MANIPULAÇÃO DE ARQUIVOS

Lendo dados de um arquivo

Uma quantidade incrível de dados está disponível em arquivos-texto. Os arquivos-texto podem conter dados meteorológicos, de tráfego, socioeconômicos, trabalhos literários e outros.

Ler dados de um arquivo é particularmente útil em aplicações de análise de dados, mas também se aplica a qualquer situação em que você queira analisar ou modificar informações armazenadas em um arquivo. Por exemplo, podemos escrever um programa que leia o conteúdo de um arquivo-texto e reescreva o arquivo com uma formatação que permita a um navegador exibi-lo.

Então, manipular um arquivo significa ler, escrever ou modificar dados armazenados em um arquivo, de diversos tipos, como:

- .txt
- .csv
- .json
- .log
- etc.

Vamos começar com algo simples, apenas um arquivo com algumas linhas contendo o valor de pi com trinta casas decimais, dez casas por linha.

Colocar em um bloco de notas e salvar como '**pi_digits.txt**':

```
3.1415926535
8979323846
2643383279
```

```
# lendo um arquivo

with open('pi_digits.txt') as linhas_arquivo:
  conteudo = linhas_arquivo.read()
  print(conteudo)
```

```
3.1415926535
8979323846
2643383279
```

- Função **open()**:
  - Para realizar qualquer tarefa com um arquivo, mesmo que seja apenas exibir o seu conteúdo, você precisará inicialmente abrir o arquivo para acessá-lo.
  - A função open() precisa de um argumento: o nome do arquivo que você quer abrir.
- A palavra reservada **with** fecha o arquivo depois que não for mais necessário acessá-lo.
- o método **read()** na segunda linha de nosso programa serve para ler todo o conteúdo do arquivo e armazená-lo em uma longa string em **'conteudo'**. Quando exibimos o valor de contents, vemos o arquivo-texto completo.

Obs.:

> Você poderia abrir e fechar o arquivo chamando open() e close(), mas se um bug em seu programa impedir que a instrução close() seja executada, o arquivo não será fechado.

> Isso pode parecer trivial, mas arquivos indevidamente fechados podem provocar perda de dados ou estes podem ser corrompidos.

> Além disso, se close() for chamado cedo demais em seu programa, você se verá tentando trabalhar com um arquivo fechado (um arquivo que não pode ser acessado), o que resultará em mais erros.

Obs. 2:

> A saída no programa e o arquivo original existe uma pequena diferença, que é justamente que o método *read()* faz com que devolva uma string vazia quando alcança o final, ou seja, é adicionado uma linha em branco no final do arquivo.

> Para remover é fácil, rstrip() pode ser usada na instrução print.

Utilizando *open()* e *close()*

- Sintaxe básica do open:

```
open(nome_do_arquivo, modo, encoding)
```

```
# open()
arquivo = open('exemplo.txt', 'r')
```

Os modos de abertura são maneiras que pode-se utilizar para abrir um arquivo, são eles:

Modo	Descrição
`"r"`	Leitura (erro se não existir)
`"w"`	Escrita (cria ou sobrescreve)
`"a"`	Anexar conteúdo (append)
`"x"`	Cria o arquivo (erro se existir)
`"r+"`	Leitura e escrita

```
arquivo = open("exemplo.txt", "w")
arquivo.write("Olá, mundo!")
arquivo.close()
```

```
arquivo = open('exemplo.txt', 'r')
cont = arquivo.read()
print(cont)
arquivo.close()
```

```
Olá, mundo!
```

! Sempre lembrar de utilizar *close()* para fechar o arquivo toda vez que finalizar a operação.

Lendo dados linha a linha

Podemos usar um laço for no objeto arquivo para analisar cada uma de suas linhas, uma de cada vez:

```
with open('pi_digits.txt') as linhas_arquivo:
  for linha in linhas_arquivo:
    # print(linha.rstrip())
    print(linha)
```

```
3.1415926535

8979323846

2643383279
```

Criando uma lista de linhas

```
with open('pi_digits.txt') as linhas_arquivo:
  linhas = linhas_arquivo.readlines()

linhas

for l in linhas:
  print(l.rstrip())
```

```
3.1415926535
8979323846
2643383279
```

Trabalhando com o conteúdo de um arquivo

Depois de ler um arquivo em memória, você poderá fazer o que quiser com esses dados.

```
file_name = 'pi_digits.txt'

with open(file_name) as file_object:
  linhas = file_object.readlines()

# Concatenando todos digitos do arquivo em uma única string
```

```
pi_string = ''
for l in linhas:
  pi_string += l.rstrip()

print(pi_string)
print(len(pi_string)) # Contando qual o tamanho da string
```

```
3.14159265358979323846264338327932
32
```

ATENÇÃO: Quando Python lê um arquivo-texto, todo o texto do arquivo é interpretado como uma string. Se você ler um número e quiser trabalhar com esse valor em um contexto numérico, será necessário convertê-lo em um inteiro usando a função int() ou convertê-lo em um número de ponto flutuante com a função float().

Um pequeno exercicio

Abra um arquivo em branco em seu editor de texto e escreva algumas linhas sobre Python até agora. Comece cada linha com a expressão "Em python podemos...". Salve o arquivo como "learnin_pyton.txt".

Escreva um programa que leia o arquivo e mostre o que você escreveu, três vezes.

- Exiba o conteúdo uma vez lendo todo o arquivo,
- uma vez percorrendo o objeto arquivo com um laço e
- outra armazenando as linhas em uma lista e então trabalhando com elas fora do bloco with.

Escrevendo dados em um arquivo vazio

Para escrever um texto em um arquivo, chame open() com um argumento que diga a Python que você quer escrever dados no arquivo (*são os modos de leitura, ditos acima*)

```
filename = 'exemplo2.txt'

# w -> dizz a python que queremos abrir em modo de escrita
with open(filename, 'w') as file_object:
  file_object.write("I LOVE PROGRAMMING!")

# A função open() cria automaticamente o arquivo no qual você vai escrever,
# caso ele não exista.
```

- O Método **write()** é utilizado no objeto arquivo para escrever uma string nesse arquivo.

Escrevendo várias linhas

```
with open(filename, 'w') as file_object:
  file_object.write("I LOVE PROGRAMMING!\n")
  file_object.write("I LOVE creating new games. \n")
```

```
with open(filename) as file_object:
  conteudo = file_object.read()
  print(conteudo)
```

```
I LOVE PROGRAMMING!
I LOVE creating new games.
```

Concatenando dados em um arquivo

A maneira que vimos anteriormente faz com que o arquivo será reescrito toda vez que iniciamos o comando. Para isso, precisamos abrir o arquivo em *modo de concatenação*, assim Python não apagará o arquivo antes de devolver o objeto arquivo.

```
with open(filename, 'a') as file_object:
  file_object.write("I also love finding meaning in large datasets.\n")
  file_object.write("I love creating apps that can run in a browser.\n")
```

- O argumento **'a'** utilizamos para abrir o arquivo para concatenação, em vez de sobrescrever o arquivo existente.

```
with open(filename) as file_object:
  conteudo = file_object.read()
  print(conteudo)
```

```
I LOVE PROGRAMMING!
I LOVE creating new games.
I also love finding meaning in large datasets.
I love creating apps that can run in a browser.
```

Dois pequenos exercícios

1. Escreva um programa que pergunte o nome ao usuário. Quando ele responder, escreva o nome em um arquivo chamado *"guest.txt"*.

```
Comece a programar ou gere código com IA.
```

2. Escreva um laço while que pergunte o nome aos usuários. Quando fornecerem seus nomes, apresente uma saudação na tela e acrescente uma linha que registre a visita do usuário em um arquivo chamado *"guest_book.txt"*. Certifique-se de que cada entrada esteja em uma nova linha do arquivo.

```
Comece a programar ou gere código com IA.
```

Exceções com Arquivos (FileNotFoundError)

Um problema comum ao trabalhar com arquivos é o tratamento de arquivos ausentes. O arquivo que você está procurando pode estar outro lugar, o nome do arquivo pode estar escrito de forma incorreta ou o arquivo talvez simplesmente não exista.

Podemos tratar todas essas situações de um modo simples com um bloco **try-except**.

```
filename = 'alice.txt'

with open(filename) as f_obj:
  cont = f_obj.read()

# Vai retornar um erro porque o Python não é capaz de
# ler um arquivo ausente. Por isso, retornará um erro
# chamado "FileNotFoundError"
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-3216047013.py in <cell line: 0>()
      1 filename = 'alice.txt'
      2
----> 3 with open(filename) as f_obj:
      4   cont = f_obj.read()
      5

FileNotFoundError: [Errno 2] No such file or directory: 'alice.txt'
```

```
# Maneira com try-except

filename = 'alice.txt'

try:
  with open(filename) as f_obj:
    cont = f_obj.read()

except FileNotFoundError:
  print("Desculpe, o arquivo não foi encontrado")
```

```
Desculpe, o arquivo não foi encontrado
```
