cujo benefício é o de efetuar uma melhor comunicação entre sistemas que precisam trocar dados em processos repetidos. Por exemplo, todos os dados das notas fiscais de um estabelecimento.

Basicamente, ele funciona de forma parecida com o HTML (Hyper Text Markup Language) e a sua estruturação é baseada em aberturas e fechamentos de tag's.

XML:

```
<?xml version="2.0">
<frutas>
  <fruta id="1">
    <nome>Laranja</nome>
    <tipo>Cítrica</resumo>
    <colheita>01/01/2023</colheita>
    <validade>01/04/2023</validade>
    <cidade>Limeira</elenco>
  </fruta>
  <fruta id="2">
    <nome>Limão</nome>
    <tipo>Cítrica</resumo>
    <colheita>05/01/2023</colheita>
    <validade>05/05/2023</validade>
    <cidade>São Paulo</elenco>
  </fruta>
</frutas>
```

Código-fonte 29 – Arquivo XML
Fonte: Elaborado pelo autor (2023)

3.2 Definição de arquivo JSON

O arquivo .JSON () é o “irmão mais novo” do XML. Este tipo de arquivo também efetua a troca de dados entre sistemas, com a vantagem de ter o código mais limpo e ser mais rápido que o XML.

Segue um exemplo similar ao que usamos no exemplo XML:

JSON:

```
{
  "id": "1",
  "nome": "Laranja",
  "tipo": "Cítrica",
  "colheita": "01/01/2023",
  "validade": "01/04/2023",
  "cidade": "Limeira"
},
{
  "id": "2",
  "nome": "Limão",
  "tipo": "Cítrica",
  "colheita": "05/01/2023",
  "validade": "05/05/2023",
  "cidade": "São Paulo"
}
```

Código-fonte 30 – Arquivo JSON
Fonte: Elaborado pelo autor (2023)

Comparando os dois formatos, é perceptível que o JSON é um código mais limpo. Entre outras vantagens, como desempenho e os comentários dentro do código.

3.3 Criando uma aplicação que gera arquivo JSON

Veremos agora como geramos arquivos JSON na prática.

Vamos construir uma aplicação explicando o **passo a passo** (fragmentando o código) e, no final do tópico, o arquivo fonte será disponibilizado integralmente.

3.3.1 Definindo um dicionário

Lembram dos dicionários? A estrutura JSON é praticamente igual! Faremos o comparativo no desenvolvimento do projeto.

Primeira coisa, vamos construir um dicionário com valores constantes chamados 'pessoas':

PYTHON:

```
# Criando o dicionário de nome 'pessoas'
pessoas = {
    'pessoa1': {
        'nome': 'Edson',
        'idade': 48,
        'email': 'eds@fiap.com.br'
    },
    'pessoa2': {
        'nome': 'Jose',
        'idade': 23,
        'email': 'jose@fiap.com.br'
    },
    'pessoa3': {
        'nome': 'Maria',
        'idade': 29,
        'email': 'maria@fiap.com.br'
    },
}
```

Código-fonte 31 – Código para criar o dicionário pessoas
Fonte: Elaborado pelo autor (2023)

Dentro deste dicionário fixamos os Nomes, Idades e e-mails de três pessoas.

Se simplesmente dermos um print neste dicionário:

PYTHON:

```
print(pessoas)
```

Código-fonte 32 – Exibição do dicionário pessoas
Fonte: Elaborado pelo autor (2023)

Veremos a seguinte exibição bruta:

PROMPT:

```
{'pessoa1': {'nome': 'Edson', 'idade': 48, 'email': 'eds@fiap.com.br'}, 'pessoa2': {'nome': 'Jose', 'idade': 23, 'email': 'jose@fiap.com.br'}, 'pessoa3': {'nome': 'Maria', 'idade': 29, 'email': 'maria@fiap.com.br'}}
```

Comando de prompt 12 – Exibição do dicionário pessoas
Fonte: Elaborado pelo autor (2023)

Até o momento não há nada de JSON.

3.3.2 Importando a biblioteca JSON

Para utilizarmos os recursos de JSON dentro do Python, devemos adicionar a biblioteca correspondente. Fazemos isso com a instrução abaixo. Este import deve ser colocado antes de começarmos a utilizar os métodos relacionados:

PYTHON:

```
import json
```

Código-fonte 33 – Importando a biblioteca JSON
Fonte: Elaborado pelo autor (2023)

Digitando esta instrução, nada de diferente ocorre na exibição da aplicação.

3.3.3 JSON: Método dumps()

O método dumps() permite codificar um objeto Python formatado jogando-o em uma string. O dump() – no singular – faz a mesma coisa, mas jogando diretamente no arquivo (veja a documentação do Python para mais detalhes).

Veja a sequência do código:

PYTHON:

Manipulação de arquivos texto e JSON

```
peessoas_json = json.dumps(pessoas)
# Exibindo as estruturas
print("Exibição do dicionário: ")
print(pessoas)
print("\nExibição do objeto JSON: ")
print(peessoas_json)
```

Código-fonte 34 – Aplicando o método dumps()
Fonte: Elaborado pelo autor (2023)

A exibição será:

PROMPT:

```
Exibição do dicionário:
{'pessoa1': {'nome': 'Edson', 'idade': 48, 'email': 'eds@fiap.com.br'}, 'pessoa2': {'nome': 'Jose', 'idade': 23, 'email': 'jose@fiap.com.br'}, 'pessoa3': {'nome': 'Maria', 'idade': 29, 'email': 'maria@fiap.com.br'}}

Exibição do objeto JSON:
{"pessoa1": {"nome": "Edson", "idade": 48, "email": "eds@fiap.com.br"}, "pessoa2": {"nome": "Jose", "idade": 23, "email": "jose@fiap.com.br"}, "pessoa3": {"nome": "Maria", "idade": 29, "email": "maria@fiap.com.br"}}
```

Comando de prompt 13 – Exibição do dicionário o objeto JSON
Fonte: Elaborado pelo autor (2023)

A única diferença é que para delimitar a Key ou o conteúdo em dicionários podemos usar as aspas simples ou duplas, enquanto o objeto JSON obrigatoriamente usa aspas duplas.

3.3.4 JSON: Método dumps() com o parâmetro indent

O “triste” é que a exibição do objeto JSON é bruta, assim como no dicionário. Para melhorar esta visualização, podemos acrescentar o parâmetro **indent** para organizar a visualização dos dados:

PYTHON:

Manipulação de arquivos texto e JSON

```
peessoas_json = json.dumps(peessoas, indent=4)
```

Código-fonte 35 – linha de aplicação do método dumps()
Fonte: Elaborado pelo autor (2023)

Este acréscimo – indent=4 – formata o conteúdo do objeto com 4 espaços de indentação, veja:

PYTHON:

Exibição do objeto JSON:

```
{
  "pessoa1": {
    "nome": "Edson",
    "idade": 48,
    "email": "eds@fiap.com.br"
  },
  "pessoa2": {
    "nome": "Jose",
    "idade": 23,
    "email": "jose@fiap.com.br"
  },
  "pessoa3": {
    "nome": "Maria",
    "idade": 29,
    "email": "maria@fiap.com.br"
  }
}
```

Comando de prompt 14 – Exibição objeto JSON
Fonte: Elaborado pelo autor (2023)

Desta forma, fica mais fácil visualizar e interpretar os dados.

3.3.5 JSON: Gravando no arquivo

Agora vamos para a código que grava o objeto JSON no arquivo .json:

PYTHON:

```
with open("arquivo.json", "w+") as file:
    # Gravando o objeto no arquivo .json
    file.write(pessoas_json)
```

Código-fonte 36 – Gravação do arquivo JSON
Fonte: Elaborado pelo autor (2023)

Não muito diferente do que já vimos na gravação de arquivos texto.

Por padrão, agora inserimos a extensão .json para diferenciar o arquivo JSON dos demais.

3.3.6 JSON: Carregando dados do arquivo

Agora vamos carregar o conteúdo do arquivo.json em um objeto (ou um dicionário) dentro do Python:

PYTHON:

```
with open("arquivo.json", "r") as file:
    # Lendo o arquivo .json
    pessoas_json = file.read()
    pessoas = json.loads(pessoas_json)
```

Código-fonte 37 – Carregando dados do arquivo JSON
Fonte: Elaborado pelo autor (2023)

Neste caso, colocamos o conteúdo do arquivo dentro de um dicionário de nome 'pessoas' através do método '**loads**' (linha de comando: `pessoas = json.loads(pessoas_json)`); o método loads é o complemento do dumps, ele carrega os dados gravados no arquivo e os transporta para um objeto dentro do código Python.

3.3.7 JSON: Exibindo os dados do arquivo na tela

Com o objeto (pessoas_json) e dicionário (pessoas) carregados, fica fácil exibirmos o conteúdo na tela. O comando **print(pessoas)** simplesmente exibe de forma bruta o conteúdo do dicionário:

PROMPT:

```
{'pessoa1': {'nome': 'Edson', 'idade': 48, 'email': 'eds@fiap.com.br'}, 'pessoa2': {'nome': 'Jose', 'idade': 23, 'email': 'jose@fiap.com.br'}, 'pessoa3': {'nome': 'Maria', 'idade': 29, 'email': 'maria@fiap.com.br'}}
```

Comando de prompt 15 – Exibição dos dados do dicionário na tela
Fonte: Elaborado pelo autor (2023)

Exibir na tela com a indentação, como no caso da gravação, é indiferente, porque o usuário da aplicação não tem autonomia para compreender o conteúdo do arquivo JSON, ainda que esteja indentado.

Contudo, se quisermos exibir na forma indentada, usamos o objeto `peessoas_json` que foi carregado pelo `file.read()` na linha de comando **`peessoas_json = file.read()`**, veja como ficará a apresentação:

PYTHON:

```
{
    "pessoa1": {
        "nome": "Edson",
        "idade": 48,
        "email": "eds@fiap.com.br"
    },
    "pessoa2": {
        "nome": "Jose",
        "idade": 23,
        "email": "jose@fiap.com.br"
    },
    "pessoa3": {
        "nome": "Maria",
        "idade": 29,
        "email": "maria@fiap.com.br"
    }
}
```

Comando de prompt 16 – Exibição dos dados do objeto JSON
Fonte: Elaborado pelo autor (2023)

3.3.8 JSON: Exibindo os dados formatados na tela

O programador é quem deve se preocupar com a exibição dos dados na tela. A partir de agora, usaremos todos os recursos aprendidos na disciplina para dar uma melhor visualização nos dados carregados do arquivo para o console.

Lembrando que neste caso estamos utilizando o dicionário 'pessoas' que foi carregada do arquivo:

PYTHON:

```
# exibição apresentável para o usuário
for k, v in pessoas.items():
    print(f"Registro.....: {k}")
    for k1, v1 in v.items():
        print("\t" + k1 + ":" + str(v1))
```

Código-fonte 38 – Código de exibição formatada do Dicionário
Fonte: Elaborado pelo autor (2023)

No primeiro for o **k** representa pessoa1, pessoa2, pessoa3... e **v** o dicionário aninhado destas pessoas. No segundo for o **v** (que é um dicionário) é colocado em evidência, sendo o **k1** representando as chaves (keys) nome, idade e e-mail e **v1** os conteúdos das chaves anteriores.

Com o acréscimo deste código, a exibição será:

PROMPT

```
Registro.....: pessoa1
    nome:Edson
    idade:48
    email:eds@fiap.com.br
Registro.....: pessoa2
    nome:Jose
    idade:23
    email:jose@fiap.com.br
Registro.....: pessoa3
    nome:Maria
    idade:29
    email:maria@fiap.com.br
```


Comando de prompt 17 – Exibição dos dados do objeto JSON formatado
Fonte: Elaborado pelo autor (2023)

3.3.9 Código JSON na íntegra

Para explicar o funcionamento do conteúdo JSON no Python, fragmentamos o código para um melhor entendimento.

Para contextualizar o aprendizado, segue o código na íntegra:

PYTHON:

```
# MANIPULACAO DE ARQUIVOS JSON
# Criando o dicionário 'pessoas'
pessoas = {
    'pessoal':{
        'nome': 'Edson',
        'idade': 48,
        'email': 'eds@fiap.com.br'
    },
    'pessoa2':{
        'nome': 'Jose',
        'idade': 23,
        'email': 'jose@fiap.com.br'
    },
    'pessoa3': {
        'nome': 'Maria',
        'idade': 29,
        'email': 'maria@fiap.com.br'
    },
}

# Importando a biblioteca JSON
import json

# Criando o objeto pessoas_json
```

Manipulação de arquivos texto e JSON

```

pessoas_json = json.dumps(pessoas, indent=4)

# Exibindo o dicionário pessoas
print("Exibição do dicionário: ")
print(pessoas)

# Exibindo o objeto JSON
print("\nExibição do objeto JSON: ")
print(pessoas_json)

# Abrindo o arquivo JSON para gravação
with open("arquivo.json", "w+") as file:
    # Gravando o objeto no arquivo .json
    file.write(pessoas_json)

with open("arquivo.json", "r") as file:
    # Lendo o arquivo .json
    file.seek(0)
    pessoas_json = file.read()
    pessoas = json.loads(pessoas_json)

# exibição 'rústica' do dicionário pessoas
print("\nExibição bruta do dicionário: ")
print(pessoas)

# Exibição do objeto com os dados edentados
print("\nExibição do objeto:")
print(pessoas_json)

# exibição apresentável para o usuário
print("Exibição formatada dos dados:")
for k, v in pessoas.items():
    print(f"Registro....: {k}")
    for k1, v1 in v.items():
        print("\t" + k1 + ":" + str(v1))
```

Código-fonte 39 – Código fonte JSON na íntegra
Fonte: Elaborado pelo autor (2023)

Manipulação de arquivos texto e JSON

Agora a execução deste código:

Exibição do dicionário:

PROMPT:

```
{'pessoa1': {'nome': 'Edson', 'idade': 48, 'email': 'eds@fiap.com.br'}, 'pessoa2': {'nome': 'Jose', 'idade': 23, 'email': 'jose@fiap.com.br'}, 'pessoa3': {'nome': 'Maria', 'idade': 29, 'email': 'maria@fiap.com.br'}}
```

Exibição do objeto JSON:

```
{
  "pessoa1": {
    "nome": "Edson",
    "idade": 48,
    "email": "eds@fiap.com.br"
  },
  "pessoa2": {
    "nome": "Jose",
    "idade": 23,
    "email": "jose@fiap.com.br"
  },
  "pessoa3": {
    "nome": "Maria",
    "idade": 29,
    "email": "maria@fiap.com.br"
  }
}
```

Exibição bruta do dicionário:

```
{'pessoa1': {'nome': 'Edson', 'idade': 48, 'email': 'eds@fiap.com.br'}, 'pessoa2': {'nome': 'Jose', 'idade': 23, 'email': 'jose@fiap.com.br'}, 'pessoa3': {'nome': 'Maria', 'idade': 29, 'email': 'maria@fiap.com.br'}}
```

Comando de prompt 18 – Exibição na íntegra – parte 1/2

Fonte: Elaborado pelo autor (2023)

Manipulação de arquivos texto e JSON

Exibição do objeto:

```
{  
  "pessoa1": {  
    "nome": "Edson",  
    "idade": 48,  
    "email": "eds@fiap.com.br"  
  },  
  "pessoa2": {  
    "nome": "Jose",  
    "idade": 23,  
    "email": "jose@fiap.com.br"  
  },  
  "pessoa3": {  
    "nome": "Maria",  
    "idade": 29,  
    "email": "maria@fiap.com.br"  
  }  
}
```

Exibição formatada dos dados:

```
Registro.....: pessoa1  
  nome:Edson  
  idade:48  
  email:eds@fiap.com.br  
Registro.....: pessoa2  
  nome:Jose  
  idade:23  
  email:jose@fiap.com.br  
Registro.....: pessoa3  
  nome:Maria  
  idade:29  
  email:maria@fiap.com.br
```

Comando de prompt 19 – Exibição na íntegra – parte 2/2
Fonte: Elaborado pelo autor (2023)

Manipulação de arquivos texto e JSON

Com a construção desta solução pudemos contemplar os conteúdos: Tupla, Lista, Dicionários, arquivos texto e JSON na prática.

Todos os sistemas profissionais utilizam algum meio de gravação de dados, geralmente através de Sistemas gerenciadores de banco de dados, como o Oracle ou SQL Server, mas este é assunto para o próximo episódio.

Quem estuda cria asas, então voem que nos encontraremos lá em cima.

EDSON

REFERÊNCIAS

PYTHON. **Built-in Functions.** 2022. Disponível em: <https://docs.python.org/3/library/functions.html#open>. Acesso em: 16 jan. 2023.

UMA INTRODUÇÃO A ASCII E UNICODE. **Treina Web.** 2021. Disponível em: <https://www.treinaweb.com.br/blog/uma-introducao-a-ascii-e-unicode>. Acesso em: 16 jan. 2023.

EXEMPLO

GLOSSÁRIO

Arquivo texto	Arquivo gravado com texto sem formatações.
Arquivo JSON	Tipo de arquivo facilitador para integração de dados em sistemas.