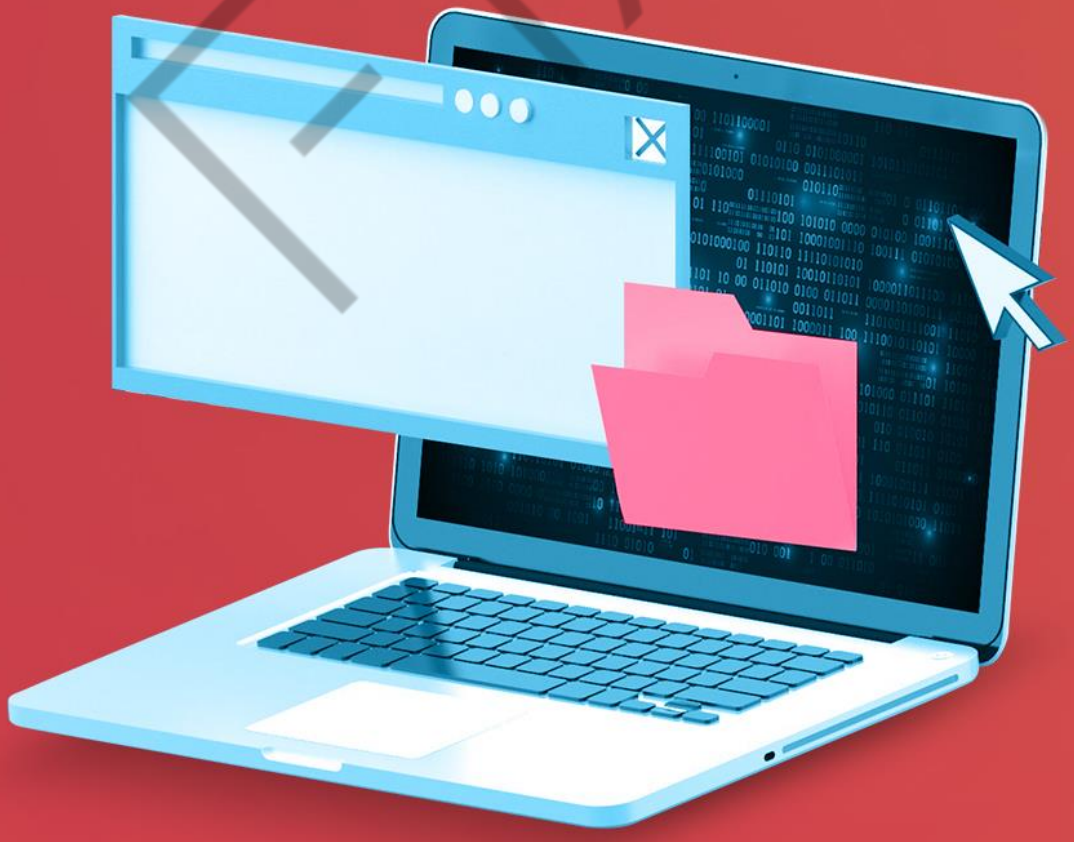
FIAP ON

MODELING

QUEM ARQUIVA AMIGO É

ANDRÉ DE FREITAS DAVID



09

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 QUEM ARQUIVA AMIGO É

1.1 Saindo do temporário

Você já parou para pensar na longa jornada que trilhou até aqui e no quanto já aprendeu sobre programação?

Usando variáveis, listas, dicionários e tuplas, aprendemos a lidar com muitos e muitos dados digitados por usuários imaginários. Mas tudo isso é um pouco... temporário, não é?

Figura 1 – Assim como letras escritas na praia se apagam, nossas variáveis são perdidas com o encerramento de um programa
Fonte: Retirado de Pixabay (2020)

Quando nossos scritps são encerrados, os dados preenchidos nas nossas estruturas são perdidos. Afinal de contas, elas se encontram na memória RAM do computador.

Neste capítulo vamos, finalmente, aprender a armazenar nossos dados de maneira mais definitiva: o Python será nosso aliado na manipulação de arquivos e na escrita no formato JSON!

Prepare-se, pois, ao final do estudo, você terá aprendido um recurso muito usado no mercado de trabalho.

1.2 Arquivos? Que tipo de arquivos?

As linguagens de programação, de maneira geral, possuem a capacidade de gerar arquivos e salvá-los em uma memória não volátil, como um HD ou um pen-drive. Mas, qual o formato de arquivo?

O nosso computador representa tudo por meio de bits (zeros e uns) e, quando aglomeramos esses bits, por meio de bytes. Uma letra do alfabeto, por exemplo, pode ser representada por meio de 1 byte (8 bits) no sistema ASCII.

“Professor, mas esse papo está muito técnico! Eu só quero saber como armazeno um arquivo no meu HD.”, você pode estar pensando. Mas entender esse papo técnico ajuda a entender que, de maneira geral, o que as linguagens de programação permitem é gravar *bytes*. E um arquivo de texto, por exemplo, é apenas um *amontoado de bytes*. Um arquivo Excel é uma sequência de bits organizados de uma forma que o Excel compreenda.

Dessa forma, vamos focar nosso aprendizado em gravar e abrir arquivos de texto usando o Python. E a chave principal desse aprendizado será uma função chamada *open*.

1.2.1 A função open

A função *open* é uma função nativa do Python, que retorna para dentro do script um objeto do tipo arquivo.

Esse é um conceito um pouco abstrato, pois o objeto do tipo arquivo pode ser um arquivo já existente ou um novo arquivo que estamos criando.

Melhor do que imaginar é testar, não é mesmo? Então, vamos observar o script do Código-fonte “Usando a função open para criar um arquivo”.

```
#usando a função open para criar um objeto do tipo arquivo
arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")
```

Código Fonte 1 – Usando a função open para criar um arquivo
Fonte: Elaborado pelo autor (2020)

Ao tentar executar esse script, porém, você deve ficar extremamente decepcionado. Afinal, nossa tela será tomada por um erro desagradável. Veja, na Figura “Erro ao tentar executar o script para abrir um arquivo”.

```
Traceback (most recent call last):
  File "C:/Users/andre/PycharmProjects/capitulos/arquivo.py", line 2, in <module>
    arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\arquivos\\arquivo_de_texto.txt'
```

Figura 2 – Erro ao tentar executar o script para abrir um arquivo
Fonte: Elaborado pelo autor (2020)

O que esse erro está indicando é que não há um arquivo ou diretório chamado “c:\arquivos\arquivo_de_texto.txt”. Ou seja, o que a função open tentou fazer foi **procurar** um arquivo chamado “arquivo_de_texto.txt” no diretório c:\arquivos e, como o arquivo não existe, o erro foi causado.

Que tal criarmos esse arquivo para vermos o que acontece agora? Criar, dentro da unidade c:\ do seu computador (ou outro diretório em que preferir), uma pasta chamada *arquivos* e, dentro dela, criar um arquivo de texto chamado *arquivo_de_texto.txt*. Observe as Figuras “Criando uma pasta chamada ‘arquivos’ na unidade C”, “Criando um arquivo de texto no diretório ‘c:\arquivos’” e “Nomeando o arquivo criado como ‘arquivo_de_texto’”.

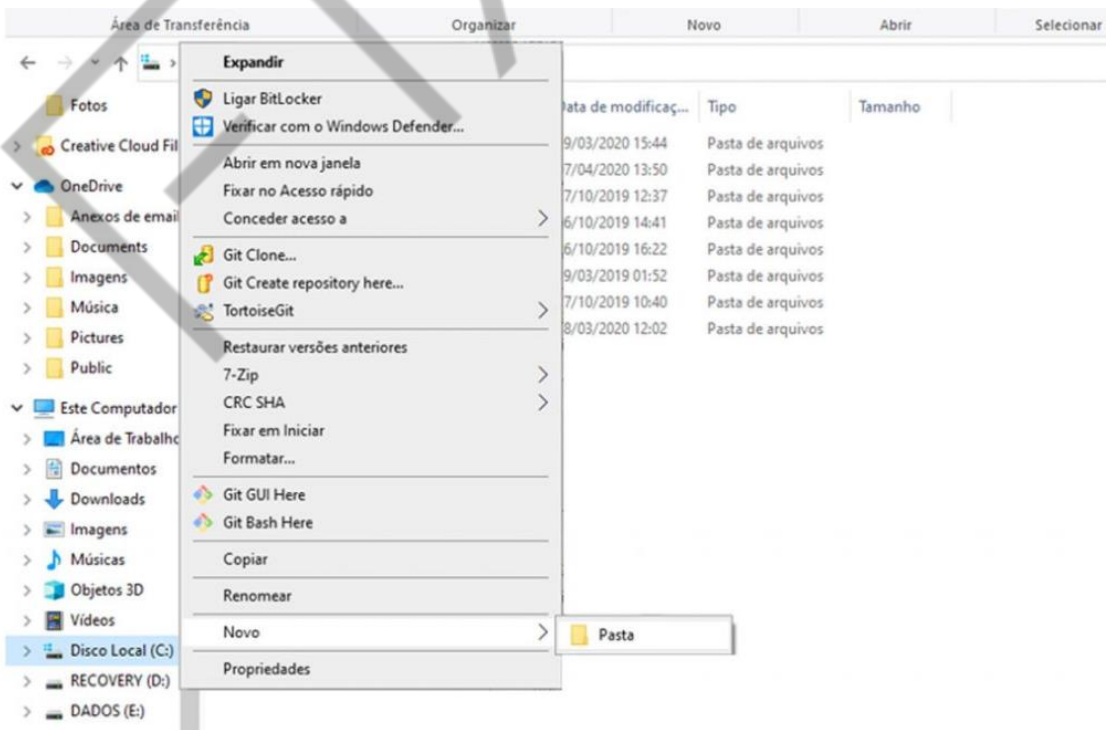


Figura 3 – Criando uma pasta chamada “arquivos” na unidade C:
Fonte: Elaborado pelo autor (2020)

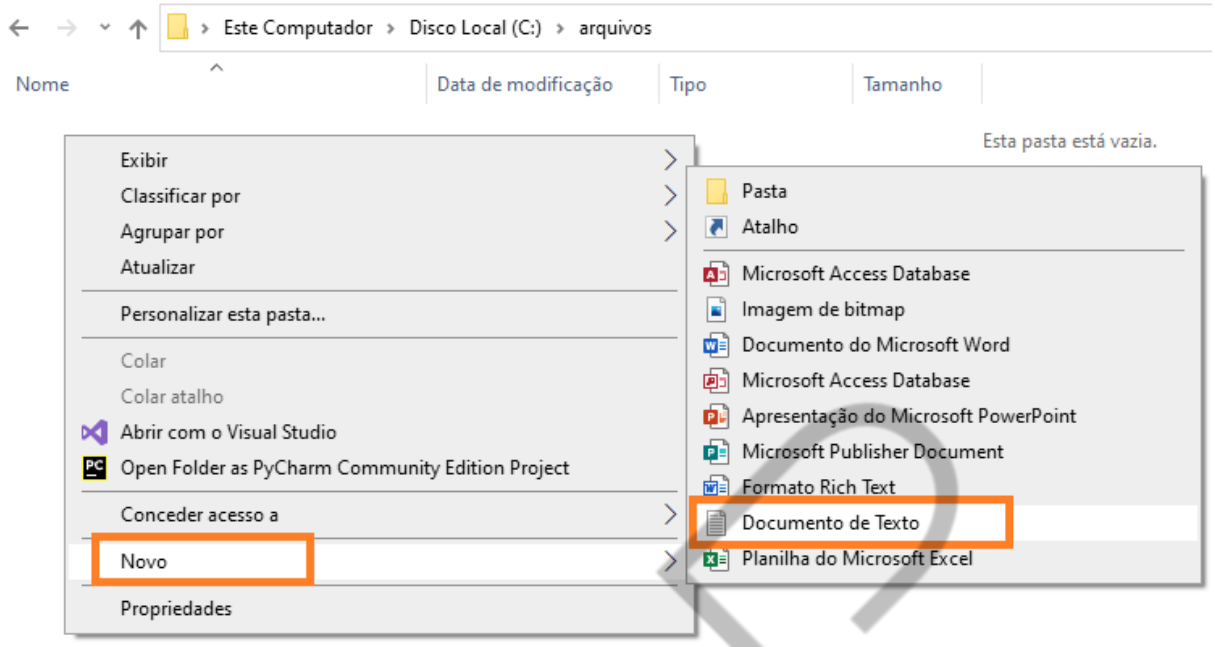


Figura 4 – Criando um arquivo de texto no diretório “c:\arquivos”
Fonte: Elaborado pelo autor (2022)

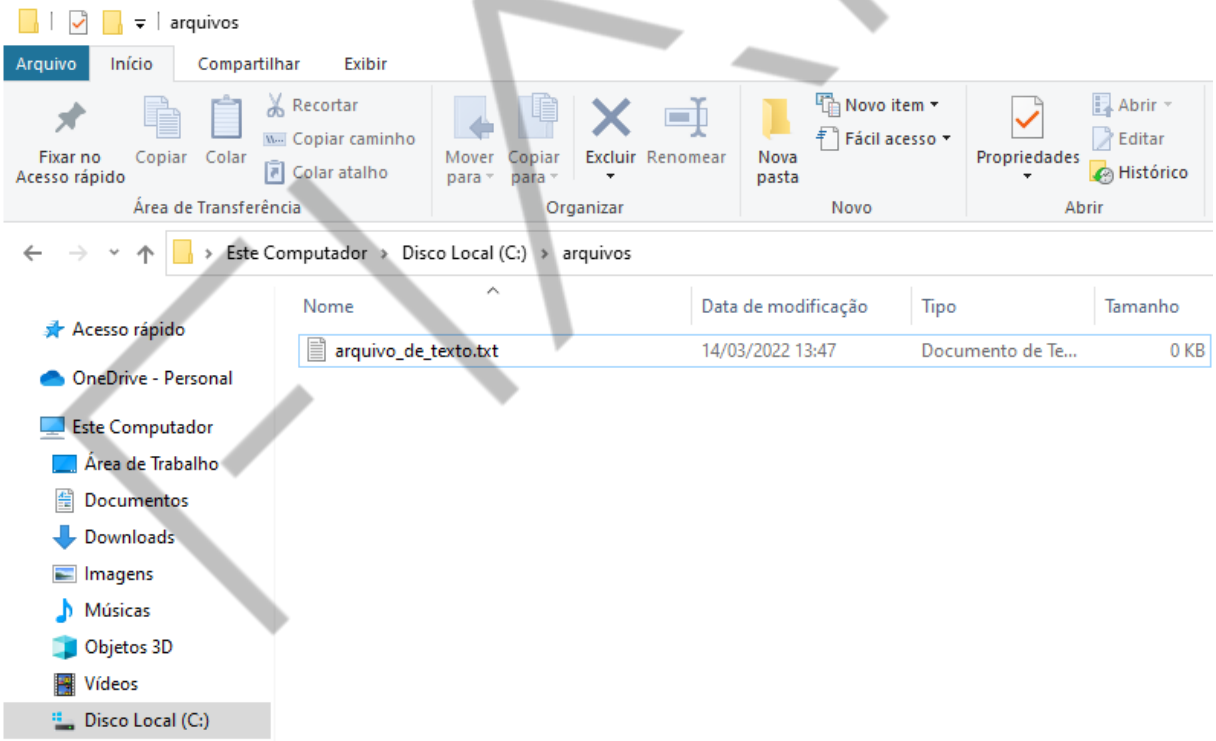


Figura 5 – Nomeando o arquivo criado como “arquivo_de_texto”
Fonte: Elaborado pelo autor (2022)

Agora, ao executar o script, ele não retorna erros e, se usarmos a boa e velha função *type*, descobriremos que o tipo do objeto é *TextIOWrapper*.

```
#usando a função open para criar um objeto do tipo arquivo
arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

#verificando o tipo do objeto arquivo
print(type(arquivo))
```

Código Fonte 2 – Verificando o tipo do objeto que recebeu o arquivo aberto
Fonte: Elaborado pelo autor (2020)

“TextIOWrapper? O que é isso? O que aconteceu com a boa e velha String?”. Calma, não precisa se desesperar. Para nós, neste momento do nosso estudo, podemos considerar apenas que se trata de um objeto que representa nosso arquivo, ok?

Se quisermos printar o conteúdo desse arquivo, podemos usar a função print, mas de nada adiantará se o arquivo estiver em branco. Portanto, usaremos o bloco de notas para inserir alguns dados dentro desse arquivo de texto.

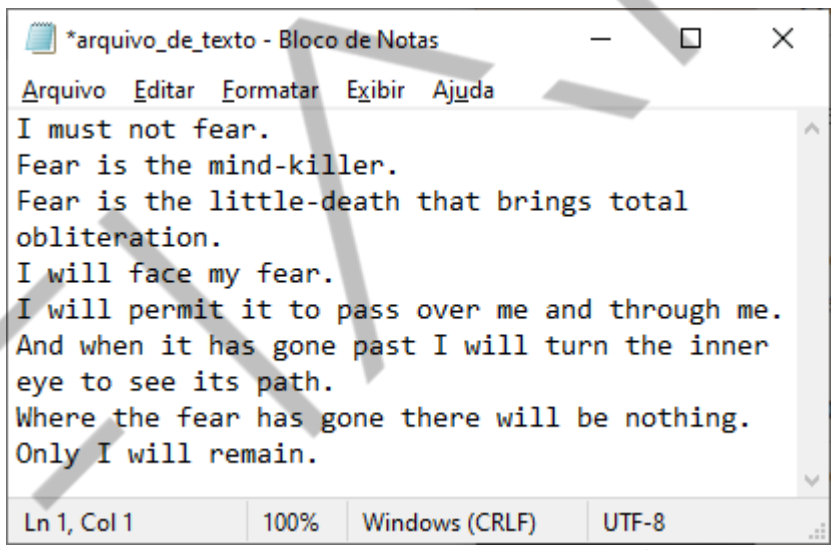


Figura 6 – Preenchendo o nosso arquivo de texto com algum conteúdo, usando o Bloco de Notas
Fonte: Elaborado pelo autor (2020)

Agora, sim, temos um cenário adequado para abrir um arquivo e exibir seu conteúdo! Afinal, criamos o arquivo no diretório correto e garantimos que tenha algo gravado dentro dele.

Com tudo pronto, o Código-fonte “Abrindo um arquivo de texto e printando o conteúdo do objeto gerado” deve retornar resultados interessantes.

```
#usando a função open para criar um objeto do tipo arquivo
arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

#printando o conteúdo do objeto arquivo
print(arquivo)
```

Código Fonte 3 – Abrindo um arquivo de texto e printando o conteúdo do objeto gerado
Fonte: Elaborado pelo autor (2020)

Perceba que o que apareceu no seu console não é o texto que salvamos dentro do arquivo, mas o caminho do arquivo com algumas informações a mais, como o modo de abertura e o tipo de codificação, como observamos na Figura “Resultado da execução do script anterior”.

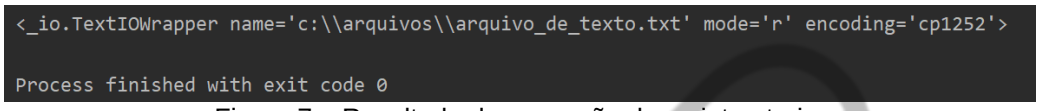


Figura 7 – Resultado da execução do script anterior
Fonte: Elaborado pelo autor (2020)

Lembra que esse objeto *representa* nosso arquivo? Se quisermos printar o conteúdo dele, devemos usar um método chamado *read*, conforme o Código-fonte “abrindo um arquivo de texto e printando o conteúdo do objeto gerado com o método *read”.*

```
#usando a função open para criar um objeto do tipo arquivo
arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

#printando o conteúdo do objeto arquivo
print(arquivo.read())
```

Código Fonte 4 – Abrindo um arquivo de texto e printando o conteúdo do objeto gerado com o método *read*
Fonte: Elaborado pelo autor (2020)

Apesar de termos passado por essa pequena montanha russa para chegar a este ponto e de você entender bem o que envolve abrir um arquivo de texto, note que nosso programa precisou de apenas duas linhas para funcionar: uma para abrir o arquivo e outra para printar seu conteúdo.

Pode ser, porém, que você não queira analisar o arquivo inteiro de uma só vez e é aí que a abertura de arquivos começa a ficar interessante: podemos ler uma linha do arquivo por vez.

Para isso, o método *readline()* pode ser usado para ler uma linha do arquivo (Código-fonte “Exibindo uma linha do arquivo usando o método readline()”).

```
#usando a função open para criar um objeto do tipo arquivo
arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

#printando uma linha do arquivo
print(arquivo.readline())
```

```
     #printando outra linha do arquivo
     print(arquivo.readline())
```

Código Fonte 5 – Exibindo uma linha do arquivo usando o método readline()
Fonte: Elaborado pelo autor (2020)

Com isso, podemos até mesmo criar um loop que passe por todas as linhas, uma de cada vez. Observe o Código-fonte “Exibindo uma linha do arquivo por vez usando o loop for e o método readlines()”.

O método readlines() faz a leitura de todas as linhas de um arquivo e retorna uma lista das linhas, separando-as umas das outras com vírgulas.

```
     #usando a função open para criar um objeto do tipo arquivo
     arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

     #Exibindo uma linha por vez, utilizando o loop for e o
método readlines()
     for linha in arquivo.readlines():
         print(linha)
```

Código Fonte 6 – Exibindo uma linha do arquivo por vez usando o loop for e o método readlines()
Fonte: Elaborado pelo autor (2020)

Aposto que você já entendeu aonde queremos chegar: se nós usarmos uma lista, podemos colocar cada linha do nosso arquivo como um item e, depois, manipular como quisermos, ao longo do script.

```
     #usando a função open para criar um objeto do tipo arquivo
     arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

     #Passando o conteúdo do arquivo para uma lista
     linhas_do_arquivo = arquivo.readlines()

     #comprovando o tipo do objeto linhas_do_arquivo
     print("Ei! Eu consegui transformar meu arquivo em uma {}
".format(type(linhas_do_arquivo)))

     #colocando a lista em ordem alfabética
     linhas_do_arquivo.sort()

     #Exibindo nossa lista, agora em ordem alfabética
     print(linhas_do_arquivo)
```

Código Fonte 7 – Colocando o conteúdo do arquivo em uma list
Fonte: Elaborado pelo autor (2020)

Toda essa manipulação pode ser muito útil na vida de um programador, como você pode imaginar, mas antes de continuarmos avançando, precisamos fazer algo

importantíssimo: fechar o arquivo que foi aberto. Se não fizermos isso, e o arquivo continuar aberto enquanto o script roda, poderão ocorrer falhas que levarão os dados a serem corrompidos.

Para realizar o fechamento do arquivo, usaremos o método *close()* imediatamente após consumirmos o conteúdo do arquivo (Código-fonte “Fechando um arquivo com o método *close()*”.

```
     #usando a função open para criar um objeto do tipo arquivo
     arquivo = open("c:\\arquivos\\arquivo_de_texto.txt")

     #Exibindo uma linha por vez, utilizando o loop for e o
método readlines()
     for linha in arquivo.readlines():
         print(linha)
     arquivo.close()
```

Código Fonte 8 – Fechando um arquivo com o método *close()*
Fonte: Elaborado pelo autor (2020)

Olha que legal! Já conseguimos abrir arquivos que estão salvos no nosso HD, ler seu conteúdo e até passar para estruturas de dados que já aprendemos a manipular. Chegou a hora de aprendermos a salvar um arquivo!

1.2.2 Usando a função open para... salvar?

A função *open* foi usada por nós para abrir um arquivo, mas ela também será responsável por nos ajudar, quando quisermos salvar conteúdos que estão no nosso programa.

Isso ocorre porque, quando usamos a função open, além do *caminho* do arquivo, podemos e devemos indicar a forma como esse arquivo será manipulado pelo nosso script. Isso é feito indicando um segundo parâmetro, após o local do arquivo, e ele pode assumir os seguintes valores:

- 'r' abrir para leitura (modo padrão).
- 'w' abrir para a escrita, sobrescrevendo o conteúdo.
- 'x' abrir para a criação de arquivo, gerando uma falha se existir um arquivo de mesmo nome.

- 'a' abrindo para escrita, anexando o novo conteúdo ao final do conteúdo já existente no arquivo.
- 'b' abrir em modo binário.
- 't' abrir em modo de texto (modo padrão).
- '+' abrir para atualização (escrita e leitura).

Vamos ver um exemplo de criação de um novo arquivo, com um novo conteúdo?

Para isso, escreveremos algum conteúdo em uma variável do tipo String e, depois, usaremos o método open para abrir um arquivo em modo de escrita. Quando quisermos escrever o conteúdo da variável no arquivo, usaremos o método *write* e, por fim, fecharemos o arquivo.

```
     #Criando uma variável de texto
     conteudo = "Estou testando criar um arquivo de texto. 
Então, estou... textando?"

     #usando a função open para criar um objeto do tipo arquivo
     arquivo = open("c:\\arquivos\\novo_arquivo.txt", "w")

     #Escrevendo o conteúdo da variável conteudo dentro do 
arquivo
     arquivo.write(conteudo)

     #fechando o arquivo
     arquivo.close()
```

Código Fonte 9 – Criando um novo arquivo e escrevendo nele o conteúdo de uma variável String
Fonte: Elaborado pelo autor (2020)

Ao final da execução do script, você notará que um novo arquivo, chamado *novo_arquivo.txt*, apareceu no diretório *c:\arquivos*, e que o conteúdo dele é o mesmo que escrevemos na variável string!

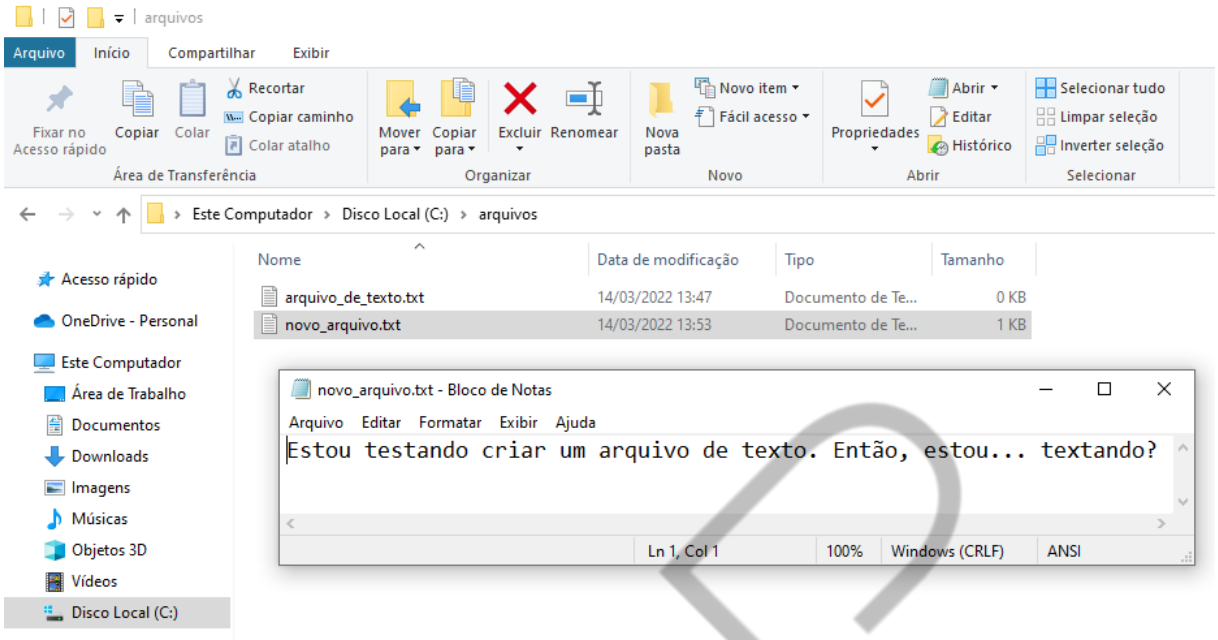


Figura 8 – Resultado da execução do script que cria um arquivo de texto
Fonte: Elaborado pelo autor (2022)

Você notará que, se executar o script diversas vezes, o conteúdo do arquivo gerado será sempre o mesmo. E também que, se mudarmos a forma de abertura do arquivo, de *w* (que cria e substitui o conteúdo antigo) para *a* (que cria um arquivo e anexa o novo conteúdo no final), a cada execução do script o novo conteúdo será anexado ao final do conteúdo antigo.

```
    #Criando uma variável de texto
    conteudo = "Estou testando criar um arquivo de texto.
Então, estou... textando?"

    #usando a função open para criar um objeto do tipo arquivo
    arquivo = open("c:\\arquivos\\novo_arquivo.txt", "a")

    #Escrevendo o conteúdo da variável conteudo dentro do
arquivo
    arquivo.write(conteudo)

    #fechando o arquivo
    arquivo.close()
```

Código Fonte 10 – Criando um novo arquivo no modo *a* e escrevendo no final dele o conteúdo de uma variável String
Fonte: Elaborado pelo autor (2020)

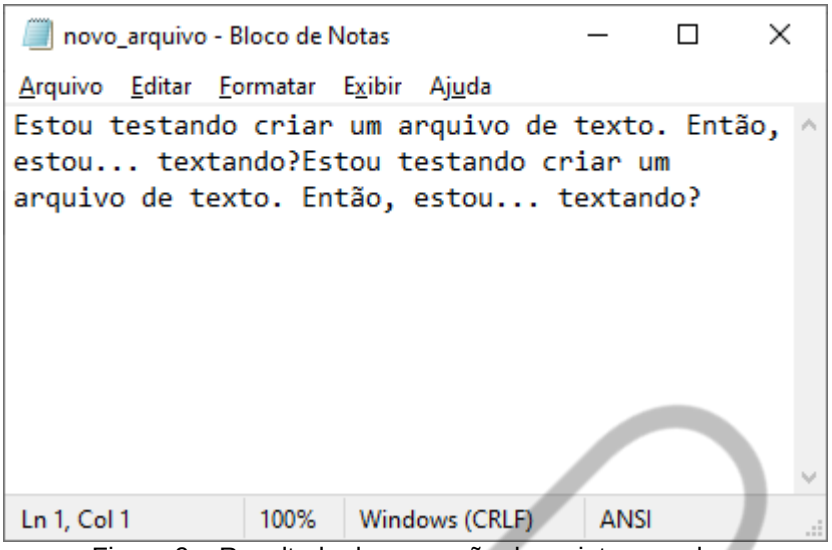


Figura 9 – Resultado da execução do script no modo *a*
Fonte: Elaborado pelo autor (2020)

O quadro a seguir apresenta uma lista com as descrições dos principais métodos que podemos utilizar com arquivos no Python.

Método	Descrição	Exemplo
close()	Fecha o arquivo.	arquivo.close()
flush()	Libera o buffer interno.	arquivo.flush()
read()	Faz a leitura do arquivo inteiro e retorna uma string.	arquivo.read()
readline()	Faz a leitura e retorna uma linha de um arquivo.	arquivo.readline()
readlines()	Faz a leitura do arquivo inteiro e retorna uma lista com cada linha.	arquivo.readlines()
seek()	Permite controlar a posição do cursor no arquivo.	arquivo.seek(coluna, posição)
tell()	Retorna a posição atual do cursor no arquivo.	arquivo.tell()
truncate()	Redimensiona (trunca) o arquivo para o tamanho especificado.	arquivo.truncate(tamanho)
writable()	Retorna o valor True se pudermos escrever no arquivo.	arquivo.writable()
write()	Grava no arquivo.	arquivo.write(string)
writelines()	Grava cada elemento de uma lista de strings no arquivo.	arquivo.writelines(lista_string)

Quadro 1 – Métodos que podem ser utilizados com arquivos
Fonte: Elaborado pelo autor (2022)

Pronto! Você já sabe o suficiente para criar programas que são capazes de salvar conteúdo e de abrir o conteúdo salvo anteriormente.

Você vai se deparar, agora, com a escolha entre salvar os arquivos em um padrão próprio (Um valor por linha? Dividir os valores com algum caractere especial?) ou com um padrão estabelecido, como o CSV (valores separados por vírgula) ou o JSON. Nós exploraremos apenas o formato JSON, dado o seu uso cada vez mais comum na criação de APIs e em outras aplicações.

1.3 O formato JSON

O *problema* de definir padrões para trocar dados entre aplicações e computadores existe desde sempre. Afinal de contas, de nada adianta criar um programa de agenda que gere os dados em um formato de texto parecido com: “nome do contato, e-mail do contato, telefone do contato”, se outro software só for capaz de interpretar dados no formato “nome do contato ***** e-mail do contato ***** telefone do contato”.

Para solucionar esse problema é que surgiu, por exemplo, o formato XML – usado até hoje nas notas fiscais eletrônicas, e tantos outros formatos específicos.

Nos anos 2000, porém, Douglas Crockford propôs um padrão legível a seres humanos, que se baseie no formato *atributo-valor*. Após algumas revisões ao longo das décadas, o formato JSON caiu no gosto dos desenvolvedores e hoje já se tornou obrigatório em qualquer aplicação de grande porte.

Para provar o quão legível é o formato JSON, vou escrever, no Código-fonte “Exemplo de dados em JSON”, um conjunto de dados nesse padrão e apostar que você vai conseguir entender sem nenhuma explicação.

```
{
   "Clark Kent":{
      "Celular":"123456",
      "Email":"super@krypton.com"
   },
   "Bruce Wayne":{
      "Celular":"654321",
      "Email":"bat@caverna.com.br"
   }
}
```

Código Fonte 11 – Exemplo de dados em JSON
Fonte: Elaborado pelo autor (2020)

Você consegue visualizar que o atributo *Clark Kent* possui como valor um objeto, que possui como valor o atributo *celular* e, como valor, o conteúdo *123456*, além do atributo *email*, que possui como conteúdo o valor o conteúdo *super@krypton.com*, não é mesmo?

Essa é a beleza do formato *atributo-valor*, e aí está uma das grandes forças do JSON.

Existem vários formatos possíveis para dados no padrão JSON e você pode encontrar todos os detalhes no site oficial <http://json.org/json-pt.html>.

Nosso enfoque será, porém, em converter estruturas que já conhecemos para esse formato.

1.3.1 Dicionário para JSON

Com certeza, você percebeu o quanto um JSON se parece com os dicionários da linguagem Python. Isso não é à toa, já que ambas as estruturas partem do mesmo princípio.

É, inclusive, tão esperado que um desenvolvedor Python queira gerar dados em formato JSON, que existe um módulo padrão dedicado a essa tarefa, e ele é chamado... json!

O módulo json conta com inúmeros métodos úteis e o primeiro deles é o método *dumps()*, que converte um objeto para o formato json.

Dê uma olhada no Código-fonte “Exibindo uma linha do arquivo por vez usando o loop for e o método readline()” para ter uma ideia da simplicidade desse processo.

```
#importando o módulo json
import json

#criando um dicionário para usarmos como exemplo
contatos = {
    "Clark Kent":
        {"Celular":"123456",
         "Email":"super@krypton.com"},
    "Bruce Wayne":
        {"Celular":"654321",
         "Email":"bat@caverna.com.br"}
}
```

```
#convertendo o dicionário para uma string o formato json
final = json.dumps(contatos)

#exibindo a string convertida
print(final)
```

Código Fonte 12 – Convertendo um dicionário para o formato JSON
Fonte: Elaborado pelo autor (2020)

A string que foi printada no seu terminal já está em formato JSON, mas não está *bonita*.

Podemos garantir o espaçamento correto dos dados adicionando o parâmetro *indent* ao método dumps.

```
#importando o módulo json
import json

#criando um dicionário para usarmos como exemplo
contatos = {
    "Clark Kent":
        {"Celular":"123456",
         "Email":"super@krypton.com"},
    "Bruce Wayne":
        {"Celular":"654321",
         "Email":"bat@caverna.com.br"}
}

#convertendo o dicionário para uma string o formato json
final = json.dumps(contatos, indent=4)

#exibindo a string convertida
print(final)
```

Código Fonte 13 – Convertendo um dicionário para o formato JSON e garantindo o espaçamento correto
Fonte: Elaborado pelo autor (2020)

Se quiser ter certeza de que o JSON gerado esteja correto, copie o resultado e utilize uma das diversas ferramentas on-line disponíveis para isso. A página <http://jsonviewer.stack.hu/> permite colar um JSON e visualizar os dados que ele contém.

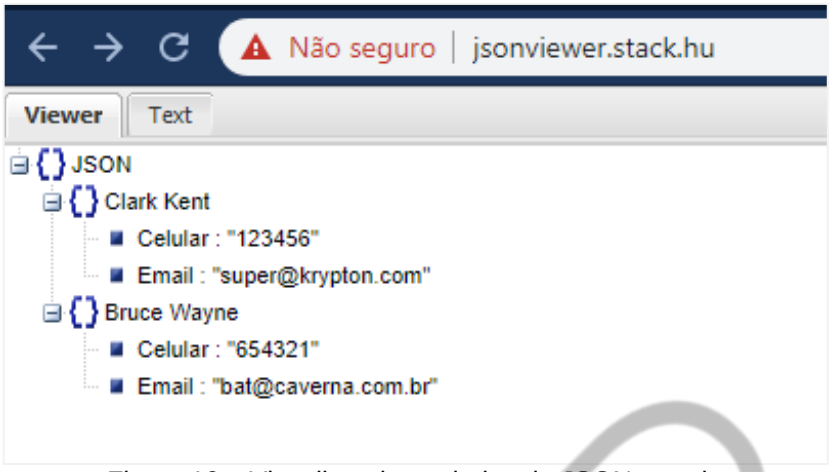


Figura 10 – Visualizando os dados do JSON gerado
Fonte: Elaborado pelo autor (2020)

É claro que simplesmente gerar um arquivo JSON na tela do computador resolverá apenas parte dos nossos problemas. Usaremos nossas habilidades para gravar um arquivo de texto nesse formato:

```
#importando o módulo json
import json

#criando um dicionário para usarmos como exemplo
contatos = {
    "Clark Kent":
        {"Celular":"123456",
         "Email":"super@krypton.com"},
    "Bruce Wayne":
        {"Celular":"654321",
         "Email":"bat@caverna.com.br"}
}

#convertendo o dicionário para uma string o formato json
final = json.dumps(contatos, indent=4)

#criando um arquivo
arquivo = open("c:\\arquivos\\agenda.json", "w")

#escrevendo o JSON dentro do arquivo
arquivo.write(final)

#fechando o arquivo
arquivo.close()
```

Código Fonte 14 – Salvando um arquivo JSON com o conteúdo de um dicionário
Fonte: Elaborado pelo autor (2020)

Já aprendemos a gerar um arquivo com um conteúdo em formato JSON. A *fronteira final*, agora, é criar um arquivo capaz de abrir JSONs!

1.3.2 JSON para dicionário

Abrir um arquivo de texto, tenha ele a extensão txt ou qualquer outra, é tarefa fácil para nós. Nossa missão, agora, será converter o conteúdo desse arquivo para um dicionário, de forma que seja possível manipular seus dados dentro do programa.

Quem nos ajudará nessa tarefa é o próprio módulo json, mas com o método *loads,* que converte uma string em dicionário.

O script a seguir está escrito de maneira didática e não visa o melhor desempenho, mas executa exatamente os passos indicados no Código-fonte “Abrindo um arquivo JSON e convertendo para dicionário”.

```
     import json
     #Criando uma variável de texto
     conteudo = "Estou testando criar um arquivo de texto. 
Então, estou... textando?"

     #usando a função open para criar um objeto do tipo arquivo
     arquivo = open("c:\\arquivos\\agenda.json")

     #colocando o conteúdo do arquivo em uma variável do tipo 
string
     conteudo_do_arquivo = arquivo.read()

     #fechando o arquivo
     arquivo.close()

     #usando o método loads para converter uma string no 
formato json em um dicionário
     agenda = json.loads(conteudo_do_arquivo)

     #comprovando que o objeto agenda é do tipo dicionário
     print("O tipo do objeto agenda é 
{}".format(type(agenda)))
```

Código Fonte 15 – Abrindo um arquivo JSON e convertendo para dicionário
Fonte: Elaborado pelo autor (2020)

Se você dominar as estruturas de dados que estudamos e conseguir salvar e abrir arquivos, terá um futuro brilhante como programador. Agora, é estudar e partir para novos desafios!

1.4 Um pequeno extra

Aprendemos a manipular arquivos com o método *open*, mas você vai descobrir que boa parte dos programadores Python usa outra forma de lidar com esse desafio.

O comando *with* é usado para garantir que um recurso que foi aberto seja finalizado. Assim, você, programador, não precisará se preocupar com o *close* ao final da manipulação de um arquivo.

Veja, no Código-fonte “Abrindo um arquivo de texto utilizando o with”, como fica o script que abre um arquivo e printa os dados na tela ao usar essa técnica.

```
    #o with usará o open para abrir o arquivo indicado, dentro
do objeto arquivo, e fará sozinho o encerramento do acesso
quando a última linha de código, dentro dele, for executada
    with open("c:\\arquivos\\arquivo_de_texto.txt", "r") as
arquivo:
        #aqui devemos escrever todos os códigos que usam o
arquivo aberto, pois, após a última linha de código, dentro
dessa estrutura, o arquivo será automaticamente encerrado
        print(arquivo.read())
```

Código Fonte 16 – Abrindo um arquivo de texto utilizando o with
Fonte: Elaborado pelo autor (2020)

Para fazer a escrita de um arquivo usando essa mesma facilidade, basta modificarmos um pouco o script. Observe o Código-fonte “Gravando um arquivo de texto usando o *with*”.

```
    #o with usará o open para abrir o arquivo indicado, dentro
do objeto arquivo, e fará sozinho o encerramento do acesso
quando a última linha de código, dentro dele, for executada
    with open("c:\\arquivos\\arquivo_de_texto.txt", "w") as
arquivo:
        #aqui devemos escrever todos os códigos que usam o
arquivo aberto, pois após a última linha de código, dentro
dessa estrutura, o arquivo será automaticamente encerrado
        arquivo.write("May the force be with you")
```

Código Fonte 17 – Gravando um arquivo de texto usando o with
Fonte: Elaborado pelo autor (2020)

Lembre-se: os estudos de um programador nunca terminam. Acostume-se a ler, conversar com colegas, pesquisar e, principalmente, codificar.

A cada nova ideia que você tiver, um novo mundo de possibilidades se abrirá!

REFERÊNCIAS

PIVA, D. J. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, S.; RISSETTI, G. **Lógica de Programação e Estrutura de Dados**. São Paulo: Pearson Prentice Hall, 2009.

RAMALHO, L. **Python Fluente**: **programação clara, concisa e eficaz**. São Paulo: Novatec, 2015.